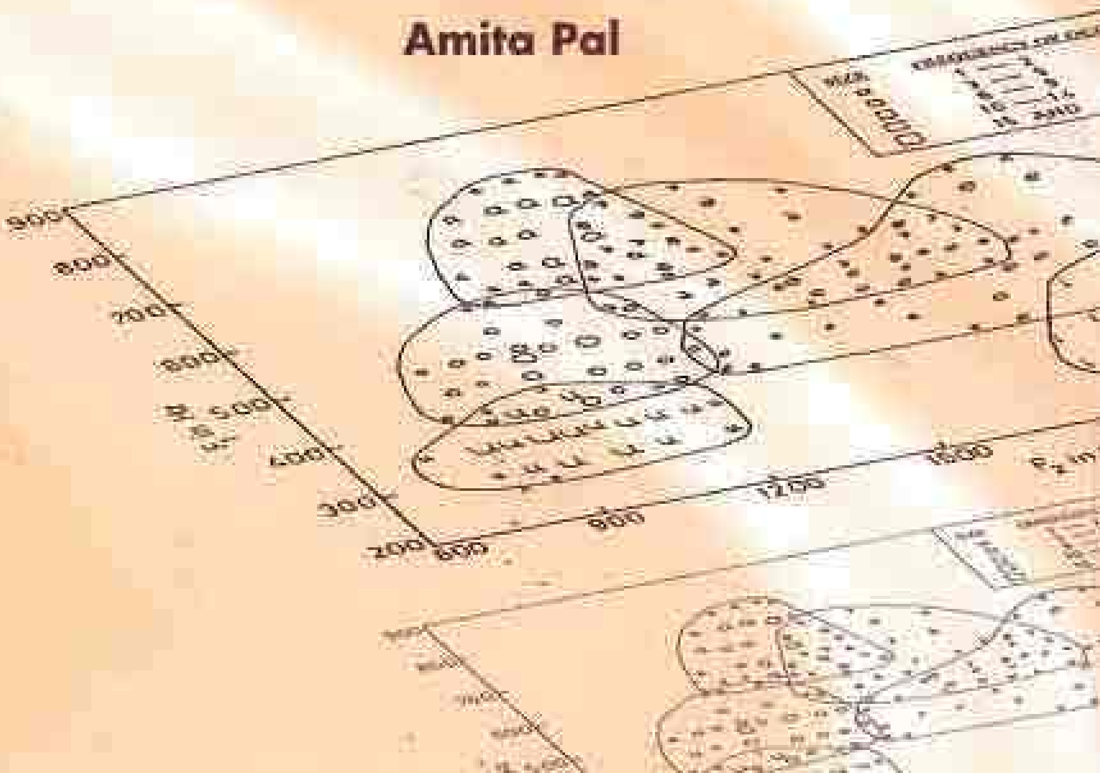
PATTERN RECOGNITION

From Classical to Modern Approaches

Editors

Sankar K. Pal

Amita Pal



World Scientific

PATTERN RECOGNITION
From Classical to Modern Approaches

PATTERN RECOGNITION
From Classical to Modern Approaches

Editors

Sankar K. Pal

Amita Pal

Indian Statistical Institute, Calcutta

World Scientific
New Jersey • London • Singapore • Hong Kong

Published by

World Scientific Publishing Co. Pte. Ltd.

P O Box 128, Farrer Road, Singapore 912805

USA office: Suite 1B, 1060 Main Street, River Edge, NJ 07661

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

PATTERN RECOGNITION
From Classical to Modern Approaches

ISBN 981-02-4684-6

Printed in Singapore.

To our parents
and our children

Foreword

Recognizing patterns is at the core of man's attempts to understand the universe that we inhabit. Through the development of sensors and instruments, we have extended our ability to gather data far beyond the range and scale of man's natural senses. As a result we are now able to gather data on phenomena occurring in the far reaches of the electromagnetic spectrum at scales from the quantum mechanical to the galactic.

In the middle of the last century, the emergence of computer-related technology led to serious starting attempts in automatic pattern recognition, machine intelligence and machine learning. The goal then, and now, is the transformation of raw data into information which helps individuals and groups solve problems, make decisions or take some action. Unlike those earlier times, when researchers faced a paucity of data and small samples for pattern recognition system design, now the converse is often the case; namely, very large datasets gathered in some cases at very high data rates. Thus the current prominence of areas of inquiry and application named data warehousing, data mining and (real-time) content-based classification and retrieval. These areas combine modern data-base technology with feature-extraction, pattern classification, machine learning and related soft-computing methodologies.

In the present volume of lecture notes on pattern recognition, Prof. Sankar Pal and co-editor Dr. Amita Pal have brought together expository, research, and applications articles by recognized experts to give a comprehensive and state-of-the art coverage of classical and modern hybrid methodologies currently being applied in pattern recognition.

The ultimate goal of pattern recognition and data mining is to acquire knowledge or understanding about a domain, and to communicate it so that information may be translated into action by individuals and groups. Knowledge acquisition from databases and humans has been a key problem in the design of AI expert systems and the authoring of content for training and education. The methodologies and thinking have developed separately in those fields without much interaction with pattern recognition and soft computing. The ideas presented in this volume have potential to enhance the possibility of bringing the thinking and methodologies from these disparate fields together to address the automation of domain knowledge acquisition.

I commend the editors and authors for their success in presenting a wealth of knowledge and experience in a form accessible to a wide audience, and highly recommend this volume to students and professionals interested in pattern recognition and its diverse applications.

College Park, MD, USA Laveen N. Kanal
May 2001

Preface

Pattern recognition and machine learning form a major area of research and development that encompasses the processing of pictorial and other non-numerical information obtained from interaction between science, technology and society. A motivation for this spurt of activity in this field is the need for the people to communicate with computing machines in their natural mode of communication. Another important motivation is that scientists are also concerned with the idea of designing and making intelligent machines that can carry out certain tasks as we human beings do. The most salient outcome of these is the concept of future generation computing systems.

The ability to recognize a pattern is an essential requirement for sensory intelligent machines. Pattern recognition is a must component of the so-called "Intelligent Control Systems" which involve processing and fusion of data from different sensors and transducers. It is also a necessary function providing failure detection, verification, and diagnosis tasks. Machine recognition of patterns can be viewed as a two-fold task, consisting of learning the invariant and common properties of a set of samples characterizing a class, and of deciding that a new sample is a possible member of the class by noting that it has properties common to those of the set of samples. Therefore, the task of pattern recognition by a computer can be described as a transformation from the measurement space M to the feature space F and finally to the decision space D.

Though the field has matured enough over the years, it remains evergreen due to the continuous cross-fertilization of ideas from disciplines like computer science, neuro-biology, psychology, physics, engineering, mathematics, statistics and cognitive science. Depending on the need and practical demand, various modern methodologies are being developed which often supplement the classical techniques. Such an evolution ranges from decision theoretic approach (both deterministic and probabilistic), syntactic and structural approach, connectionist approach, fuzzy set theoretic approach to newly-developed soft computing (integration/hybridization of fuzzy sets, artificial neural networks, genetic algorithms and rough sets) approach.

The present volume provides a collection of twenty one articles including an introductory chapter, containing both review and new material, describing, in a unified way, the recent development of various methodologies, ranging from classical to modern, with significant real life applications. These articles, written by different experts over the world, demonstrate the significance of this evolution, relevance of different approaches with characteristic features and the formulation of various modern theories.

The volume starts with an introductory article written by the editors themselves, explaining the basic concepts of pattern recognition, different tasks involved, some conventional classification techniques and the subsequent development of modern methodologies. The significance of data mining, which has recently drawn the attention of many researchers all over the world, is also mentioned. This chapter is included for the convenience of understanding the theory of pattern recognition and hence the remaining chapters of the book.

Chapters 2-5 deal with decision theoretic classification using statistical approach. T. Krishnan considers in Chapter 2 the problem of supervised learning based on the multivariate normal model with a common covariance matrix leading to Fisher's linear discriminant function. The effect of imperfect supervision is studied and the efficiency of learning under two imperfect situations, namely, deterministic but error-prone, and stochastic, is determined as compared to perfect supervision. Chapter 3 of M.A.L. Thathachar and P.S. Sastry addresses some adaptive stochastic algorithms for finding optimal decision rules based on learning automaton models. Various examples are considered to demonstrate the characteristic features of the algorithms. While these chapters consider the problem of supervised learning, Chapter 4 deals with unsupervised learning of classifiers. Here A. Pal, one of the editors, describes some significant recent Bayesian approaches that can be applied to the unsupervised classification problem, if it is formulated as one of decomposition of finite mixtures of probability densities. In Chapter 5, K. V. Mardia focuses on a specific problem of pattern recognition, namely, recognition of shapes (objects) in an image. The author presents an excellent review of the recent advances with emphasis on high level Bayesian image analysis.

Hierarchical pattern classification using decision tree is a simple and powerful approach being used in pattern recognition. In Chapter 6, R. Kothari and M. Dong provide an overview of decision tree induction algorithms and approaches, and describe a new method of decision tree construction based on the concept of "look-ahead". These are given from the perspective that decision trees of small size and depth lead to lesser computational expense in determining classes.

The limitation of the decision theoretic approach lies in its incapability to articulate the interrelationship of the pattern substructures. This has lead to the development of syntactic or structural pattern recognition. Chapter 7 by A.K. Majumdar and A. K. Roy provides a fairly extensive review on such an approach considering its different facets, *e.g.*, primitive selection, pattern grammars, formal linguistic models, grammatical inference, fuzzy syntactic models, formal power series. An application to character recognition is also shown.

When the pattern indeterminacy is due to inherent vagueness rather than randomness, fuzzy set theoretic approach becomes useful. The basic notion that a pattern may have origin from (and hence membership to) more than one class is the key concept behind that approach. Chapter 8 of W. Pedrycz and N. Pizzi presents a motivation for the use of fuzzy sets as a logic canvas for pattern recognition problems with imprecise (vague) information. The authors focus on fuzzy set based generalization of logic processors and petri nets as well as on the context-based partially supervised extensions to the fuzzy c-means clustering. In Chapter 9, M. Grabisch describes the role of fuzzy integrals and fuzzy measures in the development of two new methodologies for pattern recognition.

Artificial neural network based methods have characteristics like adaptivity (to new data or information), speed (via massive parallelism), robustness (to missing, confusing and/or noisy data) and optimality (regarding the error rates in performance). Chapters 10-12 deal with pattern recognition in neural computing paradigm. V. David Sanchez A. describes in Chapter 10, the relationship between traditional methods and neural networks for pattern recognition. Tasks like classification, clustering and regression are considered in this regard. Chapter 11 of N. B. Karayiannis and his colleagues R. Kretzschmar and H. Richner focuses on the development, testing and evaluation of pattern classifiers using quantum neural networks (QNN). It is shown that QNN based classifier can remove upto 90% of bird-contamination in wind profiler data. Chapter 12 concentrates on the application of networks of spiking neurons in data mining problems. Here the authors, K. Cios and D.M. Sala, present some new findings in design and applications of neural networks that use a biologically inspired spiking neuron model. Examples are given to show that such a network emulates several graph algorithms.

Genetic algorithms (GAs), another biologically inspired tool, have recently drawn the attention of the pattern recognition community because of its robust, adaptive and parallel searching capability. Chapter 13 which is from the group of S.K. Pal shows such an attempt in designing a nonparametric classifier. Here Pal and his colleagues, S. Bandyopadhyay and C.A. Murthy, have used the conventional and various enhanced versions of GAs for determining automatically an appropriate number of hyperplanes for approximating any kind of decision boundary with minimum error rate. Based on this principle, an algorithm is then developed for determining optimal architectures of layered networks; thereby demonstrating its significance to the domain of neurocomputing.

Chapter 14 is based on rough set theory which has emerged as another mathematical approach for dealing with imprecise or vague concepts arising mainly from the granularity in the universe of discourse. Here A. Skowron and R. Swiniarski present an application of rough sets for feature selection/extraction, discovery of patterns and their applications for decomposition of large data tables.

The next three chapters deal with synergistic integration of the merits of fuzzy logic, neural networks and genetic algorithms in order to build more efficient recognition systems under soft computing framework. Chapter 15 of L.I. Kuncheva explains how the aforesaid three technologies can be used in classifier combination, which is an established subdiscipline of pattern recognition. The problem of representing complex objects and retrieving "interesting" structures from a large database has been addressed in Chapter 16 by E. Ruspini and I. J. Zwir in the framework of soft computing. It deals with the user-defined qualitative features of the objects rather than the original data. In this regard, some results of a fuzzy clustering algorithm that utilizes evolutionary computation for designing a knowledge discovery system, are presented. Chapter 17 concerns about the neuro-fuzzy approach which is considered to be the most visible hybridization, so far made, in soft computing paradigm. Here S.K. Pal and R.K. De explain how a fuzzy feature evaluation index can be optimized in connectionist framework for feature selection under both supervised and unsupervised modes. The design of the requisite networks is explained. Results are validated with k-NN classifier and scatter plots. In another part, the authors have explained the design procedure of a fuzzy knowledge-based network for pattern classification.

Chapters 18-21 demonstrate various applications. Chapter 18 of N.M. Nasrabadi and his colleagues H. Kwon and S.Z. Der deals with adaptive segmentation (supervised and unsupervised) techniques for hyperspectral imagery. This involves template making technique, Bayes' classifier, quadtree-based segmentation, and sliding window based segmentation. Chapter 19 is concerned with pattern recognition issues in speech processing. Here B. Yegnanarayana and C. Chandra Sekhar provide a nice survey encompassing different pattern recognition methods used, followed by some of the challenging problems. The problem of on-line handwriting recognition is considered in Chapter 20 by S.H. Cha and S.N. Srihari. Some techniques for writing speed and writing sequence invariant recognition are presented using stroke direction sequence string; thereby making the recognizer more robust to noise and allowing more freedom to writers in writing. The last article (Chapter 21) of K. Wang, D. Zhang, N. Li and B. Pang addresses an important problem, namely, tongue diagnosis based on biometric pattern recognition. Biometric technology is applied to Chinese medicine. The methodology involves tongue image capturing and database design, and segmentation, feature extraction and classification of tongue images.

This comprehensive collection provides a cross-sectional view of the advances in pattern recognition comprising the evolution of various methodologies with applications. The book, which in unique in its character, will be useful in a graduate level course as a part of the subject of pattern recognition, machine learning and artificial intelligence, or as a reference book for professionals and researchers in the fields like system design, control, artificial intelligence, soft computing, pattern recognition, data mining and vision.

We take this opportunity to thank all the contributors for agreeing to write for the volume. We owe a vote of thanks to Dr. K.K. Phua and Ms Lakshmi Narayan of World Scientific Publishing Co. for inviting us to write the book. The technical/software assistance provided by Mr. Pabitra Mitra, Sanjoy Das, Mr. Indranil Dutta, Mrs. Maya Dey, Dr. Swati Choudhury and Ms Meenakshi Banerjee during the preparation of the book is gratefully acknowledged.

Calcutta, India
May 2001

Sankar K. Pal
Amita Pal

Contents

Chapter 1

PATTERN RECOGNITION: EVOLUTION OF METHODOLOGIES AND DATA MINING

A. Pal[†] and S. K. Pal[*]

[†]*Applied Statistics Unit*
[*]*Machine Intelligence Unit*
Indian Statistical Institute
Calcutta 700 035, INDIA
e-mail: {*pamita,sankar*}*@isical.ac.in*

Abstract

This chapter provides an introduction to the discipline of pattern recognition (PR), explaining the basic underlying concepts, different tasks involved, some conventional classification techniques and the subsequent development of modern methodologies. It tries to trace the evolution of PR over the years, from its humble beginnings as an extension of statistical discriminant analysis, to the multidisciplinary approach that it has become now, on account of the continuous import of ideas from various scientific disciplines. The evolution has been nurtured and aided by the likes of statistical decision theory (both deterministic and probabilistic), the theory of formal languages (which led to the syntactic or structural approach), followed by the theories of fuzzy sets, artificial neural networks, genetic algorithms and rough sets individually (leading to different modern approaches), and finally, their integration into the newly-developed theory of soft computing. The authors trace the journey of pattern recognition along this complex route, highlighting significant aspects. They also discuss the significance of data mining, which has recently drawn the attention of many PR researchers over the world, in this context.

1.1 Introduction

Pattern recognition is an activity that we humans normally excel in. We do it almost all the time, and without conscious effort. We receive information via our various sense organs, which is processed instantaneously by our brain so that, almost immediately, we are able to identify the source of the information, without having made any perceptible effort. What is even more impressive is the accuracy with which we can perform recognition tasks even under non-ideal conditions, for instance, when the information that needs to be processed is vague, imprecise or even incomplete. In fact, most of our day-to-day activities are based on our success in performing various pattern recognition tasks. For example, when we read a book, we recognize the letters, words and, ultimately, concepts and notions, from the visual signals received by our brain, which processes them speedily and probably does a neurobiological implementation of template-matching!

The discipline of Pattern Recognition (PR) or Pattern Recognition by machine essentially deals with the problem of developing algorithms and methodologies/devices that can enable the computer-implementation of many of the recognition tasks that humans normally perform. The motivation is to perform these tasks more accurately, or faster, and perhaps, more economically than humans and, in many cases, to release them from drudgery resulting from performing routine recognition tasks repetitively and mechanically. The scope of PR also encompasses tasks humans are not good at, like reading bar codes. The goal of pattern recognition research is to devise ways and means of automating certain decision-making processes that lead to classification and recognition. PR has been a thriving field of research for the past few decades, as is amply borne out by the numerous books (References [6, 7, 13, 14, 15, 16, 19, 20, 40, 44, 50, 55, 64, 65], for example) and journals devoted exclusively to it. In this regard, mention must be made of the seminal article by Kanal [30], which gives a comprehensive review of the advances made in the field till the early nineteen-seventies. More recently, a review article by Jain *et al.* [27] provides an engrossing survey of the advances made in statistical pattern recognition till the end of the twentieth century.

Though the subject has attained a very mature level during the past four decades or so, it remains evergreen to the researchers due to continuous cross-fertilization of ideas from disciplines like computer science, physics, neurobiology, psychology, engineering, statistics, mathematics and cognitive science. Depending on the practical need and demand, various modern methodologies have come into being, which often supplement the classical techniques. The present article gives a bird's-eye view of the different methodologies that have evolved so far including the emergence of data mining. Before we describe them, we explain briefly the basic concept of PR including supervised and unsupervised classification, and feature selection/extraction.

1.2 The pattern recognition problem

Let Ω denote the universe of patterns that are of interest, and let $\boldsymbol{X}$ be a vector of p variables (called *features*) defined on objects in Ω, which together provide some sort of numerical description for them. Let $\mathcal{X}$ be the feature space, or the domain of variation of $\boldsymbol{X}$ corresponding to all patterns in Ω, which contains K categories of objects, where K may or may not be known *a priori*. Let $C_1, C_2, \ldots, C_K$ denote the corresponding categories or pattern classes. In this setup, the pattern recognition problem is to determine, for any pattern of unknown categorization (or *label*) from Ω and having a corresponding feature vector x, which pattern class it belongs to. Essentially, the general approach to solving the problem is to find, in some way, a partition of $\mathcal{X}$ into $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_K$, so that if $x \in \mathcal{X}_j$, we can infer that the unknown pattern comes from the pattern class C_j. Obviously, this is not possible unless some additional information is provided, say, in the form of a set of n patterns, called *training samples*.

1.2.1 Supervised vs unsupervised classification

Human pattern recognition capability is mainly learnt from past experiences, though it is certainly not possible to describe the procedure by which the human brain accomplishes this. Thus *learning* is an indispensable component of pattern recognizers, both human and mechanical. The information contained in the training samples provides the basis for learning in pattern recognition systems. In some cases, learning is done with the help of a *teacher*, that is, an external agency of some sort that provides the correct *labels*indextraining samples!labels of or classifications of the training samples provided for building the classifier. The training samples in such cases become representatives of the classes they belong to, and can be processed in a suitable manner so that the class-specific information they carry may be distilled from them. This is referred to as *supervised pattern recognition*. References [15, 16, 20, 59, 60, 62, 64, 65, 66] are a few of the many books in which detailed information on this is available.

On the other hand, if no teacher is available for a pattern classification task, that is, the training samples are not *labeled*, then we have a case of *unsupervised pattern recognition*. In such cases, learning essentially means discovery of the natural groupings inherent in the training set. The generic name for computational techniques applicable to unsupervised classification is *clustering*, for which there is no dearth of literature [1, 6, 7, 8, 25, 28, 63].

1.2.2 Feature selection and extraction

Most of the approaches designed for solving PR problems presuppose the representation of patterns by a set of measurements, called features. A judicious selection of features for building classifiers is a very crucial aspect of classifier design, and deserves careful consideration. On one hand, there is certainly nothing to lose in using all available measurements in classifier design. On the other hand, too many features make the classifier increasingly complex (sometimes confusing too), in fact, unnecessarily so, in case some of the measurements are redundant. It is encouraging to see that this aspect of classifier design has indeed been given the importance it deserves, judging from the work reported. References may be found, for example, in [8, 13, 40, 44, 64, 65]. Two broad approaches have been used traditionally. The first is called *feature selection*, and is essentially the selection of the subset of measurements that optimizes some criterion of separability of classes, since, intuitively, the best set of features should discriminate most efficiently among the classes, that is, enhance the separability among them, while increasing homogeneity within classes at the same time. The other approach, called *feature extraction*, aims to reduce the number of measurements available in a different way by looking for a transformation of the original vector of measurements that optimizes some appropriately defined criterion of separability among classes, possibly leading to fewer features at the same time.

The following sections give a bird's-eye view of the major developments that have taken place as the discipline of pattern recognition evolved gradually to reach the position it occupies at present. On going through them, the reader is sure to get the impression that PR is thriving and growing at a phenomenal pace, not in the least shy in borrowing methodologies from myriad scientific disciplines in order to provide progressively better techniques over the years.

1.3 The statistical approach

When pattern recognition was just beginning to develop as a distinct entity, its practitioners were quick to realize that statistics and probability theory were ideal tools for the task that they had in mind. Statistics could help them to model the inherent variability of patterns in the pattern space via multivariate probability distributions. The classical method of linear discrimination, first proposed by Fisher [18] and later extended by Rao [54] suggested the use of linear combinations of the features, whose coefficients were chosen so as to maximize the ratio of the between-group variance to the within-group variance (the so-called Fisher separability criterion). Such functions, called linear discriminants, could also be constructed using other approaches, like minimum least-squares, linear programming, and so on. A Bayesian decision theoretic approach for building classification rules was seen to provide an elegant solution to the PR problem in the form of the famous Bayes classifier [13, 15, 16, 64]. It was also possible to build Bayes and other classifiers using nonparametric estimates of the probability density functions involved, in case one did not wish to assume unnecessarily restrictive probabilistic models for the classes. All these seemed to work pretty well in the case of supervised classification, and still do for a multitude of problems. For implementing unsupervised classification, too, statistics has been able to provide tools like clustering and estimation of parameters of mixture densities that are used to model the data in such cases.

For a variety of pattern recognition problems, statistical solutions work reasonably well, both for supervised and unsupervised types. However, there are many other scenarios where they fall short of expectations. Over the years, a multitude of alternative techniques have appeared to take care of such problems. The more noteworthy of these are discussed in some of the following sections. Before moving on to them, let us take a quick look at the major contributions of statistics to PR, that have withstood the test of time. A comprehensive survey of statistical PR is provided by Jain *et al.* in [27].

1.3.1 Bayes decision theoretic classification

This is essentially a parametric classification procedure, which means that the class-conditional probability densities of the feature vector are assumed to be of a specific type, say, $f(\boldsymbol{X}|\mathcal{C}_j)$ for the jth class. Moreover, it is Bayesian, which means that an *a priori* probability distribution is assumed for the parameter of interest which, in this case, is the class label. This associates an *a priori* probability π_j with class $\mathcal{C}_j$, $j = 1, 2, \ldots, K$. Reverting to the notation of Section 1.2, we can represent the Bayes classifier for the PR problem mentioned therein, as follows:

> Assign an unknown pattern with feature vector x to class $\mathcal{C}_k$ if
>
> $$p(\mathcal{C}_k|x) = \max_{j=1,2,\ldots,K} p(\mathcal{C}_j|x)$$
>
> where $p(\mathcal{C}_j|x)$ is the *a posteriori* probability for the class $\mathcal{C}_j$, defined as
>
> $$p(\mathcal{C}_j|x) = \frac{f(x|\mathcal{C}_j)\pi_j}{\sum_{r=1}^{K} f(x|\mathcal{C}_r)\pi_r}$$

It has been shown [2, 15, 16, 20, 55, 64] that this rule has the smallest misclassification probability if one assumes a simple zero-one loss function, and the smallest *Bayes risk* (that is, the total expected loss assuming both the feature vector and the class label to be random) if we assumes a more general loss-function, as compared to rival classifiers for the same problem. Even now the performance of the Bayes classifier for a given PR problem serves as a benchmark against which all other classifiers for the same problem are judged.

1.3.2 Discriminant analysis

The simplest example of this is linear discriminant analysis (LDA), which essentially amounts to approximating boundaries between classes by placing hyperplanes *optimally* in the p-dimensional feature space among regions corresponding to the different classes, assuming them to be *linearly separable*. Unknown patterns are classified on the basis of where they lie with respect to the hyperplanes. Since hyperplanes are defined by linear functions of the features (called *linear discriminants* in this context), essentially this amounts to classifying samples on the basis of the linear discriminant function, disjoint sets of values corresponding to different classes. Classifier design amounts to optimal estimation of the coefficient vector for the linear discriminant on the basis of labeled samples. The crudest way of doing this is to use linear programming techniques to solve the inequalities obtained corresponding to the training samples. More sophisticated methods optimize criteria like mean squared error, among others.

The same idea can easily be generalized to the optimal placement of more complex surfaces corresponding to nonlinear discriminant functions.

1.3.3 Nonparametric approach

In situations where there is not enough prior information available to us for making distributional assumptions about the feature vector in the different classes, or we do not wish to make too rigorous assumptions about it, then it is possible to take recourse to *nonparametric* or *distribution-free* methods in statistics to perform the pattern recognition task. The most obvious way of doing this is to implement parametric classifier like the Bayes rule, by replacing the probability densities therein by their nonparametric estimates, and then implementing them. A variety of probability density estimators are available in statistical literature, beginning with simple ones like histogram estimators, followed by window estimators, kernel estimators, and so on [13, 15, 16, 24, 55]. The other approach is to use metric-based classification rules like the k-nearest neighbor rules and minimum distance classifiers [13, 15, 16, 65] that are based on the premise that points in the feature space that are close to each other are very likely to correspond to patterns belonging to the same class. The k-nearest neighbor family of classification rules have certain attractive properties, and certain modifications of it perform almost as well as the Bayes rule.

1.3.4 Clustering

The problem of unsupervised pattern classification may be solved, like problems of various other disciplines, by a method of data analysis known as *clustering*. The objective of cluster analysis is to partition the given data set into a certain number of natural and homogeneous subsets where the elements of each subset are as similar as possible and dissimilar from those of the other subsets. These subsets are called clusters. The number of such sets may be fixed beforehand or may result as a consequence of some constraints imposed on them. Various algorithms have been developed to date to obtain the clusters from a given data sets. Generally they belong to one of two broad categories: partitioning methods and hierarchical methods. A survey of these conventional algorithms is available in many books, for instance, [1, 15, 16, 25, 63, 64, 65].

1.4 The syntactic approach

A significant shortcoming of the statistical and most other subsequent approaches to pattern recognition is that they are ill-equipped to handle contextual or structural information in patterns, that may be very significant in distinguishing among the various pattern classes of interest. In such cases, the pattern can typically be reconstructed from the knowledge of its parts and their interrelationships, and this knowledge may well help in discriminating among pattern classes if the parts and/or interrelationships (that is, the structure) of patterns in one class differs significantly from one class to another. Statistical theory was just not capable to incorporating such complex information as discriminating features. So practitioners of pattern recognition had to search for other approaches which could enable them to model the representation of complex patterns in terms of a recursive decomposition into progressively simpler subpatterns, the interrelationships in the hierarchy of decompositions being well-defined. This was very similar to the problem of decomposing a sentence in some natural language into phrases of different categories which, in turn, are decomposed into words from the vocabulary of the language, using the grammatical rules valid for it (the language). So researchers turned quite naturally to the theory of formal languages [26], and found that much of it was applicable to problems of this type. Hence they used the qualifier *linguistic* or *syntactic* to describe the PR approach that resulted.

Patterns that they were concerned with, were no longer looked upon as arrays of numbers. Rather, they could be described in terms of very simple subelements, called primitives, and certain rules, called syntactical or production rules, governing the relationships among them. The collection of primitives and syntactical rules together formed the pattern grammar that characterized the pattern class and specified as completely as possible the patterns that legitimately constitute the class. In order to use this idea for pattern recognition, there should be one grammar for each pattern class, patterns being represented by strings of primitives. The classification strategy was to infer that a given pattern came from a certain class if it belonged to the language (the set of all possible strings of primitives that can be constructed by the grammar) generated by its grammar. The formal procedure that makes this possible is called syntax analysis or parsing. Of course, there was the very important problem of grammatical inference, namely, constructing the syntax rules for each class on the basis of a set of (supervised) training samples (in the form of strings of primitives) belonging to it. Tailor-made procedures for both parsing and grammatical inference were already a part of formal language theory and could easily be applied. Further, the utility, in the context of PR, of various types of automata that serve as recognizers of specific formal languages, was also self-evident.

The approach worked pretty well for idealized patterns, but was, quite naturally, found wanting when it was applied to real-life problems where patterns tended to be noisy and distorted, and hence more complicated, leading very often to ambiguity, namely, a situation where one string or pattern could be generated by more than one grammar. This led, again quite naturally, to the incorporation of probability theory, and later on, fuzzy-set theoretic concepts to model the randomness and vagueness/ imprecision. The outcome of all this activity has been in the form of various types of stochastic and fuzzy grammars and languages. A comprehensive survey, of these as well as other significant developments in the area, is available in the excellent book by Fu [19]. The approach has great potential, yet has not witnessed much research activity, particularly after the death of Fu, whose contribution to syntactic PR is universally acknowledged.

1.5 Classification trees

Tree-based methods for classification have traditionally been popular in fields like biology and the medical sciences and have the advantage of making the classification procedure easy to comprehend. Classification trees are essentially symbolic systems that assign symbolic decisions to examples and are built upon attributes of the patterns that are symbolic in nature or, at the least, discretized if they are not symbolic to begin with. The terminology associated is graph-theoretic, the root being the top node, and examples are passed down the tree. At each node decisions are made, until a terminal node, or leaf, is reached. A question is associated with each non-terminal node, based on the answer of which a split is made. The label of a classification is contained in each leaf. A classification tree partitions the pattern space (or, equivalently, the feature space) into subregions corresponding to its leaves, and each unknown pattern is classified by the label of the leaf it reaches. It is not difficult to visualize that a classification tree provides a structured or hierarchical description of the knowledge base. The commonest type of tree-structured classifier is the binary tree classifier in which each non-terminal node has exactly two children. Detailed discussion on this particular approach to classification can be found, for example, in the books by Breiman *et al.* [9], Devroye *et al.* [14] and Ripley [55].

The most crucial aspect of tree-based classification is the automatic construction of trees from a set of examples, that is, tree induction. As can be expected, a variety of algorithms are available in literature, one of the better-known ones being the ID3 of Quinlan [53]. The main differences in the algorithms available for tree construction lie in the rule they use for splitting nodes and in the pruning strategy they use, pruning being the term used to describe the removal of redundant subtrees. For both activities, optimality criteria like information gain or entropy are used. For example, when splitting nodes, the attribute that is used to perform the splitting is selected to be the one for which the optimality criterion is optimized. Similarly, when pruning, the subtree whose removal results in the optimization of the criterion being used, is selected. Many of the algorithms available in literature for tree induction are discussed in [9, 14, 55].

Like syntactic PR, tree-based classification too has imbibed concepts and notions from fuzzy set theory, leading to fuzzy decision trees [11, 29] that exploit the flexibility in knowledge representation provided by fuzzy logic, in order to deal with vague or imprecise symbolic information. This has, in turn, led to fuzzy versions of many tree-induction algorithms like the ID3 [8].

1.6 The fuzzy set theoretic approach

In 1965, Zadeh [67] proposed a novel approach to the modeling of vagueness and imprecision by means of *fuzzy sets*, which are generalizations of conventional (crisp) sets. A fuzzy (sub)set of a universe Ω is defined to be a collection of ordered pairs

$$A = \{(\mu_A(x), x) \;\; \forall\, x \in \Omega\},$$

where $\mu_A(x)$, $(0 \leq \mu_A(x) \leq 1)$ gives the degree of belonging of the element x to the set A or the degree of its possession of an imprecise property represented by A. Being a generalization of classical set theory, the theory of fuzzy sets allows for greater flexibility in the representation and processing of imprecise or vague information. Various aspects of fuzzy set theory and its relevance to PR are discussed in [3, 5, 6, 7, 31, 32, 40, 52].

It is well-established by now that fuzzy sets can be applied at the feature level to represent input data as an array of membership values denoting the degrees of possession of certain properties; in representing linguistically phrased input features for their processing; in weakening the strong commitments for extracting ill-defined image regions, properties, primitives, and relations among them; and, at the classification level, for representing class memberships of objects, and for providing an estimate of missing information in terms of membership values. In other words, fuzzy set theory provides a notion of embedding, in the sense that a better solution to a crisp problem can be found by visualizing it initially in a wider space, by subjecting it to different (usually fewer) constraints, thus allowing the algorithm greater freedom to avoid errors that are forced upon it by imposition of hard solutions at intermediate levels.

It is worth noting that fuzzy set theory has led to the development of the concept of soft computing (to be discussed later) as a foundation for the conception and design of a high machine IQ (MIQ) system.

The earliest application [5] of the notion of fuzzy sets to supervised pattern recognition was to visualize pattern classes, particularly overlapping ones, as fuzzy subsets of the pattern space. Classification involved *abstraction*, namely, the estimation of the membership functions characterizing the fuzzy pattern classes from the training samples, and *generalization* , which involves imputation of these estimates in the evaluation of the membership values for each class of unknown patterns. Representation of pattern classes in terms of linguistic features and fuzzy relations is also possible [68, 69]. Decision-theoretic classifiers based on linguistically phrased features have also been proposed [39], as also classification models that incorporate *a priori* knowledge about the classifier from experts in a linguistic form [37]. Pal and Mandal [42] proposed a multivalued approach to supervised classification based on approximate reasoning, that can accept imprecise input in linguistic form and provide output in multiple states.

Fuzzy versions of many crisp notions and procedures have been developed, *e.g.*, the k-NN rule, decision trees, phrase structure grammars, tree grammars, and so on, which lead to fuzzy forms of classical supervised classification methods. The corresponding references may be found in, for example, [7, 44].

Fuzzy sets have also been used in knowledge-based (KB) approaches to PR that emerged in the beginning of the eighties. In the KB approach, classes are described by rules and the task of recognition is accomplished through automated reasoning or inference procedure. Fuzzy logic can be implemented for describing rules in natural linguistic terms and in fuzzy inferencing with a degree of certainty.

Decidedly more activity has been seen in the area of clustering based on fuzzy set theory, as is borne out by literature [3, 6, 7, 8]. In all classical clustering algorithms, it is implicitly assumed that disjoint clusters exist in the set of data while in practice, in many cases, the clusters are not completely disjoint; rather, the separation of clusters is a fuzzy notion. The concept of fuzzy subsets offers special advantage over conventional clustering and allows representation of intractable overlapping configurations of pattern classes. In fuzzy clustering each element is assigned a finite membership to each of the clusters.

The problem of fuzzy clustering was first posed by Ruspini [58] who introduced the notion of a fuzzy partition to represent the clusters in a data set. Zadeh [70] proposed the fuzzy affinity property in order to characterize fuzzy clusters induced by a fuzzy relation R. A new direction in the application of fuzzy set theory to cluster analysis was initiated by Bezdek and Dunn in their work on fuzzy ISODATA [6, 17]. A fuzzy version of the clustering algorithm DYNOC has also been proposed by Pal and Mitra [43]. Backer [3] introduced a clustering model which tries to achieve an optimal decomposition of a group of objects into a collection of induced fuzzy sets by means of a point-to-subset affinity concept, based on their structural properties, among objects in the representation space. A broad overview of the state of the art, as far as fuzzy clustering is concerned, can be had from [7, 8].

1.7 The connectionist approach

The normal human brain performs countless complex cognitive (and other) tasks effortlessly, accurately and instantaneously by virtue of its architecture, namely, a massively parallel network of biological neurons (or nerve cells), that are basically extremely simple information processing units. Its information processing capability is so robust that the death or malfunctioning of a few of its component neurons generally does not affect its performance perceptibly. Impressed by the efficiency and fault-tolerance of this type of architecture, researchers have tried, over the years, to model information processing systems on it. The models that have emerged from their endeavors, albeit much simpler versions, have succeeded resoundingly in solving a variety of problems, and are called Artificial Neural Networks (ANNs) . Basically, each such network consists of numerous densely interconnected simple processing units (or neurons, that is naturally capable of storing prior knowledge and making it available as and when needed. It resembles the human brain in that it acquires knowledge through a learning process and stores the same in a distributed fashion as synaptic weights in the interconnections of the neurons. Numerous books are available on the theory and applications of ANNs [23, 33, 34, 56, 57], and a number of journals devoted exclusively to various aspects of this relatively new and revolutionary information processing system are in circulation since the early nineties.

Various categories of artificial neural networks (ANNs) have made their appearance over the years, for example, Hopfield networks, Multi-Layer Perceptrons (MLPs), Kohonen self-organizing feature maps (SOFM), Adaptive Resonance Theory (ART) networks, Radial Basis Function (RBF) networks, among others. Some of the tasks that ANNs can perform include supervised pattern classification, clustering, function approximation and optimization. In the context of pattern recognition, it has been established beyond doubt that neural networks are natural classifiers having resistance to noise, tolerance to distorted images or patterns (ability to generalize), superior ability to recognize partially occluded or degraded images or to discriminate among overlapping pattern classes or classes with highly nonlinear boundaries, or potential for parallel processing.

ANNs can be viewed as weighted directed graphs in which neurons for nodes and the inter-neuron connections as directed edges. They can be broadly grouped into two categories on the basis of their architectures – *feedforward* and *feedback* or *recurrent* networks. The former are characterized by graphs with no loops, unlike the latter, which have loops on account of feedback connections. Their adaptability stems from their capacity for learning from their "environments". There are three broad paradigms of learning – supervised, unsupervised and reinforcement. Various learning algorithms are available in each category. In supervised learning, adaptation is done on the basis of a direct comparison of the network output with the correct or desired label. In unsupervised learning, the network is designed to detect natural groupings in the training set, and forms categories by optimization of some criterion for the quality of clustering induced by the network. Reinforcement learning is looked upon as a special type of supervised learning that tries to learn the input-output relation by optimizing the so-called *reinforcement signal*. It makes the network aware of the correctness of the decision, but not what the actual decision is. MLPs that learn by *backpropagation of error* [57] have become extremely popular for solving supervised PR problems. On the other hand, the Kohonen SOFMs [33, 34] perform unsupervised classification. *Learning* *vector quantization* is a special case of SOFM learning, both being particular instances of *competitive learning*, in which the input data is replaced by a much smaller set of prototypes that are good representatives of structure in the data for classifier design. Hectic research activity has been seen since the eighties in the area of pattern recognition by ANNs, and a good overview can be obtained from [44, 48, 50, 55, 59].

1.8 Use of genetic algorithms

Genetic algorithms (GAs) [12, 21, 22, 36, 49] are randomized search and optimization techniques, inspired by the principle of *survival of the fittest* governing evolution and selection in natural populations, and are therefore regulated by the laws of genetics. They are capable of obtaining near-optimal solutions by performing efficient, adaptive and robust search, and have parallelism as a major ingredient. In order to approach an optimal solution to a computational problem, a GA starts from a set of assumed solutions (likened to and called chromosomes) and evolves different yet better sets of solutions over a sequence of iterations, called generations. In each generation, the objective function (a measure of fitness for survival) determines the suitability of each solution and, on the basis of its values, some of them (called parent chromosomes) are selected for reproduction. Genetic operators like *selection/reproduction*, *crossover* and *mutation* are applied on these and new chromosomes (offsprings) are generated. The number of copies (offsprings) reproduced by an individual parent is expected to be directly proportional to its fitness value. Chromosomes with higher fitness values thus tend to have greater representation in the next generation. Ultimately, after many generations, only those chromosomes with very high fitness values (corresponding to the optimal solution) proliferate.

Like many other scientific and technical fields of research and development, most approaches for pattern recognition involve the solution of optimization problems of one type of another, thereby making the output dependent on appropriate selection of some parameters. For example, unsupervised classification by clustering involves the optimization of certain objective functions depending upon some parameter values, and so does the induction of tree-structured classifiers. Therefore GA becomes appropriate and a natural choice for solving many of these problems robustly, and speedily with no fear of getting trapped at local optima. Based on this realization, many researchers have been concentrating on developing GA-based PR methods in the past decade.

The book by Pal and Wang [49] gives an interesting cross-section of PR-related problems that have been successfully solved by the application of GAs.

1.9 The hybrid approach and soft computing

Integration and further development, in the context of PR, of the aforesaid approaches have been observed in recent years. Various hybrid methods have been developed by taking into account the merits of the constituent technologies. Recently, a consolidated effort is being made to integrate mainly fuzzy logic, artificial neural networks and genetic algorithms, for developing an efficient new paradigm called *soft computing,* that endows, to an information processing system, the sort of flexibility that is required for handling real-life ambiguous, imprecise situations. The objective is to develop computational paradigms that provide reasonably good solutions at low cost under non-ideal conditions, and bear close resemblance to human-like decision-making. The salient features of each component have been highlighted in preceding sections. Obviously, a combination of two or more of these methodologies imparts the benefits of each to the hybrid system, leading to more robust solutions. It is no wonder, therefore, that rapid strides are being made in the hybridization of methodologies in the context of pattern recognition. Among the various possibilities, neuro-fuzzy integration [10, 41, 44] is the most visible one. Fig. 1.1 shows the integration of such fuzzy-neural networks with GAs and Rough Sets. Here the layered network can accept input in fuzzy terms and provides fuzzy output, thereby augmenting its application domain. GAs are used to tune the input membership functions and output decision boundaries, and to determine optimum network parameters. Rough set theory [51], which is being considered as another new tool for handling uncertainty, is used for extracting domain knowledge from training samples in the form of crude rules and in encoding initial network parameters for its faster learning [4, 45, 46, 47]. A good idea of current developments in soft computing approaches to PR can be obtained from recent issues of most PR journals, and also from [41, 44, 45, 46, 47, 49].

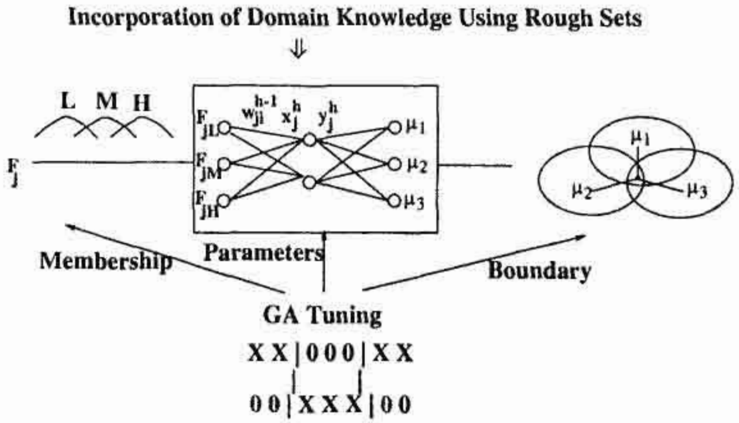


Fig. 1.1 Integration of fuzzy neural networks with GAs and rough sets

1.10 Data mining and knowledge discovery

In recent years, the rapid advances being made in computer technology have ensured that large sections of the world population have been able to gain easy access to computers on account of falling costs worldwide, and their use is now commonplace in all walks of life. Government agencies, scientific, business and commercial organizations are routinely using computers not just for computational purposes but also for storage, in massive databases, of the immense volumes of data that they routinely generate, or require from other sources. Large-scale computer networking has ensured that such data has become accessible to more and more people. In other words, we are in the midst of an information explosion, and there is urgent need for methodologies that will help us bring some semblance of order into the phenomenal volumes of data that can readily be accessed by us with a few clicks of the keys of our computer keyboard. Traditional statistical data summarization and database management techniques are just not adequate for handling data on this scale, and for extracting intelligently, information or, rather, knowledge that may be useful for exploring the domain in question or the phenomena responsible for the data, and providing support to decision-making processes. This quest had thrown up some new phrases, for example, *data mining* [61] and *knowledge discovery in databases (KDD)*, which are perhaps self-explanatory, but will be briefly discussed in the next few paragraphs. Their relationship with the discipline of pattern recognition will also be examined.

The massive databases that we are talking about are generally characterized by the presence of not just numeric, but also textual, symbolic, pictorial and aural data. They may contain redundancy, errors, imprecision, and so on. KDD is aimed at discovering natural structures within such massive and often heterogeneous data. Therefore PR plays a significant role in KDD process. However, KDD is being visualized as not just being capable of knowledge discovery using generalizations and magnifications of existing and new pattern recognition algorithms, but also the adaptation of these algorithms to enable them to process such data, the storage and accessing of the data, its preprocessing and cleaning, interpretation, visualization and application of the results, and the modeling and support of the overall human-machine interaction. What really makes KDD feasible today and in the future is the rapidly falling cost of computation, and the simultaneous increase in computational power, which together make possible the routine implementation of sophisticated, robust and efficient methodologies hitherto thought to be too computation-intensive to be useful. A block diagram of KDD is given in Figure 1.2.

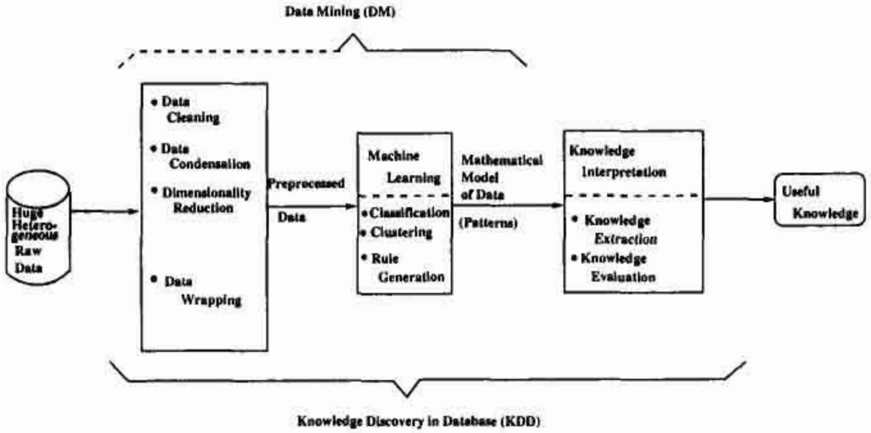


Fig. 1.2 Block diagram for Knowledge Discovery in Databases (KDD)

Data mining is that part of knowledge discovery which deals with the process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data, and excludes the knowledge interpretation part of KDD.

Therefore, as it stands now, data mining can be viewed as applying PR and machine learning principles in the context of voluminous, possibly heterogeneous data sets. Furthermore, soft computing-based (involving fuzzy sets, neural networks, genetic algorithms and rough sets) PR methodologies and machine learning techniques seem to hold great promise for data mining. The motivation for this is provided by their ability to handle imprecision, vagueness, uncertainty, approximate reasoning and partial truth and lead to tractability, robustness and low-cost solutions. In this context, case-based reasoning [35], which is a novel Artificial Intelligence (AI) problem-solving paradigm, has a significant role to play, as is evident from the recent book edited by Pal, Dillon and Yeung [38]. Some of the challenges that researchers in this area are likely to deal with, include those posed by massive data sets and high dimensionality, nonstandard and incomplete data, and overfitting. The focus is most likely to be on aspects like user interaction, use of prior knowledge, assessment of statistical significance, learning from mixed media data, management of changing data and knowledge, integration of tools, ways of making knowledge discovery more understandable to humans by using rules, visualization, and so on.

1.11 Conclusions

We have discussed about the various approaches to PR that have emerged so far over the past four decades or so. These range from the statistical, the syntactic, the classification-tree-based, the fuzzy-set-theoretic, the connectionist, the evolutionary-computation-based, to newly-emerged soft computing (hybrid) techniques. Their relevance and salient features are highlighted. Finally, data mining and knowledge discovery in databases, which has recently drawn the attention of researchers significantly, have been explained from the view-point of PR. As it stands, soft computing methodologies, coupled with case-based reasoning and the computational theory of perception (CTP) [71], have great promise for efficient mining of large, heterogeneous data and solution of real-life recognition problems. We believe the next decade will bear testimony to this.

References

[1] M. R. Anderberg, *Cluster Analysis for Applications.* New York: Academic Press, 1973.

[2] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis.* New York: Wiley, 2nd ed., 1984.

[3] E. Backer, *Cluster Analysis by Optimal Decomposition of Induced Fuzzy Sets.* Delft, The Netherlands: Delft University Press, 1978.

[4] M. Banerjee, S. Mitra, and S. K. Pal, "Rough-fuzzy MLP: knowledge encoding and classification," *IEEE Transactions on Neural Networks,* vol. 9, pp. 1203–1216, 1998.

[5] R. Bellman, R. Kalaba, and L. A. Zadeh, "Abstraction and pattern classification," *Journal of Mathematical Analysis Applications,* vol. 13, pp. 1–7, 1966.

[6] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Functions.* New York: Plenum Press, 1981.

[7] J. C. Bezdek and S. K. Pal, eds., *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data.* New York: IEEE Press, 1992.

[8] J. C. Bezdek, J. Keller, R. Krisnapuram, and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing.* Boston: Kluwer Academic, 1999.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees.* Monterey, CA: Wadsworth and Brooks, 1984.

[10] H. Bunke and A. Kandel, eds., *Neuro-fuzzy Pattern Recognition.* Singapore: World Scientific, 2000.

[11] R. L. P. Chang and T. Pavlidis, "Fuzzy decision tree algorithms," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 7, pp. 28–35, 1977.

[12] L. Davis, ed., *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.

[13] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach.* London: Prentice-Hall, 1982.

[14] L. Devroye, L. Gyorfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition.* New York: Springer, 1996.

[15] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[16] R. O. Duda, D. G. Stork, and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons, second ed., 2000.

[17] J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, pp. 32–57, 1973.

[18] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

[19] K. S. Fu, *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[20] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Morgan Kaufmann, 1990.

[21] E. S. Gelsema, ed., "Special issue on genetic algorithms," *Pattern Recognition Letters*, vol. 16, no. 8, 1995.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[23] S. Grossberg, ed., *Neural Networks and Natural Intelligence*. Cambridge, Massachusetts: The MIT Press, 1988.

[24] D. J. Hand, *Discrimination and Classification*. Chichester: John Wiley, 1981.

[25] J. A. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.

[26] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Reading, MA: Addison-Wesley, 1979.

[27] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1–37, 2000.

[28] A. K. Jain and R. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[29] C. Z. Janikow, "Fuzzy decision trees: issues and methods," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 28, pp. 1–14, 1998.

[30] L. N. Kanal, "Patterns in pattern recognition:1968–1974," *IEEE Transactions on Information Theory*, vol. IT-20, pp. 697–722, 1974.

[31] A. Kandel, *Fuzzy Mathematical Techniques with Applications*. Reading, MA: Addison-Wesley, 1986.

[32] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic– Theory and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[33] T. Kohonen, *Self-Organization and Associative Memory*. 3rd ed.,

Berlin: Springer, 1989. (First edition, 1984.)

[34] T. Kohonen, *Self-organizing Maps.* Berlin: Springer, 1995.

[35] J. L. Kolodner, *Case-Based Reasoning.* San Mateo, CA: Morgan Kaufmann, 1993.

[36] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA: The MIT Press, 1996.

[37] A. K. Nath, S. W. Liu, and T. T. Lee, "On some properties of linguistic classifier," *Fuzzy Sets and Systems,* vol. 17, pp. 297–311, 1985.

[38] S. K. Pal, T. S. Dillon, and D. S. Yeung, eds., *Soft Computing in Case Based Reasoning.* London: Springer, 2001.

[39] S. K. Pal and D. Dutta Majumder, "Fuzzy sets and decision making approaches in pattern recognition," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 7, pp. 625–629, 1977.

[40] S. K. Pal and D. Dutta Majumder, *Fuzzy Mathematical Approach to Pattern Recognition.* New York: Wiley (Halsted Press), 1986.

[41] S. K. Pal, A. Ghosh, and M. K. Kandu, eds., *Soft Computing for Image Processing.* Heidelberg: Physica-Verlag, 2000.

[42] S. K. Pal and D. P. Mandal, "Linguistic recognition system based on approximate reasoning," *Information Sciences,* vol. 61, pp. 135–162, 1992.

[43] S. K. Pal and S. Mitra, "Fuzzy dynamic clustering algorithm," *Pattern Recognition Letters,* vol. 11, pp. 525–535, 1990.

[44] S. K. Pal and S. Mitra, *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing.* New York: John Wiley, 1999.

[45] S. K. Pal, W. Pedrycz, R. Swiniarski, and A. Skowron, eds., "Special issue on Rough-Neuro Computing," *Neurocomputing,* vol. 36, no. 124, 2001.

[46] S. K. Pal and A. Skowron, eds., *Rough–Fuzzy Hybridization: A New Trend in Decision Making.* Singapore: Springer-Verlag, 1999.

[47] S. K. Pal and A. Skowron, eds., "Special issue on Rough Sets, Pattern Recognition and Data Mining," *Pattern Recognition Letters,* 2001. (to appear).

[48] S. K. Pal and P. K. Srimani, eds., "Special issue on Neural Networks: theory and applications," *IEEE Computer,* vol. 19, no. 3, 1996.

[49] S. K. Pal and P. P. Wang, eds., *Genetic Algorithms for Pattern Recognition.* Boca Raton, FL: CRC Press, 1996.

[50] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks.* Read-

ing, MA: Addison Wesley, 1989.

[51] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data.* Dordrecht: Kluwer Academic, 1991.

[52] W. Pedrycz, "Fuzzy sets in pattern recognition," *Pattern Recognition*, vol. 23, pp. 121–146, 1990.

[53] J. R. Quinlan, "Induction of decision trees," in *Machine Learning*, pp. 81–106, Boston: Kluwer Academic, 1986.

[54] C. R. Rao, "The utilization of multiple measurements in problems of biological classification," *Journal of the Royal Statistical Society Series B*, vol. 10, pp. 159–203, 1948.

[55] B. D. Ripley, *Pattern Recognition and Neural Networks.* Cambridge: Cambridge University Press, 1996.

[56] D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol. I. Cambridge, Massachusetts: The MIT Press, 1986.

[57] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition* (D. E. Rumelhart and J. L. McClelland, eds.), vol. I, Cambridge, Massachusetts: The MIT Press, 1986.

[58] E. H. Ruspini, "A new approach to clustering," *Information and Control*, vol. 15, pp. 22–32, 1969.

[59] R. Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches.* New York: John Wiley and Sons, 1992.

[60] G. S. Sebestyen, *Decision-Making Processes in Pattern Recognition.* New York: McMillan, 1962.

[61] J. G. Shanahan, *Soft Computing for Knowledge Discovery: Introducing Cartesian Granule Features.* Boston, MA: Kluwer Academic, 2000.

[62] J. Sklansky and G. N. Wassel, *Pattern Classifiers and Trainable Machines.* New York: Springer-Verlag, 1981.

[63] P. H. A. Sneath and R. Sokal, *Numerical Taxonomy.* San Fransisco: Freeman, 1973.

[64] S. Theodoridis and K. Koutroumbas, *Pattern Recognition.* San Diego: Academic Press, 1999.

[65] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles.* New York: Addison–Wesley, 1974.

[66] T. Y. Young and T. W. Calvert, *Classification Estimation and Pattern Recognition.* New York: Elsevier, 1974.

[67] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[68] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 3, pp. 28–44, 1973.

[69] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning: Parts 1, 2 and 3," *Information Sciences*, vol. 8, 8, 9, pp. 199–249, 301–357, 43–80, 1975.

[70] L. A. Zadeh, "Fuzzy sets and their application to pattern classification and cluster analysis." Memo no. UCB/ERL M-607, University of California, Berkeley, 1976.

[71] L. A. Zadeh, "A new direction in AI: toward a computational theory of perceptions," *AI Magazine*, vol. 22, pp. 73–84, 2001.

Chapter 2

IMPERFECT SUPERVISION IN STATISTICAL PATTERN RECOGNITION

T. Krishnan

Curtin University of Technology
GPO Box U1987
Perth 6845, AUSTRALIA
e-mail: *krishnat@cs.curtin.edu.au*
and
SQC & OR Unit
Indian Statistical Institute
Chennai 600 029, INDIA
e-mail: *krishnant1@yahoo.com*

Abstract

We consider the two-class statistical pattern recognition problem based on the multivariate normal model with a common covariance matrix leading to Fisher's linear discriminant function. We consider mainly two situations of imperfect supervision, namely, deterministic but error-prone, and stochastic. We work out the efficiency of learning in these situations *vis a vis* perfect supervision, in terms of the number of sample units required to attain the same learning efficiency as given by, say one perfectly supervised sample unit, for various values of the multivariate normal parameters and the imperfect supervision parameters.

2.1 Statistical pattern recognition

2.1.1 Supervised learning

We consider a p-dimensional continuous feature vector x and a two-class pattern recognition problem. Suppose we denote the two classes by C_0 and C_1 and the *class-conditional densities* by $f(x \mid 0)$ and $f(x \mid 1)$ respectively. Then the *Bayes classification rule* is the following:

$$\begin{array}{l} \text{Classify into } C_0 \text{ if } \log \pi_0 + \log f(x \mid 0) > \log \pi_1 + \log f(x \mid 1); \\ \text{Classify into } C_1 \text{ if } \log \pi_0 + \log f(x \mid 0) \leq \log \pi_1 + \log f(x \mid 1) \end{array} \tag{2.1}$$

where π_0 and π_1 $(\pi_0 + \pi_1 = 1)$ are the *prior* probabilities of the classes C_0 and C_1 respectively. However, in the absence of knowledge of $\pi_i, f(x \mid i)$, $i = 0, 1$, a sample-based formula needs to be developed, based on estimates of these quantities. The development of such a sample-based formula is facilitated by an initial sample, called *training sample*. If the elements of the training sample are classified (presumed correctly) into one of the two classes, then the problem of estimation (learning) is relatively easy and this situation is called *supervised learning*. Even if the elements of the training sample are not classified, learning is possible if the class-conditional densities belong to certain families of distributions, such as *multivariate normal (multinormal)* distributions; such learning is called *unsupervised learning*. There are basically two approaches to this estimation—parametric and nonparametric. The reader is referred to standard books such as Devijver and Kittler [13] Fukunaga [17], McLachlan [36] and Watanabe [50], on the subject, for details of the parametric approach and a description of other approaches. If a supervised training sample is available and the supervisor classification is correct, then let us call the situation *perfectly supervised learning*.

2.1.2 Unreliable supervision

However, in many applications of pattern recognition, classifying a training sample is expensive and difficult and is subject to error; some examples of this situation are remote sensing [4, 5, 6, 7] and medical diagnosis [28, 29, 47]. In a problem of remote sensing of crop discrimination using spectral data from a satellite, the training sample may be visually classified by an analyzer-interpreter who examines imagery film and uses such auxiliary information as historical information, cropping practices, crop calendar models, *etc.* This procedure is not foolproof and may lead to a certain amount of mislabeling. Lack of adequate spectral separation between classes can also lead to mislabeling. Further, the area corresponding to a training sample unit may have mixed crops and the analyzer-interpreter may classify the unit into the majority-area crop by a visual estimate. These classifications are prone to error and even ambiguity of definition of class.

In a medical diagnosis situation, the expert may classify a doubtful case in the training sample into the most probable type rather than reject it, in view of paucity of samples and to satisfy conditions of deterministic initial classification [2]. We call this type of deterministic and error-prone supervision *unreliable supervision*. Moreover, in a medical diagnosis problem, the training sample may be classified by experts on the basis of the same feature vector x as the one used for learning and hence may be prone to error. In the first case, the errors in supervision may be presumed to be independent of the feature vector x and in the second dependent on x. These two situations of unreliable supervision are called *random misallocation* and *nonrandom misallocation* [6, 28, 29] of training samples.

2.1.3 Stochastic supervision

In the foregoing situation, the supervisor classification is deterministic, that is, each training sample unit is classified by the supervisor as belonging to one or other of the classes, although prone to error. However, the supervisor may not always be inclined to classify deterministically, because they may have used a statistical mechanism to obtain their classification or that there is a genuine doubt as to the class to which the sample unit belongs, which they may be in a position to quantify. The first kind of situation may arise when the supervisor uses a discriminant function or a diagnostic formula of the type arising from standard supervised learning, on the basis of features which are different from the ones for which the current formula is to be developed. One can see heuristically that such a supervisor classification is useful as long as the relationship between the features they use and the classes is 'stronger' than the relationship between the current features and the classes; for instance, in a medical diagnosis situation the 'stronger' features may have been observed at the time of autopsy and the 'weaker' features are the ones to be used for building a diagnostic formula on living patients. The second kind of situation may arise when the information on which the supervisor bases their classification is ambiguous. For instance, in a remote sensing application of crop discrimination, the pixel concerned may correspond to an area of mixed cropping; the supervisor may like to give a percentage distribution of crops rather than a single crop. In both kinds of situations above, it seems sensible to use the supervisor's assessment in the form of a (probability) distribution over the classes rather than a deterministic assessment. We call this *stochastic supervision.* We consider unreliable supervision and stochastic supervision as two types of *imperfect supervision.*

2.1.4 Imperfect supervision

Starting with (perfectly) supervised learning, we moved on to a situation where the supervision is imperfect. The question naturally arises as to whether learning can be done under imperfect supervision. A related and somewhat more general question is whether learning can be done without supervision. The answer to this question is: 'Yes, under certain circumstances'. In Section 2.3.1, we discuss briefly this question of unsupervised learning. If it is possible to learn without supervision, then it seems possible to learn with imperfect supervision, at least under those circumstances where unsupervised learning is possible. As we noted earlier, unsupervised learning is possible for the two-group multinormal case with a common covariance matrix and so we consider learning under imperfect supervision for this case; in this article we consider mainly this case.

Thus at one end of the spectrum of learning schemes, there is (perfectly) supervised learning and at the other end there is unsupervised learning. Imperfect supervision is in between. Considering unreliable supervision, most models for supervision errors will lead to perfect supervision as one extreme case and lack of supervision (we shall henceforth call it *nonsupervision*, for the sake of convenience) as the other extreme. For instance, if we consider a two-class situation (C_0 and C_1) and a random misallocation model where the probability of supervisor misallocation is α (not depending on the actual class or $\boldsymbol{x}$), then $\alpha = 0$ is the case of perfect supervision and $\alpha = \frac{1}{2}$ is the case of nonsupervision. The second part of this statement can be heuristically seen to be true, since in the case of supervisor allocation error being $\frac{1}{2}$, the allocation is 'purely random' and is uninformative. We shall see this more formally subsequently. Similarly, in the case of stochastic supervision, assigning equal probability to all the classes will be nonsupervision and assigning probability 1 to the correct class is (perfect) supervision.

In many situations, supervision is expensive, whereas obtaining an unsupervised observation is relatively inexpensive. Since both types of supervision provide information, it may be a useful idea to combine both types of training samples. In a similar manner, it may be worthwhile combining stochastic supervision with perfect supervision . However, in the literature, only the combination of perfectly supervised and unsupervised learning schemes has been considered in detail. We call this *combined learning*, although, strictly speaking it can be applied to any combination of learning schemes.

2.1.5 Questions on imperfect supervision

Before proceeding to deal with learning problems under imperfect supervision, it is first necessary to understand the consequences of imperfection in the case of learning under a presumed perfect supervisor. Such investigations indeed need an understanding of the mechanism of supervisor classification. Thus realistic and tractable models for unreliable supervision and stochastic supervision are first needed. In Section 2.4, we discuss these models. In Section 2.5, we discuss the effects of supervisor misallocationon the allocation rules derived from the assumption of perfect supervision. From these studies, it seems to follow that supervisor imperfection cannot always be ignored and that a methodology is required for learning under unreliable supervision. The case of stochastic supervision cannot be dealt with by supervised learning methods and a different methodology is needed for this case. In Section 2.6, we discuss learning under unreliable supervision and in Section 2.7, we discuss learning with stochastic supervision.

Evidently, a perfectly supervised training sample unit is more valuable for learning than an imperfectly supervised unit. As pointed out already, perfect labeling is very expensive. Since decisions may have to be made regarding the design of supervision, it is necessary to know what the worth is of supervision carried out according to a certain scheme. Thus the question arises as to how much information is contained in a training sample obtained according to a certain supervision scheme, relative to say, perfect supervision. That is, what are the relative efficiencies of the various supervision schemes? To put it differently, how many specimens are required for a certain imperfect supervision scheme to obtain the same level of efficiency as a perfectly supervised scheme with, say 100 specimens? The answer to this question depends on many factors. Firstly, it depends upon how the training sample is to be used for learning. We shall mostly be concerned with parametric learning for the two-class multinormal case with a common covariance matrix. In that case, learning means the estimation of the *linear discriminant function*. We shall, in general, be dealing with *maximum likelihood estimates* of the linear discriminant function and efficiency in this context relates to the error rate of the estimated linear discriminant function. Secondly, this efficiency depends upon the model for imperfect supervision and the parameters in the model. Thirdly, the efficiency depends upon the parameters of the models used for class-conditional distributions as well as the mixture proportions of the classes. In Sections 2.6 and 2.7, we discuss the efficiencies of the various imperfect learning schemes under various models, derive formulae for these efficiencies and present tables of numerical values of these efficiencies for a reasonable range of parameter values. These then give us an understanding of the use and implications of statistical pattern recognition under imperfect supervision.

We must point out here that the terminology relating to imperfect supervision is not in a standardized form and different authors use different terms to mean the same concept and sometimes use the same term to mean different concepts. There is generally no confusion, except in a few situations like as follows: In the article of Imai and Shimura [20], the term probabilistic labeling is used in a way different from ours. They are considering an unsupervised learning algorithm in which one of the steps is to allocate a specimen probabilistically to a class. This allocation is not the supervisor allocation that we are talking about. In this article, we have attempted to use a set of terms consistently according to the formulation in the paragraphs above and have also applied it to the work of other authors.

2.2 Preliminaries

2.2.1 Multivariate normal model and Fisher's linear discriminant function

In the parametric approach, the class-conditional densities of the p-dimensional feature column vector x, $f(x \mid 0)$ and $f(x \mid 1)$ are modeled. We describe here one of the basic models—the homoscedastic normal model— which has been extensively studied and used, and about which this chapter is mostly concerned. Here the class-conditional densities in the two classes are modeled as *multinormal* densities with a common (nonsingular) covariance matrix Σ and mean vectors $\boldsymbol{\mu}_0$ and $\boldsymbol{\mu}_1$ respectively:

$$f(x \mid i) = 2\pi^{-p/2}|\Sigma|^{-1/2}exp[-\frac{1}{2}(x - \boldsymbol{\mu}_i)'\Sigma^{-1}(x - \boldsymbol{\mu}_i)], \quad i = 1,0.$$

Then it is easily seen that the Bayes classification rule (2.1) is obtained as follows (see, for instance [3, 36]): Let

$$\begin{aligned} \beta_0 &= \log \tfrac{\pi_1}{\pi_0} - \tfrac{1}{2}(\mu_1'\Sigma^{-1}\mu_1 - \mu_0'\Sigma^{-1}\mu_0) \\ \beta' &= (\mu_1 - \mu_0)'\Sigma^{-1} \\ \Lambda(x) &= \beta_0 + \beta' x. \end{aligned} \tag{2.2}$$

Then

$$\begin{aligned} &\text{Classify into } C_0 \text{ if } \Lambda(x) < 0; \\ &\text{Classify into } C_1 \text{ if } \Lambda(x) \geq 0. \end{aligned} \tag{2.3}$$

This is the well-known *Linear Discriminant Function*. The class-conditional error rate when the true class is C_0 is

$$\Phi\{(\lambda - \frac{1}{2}\Delta^2)/\Delta\}$$

where

$$\lambda = \log \frac{\pi_1}{\pi_0}$$

and

$$\Delta^2 = (\mu_1 - \mu_0)'\Sigma^{-1}(\mu_1 - \mu_0) \tag{2.4}$$

the squared Mahalanobis distance between the two classes. Similarly, the class-conditional error rate when the true class is 1 is

$$\Phi\{-(\lambda + \frac{1}{2}\Delta^2)/\Delta\}.$$

When the two classes have equal prior probabilities, the overall error rate is $\Phi(-\frac{1}{2}\Delta)$.

If $\pi_1, \mu_1, \mu_0, \Sigma$ are known, x can be assigned using the linear discriminant function. However, generally $\pi_1, \mu_0, \mu_1, \Sigma$ are unknown. Suppose a training sample is available, which is of the form: $(y_1, x_1), (y_2, x_2), \ldots, (y_n, x_n)$ where y_j indicates the class to which the sample unit belongs; and

$$x_j \mid y_j \sim \mathcal{N}_p(\mu_{y_j}, \Sigma)$$

where the notation $\sim$ means 'distributed as' and the notation $\mathcal{N}_p(\mu_{y_j}, \Sigma)$ means 'p-dimensional multinormal distribution with mean vector

$\boldsymbol{\mu}_{y_j}$ and covariance matrix Σ'. If $(y_j, \boldsymbol{x}_j)$, $j = 1, 2, \ldots, n$ is a random sample from the mixture

$$\sum_{i=1}^{2} \pi_i \mathcal{N}_p(\boldsymbol{\mu}_i, \Sigma)$$

then the following formulae yield maximum likelihood estimates of the unknown parameters, where $n_1 = \sum_{j=1}^{n} y_j$; $n_0 = n - n_1$ (see [3, 36]).

$$\widehat{\pi_1} = \frac{n_1}{n}; \quad \widehat{\pi_0} = \frac{n_0}{n};$$

$$\widehat{\mu_1} = \overline{\boldsymbol{x}}_1 = \frac{1}{n_1} \sum_{y_j=1} \boldsymbol{x}_j; \quad \widehat{\mu}_0 = \overline{\boldsymbol{x}}_0 = \frac{1}{n_0} \sum_{y_j=0} \boldsymbol{x}_j;$$

$$\widehat{\Sigma} = \frac{1}{n} \left[\sum_{y_j=1} (\boldsymbol{x}_j - \overline{\boldsymbol{x}}_1)(\boldsymbol{x}_j - \overline{\boldsymbol{x}}_1)' + \sum_{y_j=0} (\boldsymbol{x}_j - \overline{\boldsymbol{x}}_0)(\boldsymbol{x}_j - \overline{\boldsymbol{x}}_0)' \right].$$

Upon substitution of these estimates in the linear discriminant function we get Anderson's [3] estimated discriminant procedure to assign a new $\boldsymbol{x}$. This is *supervised learning* or *learning with a teacher* or *supervisor*.

The sample-based discriminant function is subject to sampling fluctuations and is a random variable. Consequently, the error rates associated with the sample-based discriminant function are also random variables. If we are interested in the performance of the sample-based discriminant function, then we could study the sampling distribution of these error rates, especially the mean value of the overall error rate. Such a study will investigate the performance of the sampling and learning schemes rather than the sample-based particular discriminant function and this performance will depend upon the size of the training set and the parameters of the class-conditional distributions and the class proportions. Given the parameters and the sample size, however, the error rate distributions and summary measures thereof will help understand the efficiency of the particular sampling and learning procedures. A study of efficiencies of different types of learning procedures or supervision is the main theme of this chapter.

Since the Bayes error rate is the best we can achieve, the overall error rate of the sample-based linear discriminant function is larger than the Bayes error rate. In the case of a two-class p-variate normal with a common covariance matrix with equal prior probabilities and where separate samples of the same size are taken from each class, the overall error rate exceeds the Bayes error rate of $\Phi(-\frac{1}{2}\Delta)$ approximately by

$$\frac{1}{n}\{\phi(\frac{1}{2}\Delta)/4\}\{p\Delta + 4(p-1)\Delta^{-1}\}$$

where ϕ refers to the standard normal density. The error of approximation is of order $O(n^{-2})$ [36, 42]. This formula will help determine, for this sampling scheme and the learning scheme, what the required sample size n is in order to attain a level of error rate as compared to the Bayes error rate. This of course depends upon Δ (how well the two classes are separated) and the dimension p of the feature vector. However, Δ is generally unknown and hence its estimate has to be used. It is well known that estimates of error rates of a discriminant function from the same data set as that from which the discriminant function has been computed, lead to optimistically biased estimates. Methods are available for bias-free estimates of error rates; leave-one-out and such jackknife methods are popular in this context.

The method of estimation or learning gets more complicated if the data are of the unsupervised kind. In our case of the multinormal model, maximum likelihood estimation can be carried out by a fairly efficient algorithm called the Expectation–Maximization (EM) algorithm (see [12, 38]).

2.2.2 Logistic regression

It follows from Bayes theorem that $\Lambda(\boldsymbol{x})$ as given in (2.2) is the posterior log odds ratio of Class 1 with respect to Class 0 using observation $\boldsymbol{x}$. Thus

$$\Lambda(\boldsymbol{x}_j) = \log\frac{\pi_1(\boldsymbol{x}_j)}{\pi_0(\boldsymbol{x}_j)}$$

where $\pi_i(\boldsymbol{x}_j) = \text{Prob}(y_j = i|\boldsymbol{x}_j)$, $i = 1, 0$. Given observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, the y_j are conditionally independent binary random variables. Thus,

$$\begin{aligned}\pi_1(\boldsymbol{x}_j) &= \text{Prob}(y_j = 1|\boldsymbol{x}_j) = \tfrac{\exp(\boldsymbol{\beta}_0+\boldsymbol{\beta}'\boldsymbol{x}_j)}{[1+\exp(\boldsymbol{\beta}_0+\boldsymbol{\beta}'\boldsymbol{x}_j)]},\\ \pi_0(\boldsymbol{x}_j) &= \text{Prob}(y_j = 0|\boldsymbol{x}_j) = \tfrac{1}{[1+\exp(\boldsymbol{\beta}_0+\boldsymbol{\beta}'\boldsymbol{x}_j)]}.\end{aligned} \tag{2.5}$$

This itself can be considered as a model instead of the more restrictive normal model with $\boldsymbol{\beta}_0, \boldsymbol{\beta}$ as parameters. This is known as the *logistic regression model*. Estimation of the parameters can be carried out by maximizing the conditional likelihood

$$\prod_{j=1}^{n} \pi_1(\boldsymbol{x}_j)^{y_j} \pi_0(\boldsymbol{x}_j)^{(1-y_j)} = \prod_{j=1}^{n} \frac{\exp[(\boldsymbol{\beta}_0 + \boldsymbol{\beta}'\boldsymbol{x}_j)y_j]}{[1+\exp(\boldsymbol{\beta}_0 + \boldsymbol{\beta}'\boldsymbol{x}_j)]}$$

in terms of $\boldsymbol{\beta}_0, \boldsymbol{\beta}$. Such estimates are called *logistic regression estimates*. The logistic regression is then used for assigning a new specimen in much the same way as the linear discriminant function. Efron [15] has shown that the information matrix of the estimates of $\beta_0, \boldsymbol{\beta}$ by logistic regression is

$$\boldsymbol{I}_{LR} = \pi_0\pi_1 \begin{pmatrix} A & \bigcirc \\ \bigcirc & a_0 I_{p-1} \end{pmatrix} \tag{2.6}$$

where I_{p-1} is the $(p-1)$-dimensional identity matrix,

$$A = \begin{pmatrix} a_0 & a_1 \\ a_1 & a_2 \end{pmatrix}$$

and

$$a_i = \int \frac{x^i \phi(x) \exp\{-(\Delta^2/8)\}}{\pi_1 \exp(\Delta x/2) + \pi_0 \exp\{-(\Delta x/2)\}} dx.$$

2.2.3 Efron's asymptotic relative efficiency

The asymptotic relative efficiency relevant to a situation is based on the error rate of the Fisher's linear discriminant function. As we noticed earlier, the Bayes rule is the one with the least error rate. The *Asymptotic Error Rate (AER)* of a procedure or a learning scheme based on estimates $\begin{pmatrix} \hat{\beta}_0 \\ \hat{\boldsymbol{\beta}} \end{pmatrix}_n$ of vector $\begin{pmatrix} \boldsymbol{\beta}_0 \\ \boldsymbol{\beta} \end{pmatrix}_n$ from a sample size n is defined to be the limiting value (as $n \to \infty$) of the additional error rate ('expected regret') of $\begin{pmatrix} \hat{\beta}_0 \\ \hat{\boldsymbol{\beta}} \end{pmatrix}_n$ over the Bayes error rate. This AER will naturally depend on the nature of the learning scheme, the estimates used for the parameters and the actual values of the parameters. We shall consider maximum likelihood estimators (MLE). For instance, the AER of perfectly supervised and unsupervised schemes are different and the unsupervised procedure will have a larger AER. Again, for the same supervision scheme, the AER of logistic regression estimates will be different from that of the MLE. When several schemes less efficient than the supervised one are considered, such as unsupervised, combination of supervised and unsupervised, error-prone supervised or stochastically supervised, the (perfectly) supervised scheme may be used as a basis of comparison. Moreover, the error rate is a property of particular estimates and hence when we consider a certain learning scheme, it is a random variable. Thus AER has to be considered in terms of its average properties. This discussion leads us to the notion of *Asymptotic Relative Efficiency*, which we define below.

Since error rates of discriminant rules based on $\boldsymbol{\beta}_0, \boldsymbol{\beta}$ or its estimates are invariant under linear transformations on x and depend on $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma$ only through the Mahalanobis distance Δ between the two classes, defined in (2.4), we assume the canonical form for $(\boldsymbol{\mu}_0, \Sigma)$ and $(\boldsymbol{\mu}_1, \Sigma)$ to be $(\frac{\Delta}{2}e_1, I_p)$ and $(-\frac{\Delta}{2}e_1, I_p)$, where e_1 is the vector $(1, 0, \ldots, 0)$ and I_p is the $p \times p$ identity matrix. This canonical form is obtainable by a linear transformation on x. Let $(\hat{\beta}_0, \hat{\boldsymbol{\beta}})_n$ denote the estimate of $(\beta_0, \boldsymbol{\beta})$ based on a sample of n by a certain procedure and let $\mathsf{ER}(\hat{\beta}_0, \hat{\boldsymbol{\beta}})$ denote the error rate upon using $(\hat{\beta}_0, \hat{\boldsymbol{\beta}})$ for $(\beta_0, \boldsymbol{\beta})$ in (2.2) and (2.3). Efron [15] shows that if

$$\sqrt{n}\left[\begin{pmatrix}\hat{\beta}_0\\ \hat{\boldsymbol{\beta}}\end{pmatrix}_n - \begin{pmatrix}\boldsymbol{\beta}_0\\ \boldsymbol{\beta}\end{pmatrix}_n\right] \xrightarrow{L} \mathcal{N}_{p+1}(\boldsymbol{O}, \boldsymbol{M})$$

then

$$n[\mathsf{ER}(\hat{\beta}_0, \hat{\boldsymbol{\beta}})_n - \mathsf{ER}(\beta_0, \boldsymbol{\beta})] \xrightarrow{L}$$

$$\frac{\pi_1}{2\Delta}\phi\left(\frac{\Delta}{2} - \frac{\lambda}{\Delta}\right)\left[z_0^2 - \left(\frac{2\lambda}{\Delta}\right)z_0 z_1 + \left(\frac{\lambda}{\Delta}\right)^2 z_1^2 + z_2^2 + \cdots + z_p^2\right]$$

where $\xrightarrow{L}$ means convergence in law or distribution, $z = (z_1, z_2, \ldots, z_p) \sim \mathcal{N}_{p+1}(\boldsymbol{O}, \boldsymbol{M})$, ϕ is standard normal density, and $\boldsymbol{O}$ the $(p+1)$-null vector. The *Asymptotic Error Rate (AER)* of a procedure with estimates $(\hat{\beta}_0, \hat{\boldsymbol{\beta}})_n$ is then defined to be the expectation of the limit above, which is equal to

$$\frac{\pi_1}{2\Delta}\phi\left(\frac{\Delta}{2} - \frac{\lambda}{\Delta}\right)\left[m_{00} - \left(\frac{2\lambda}{\Delta}\right)m_{01} + \left(\frac{\lambda}{\Delta}\right)^2 m_{11} + m_{22} + \ldots + m_{pp}\right]$$

where $((m_{ij})) = \boldsymbol{M}$. This is denoted for convenience by AER$(\hat{\beta}_0, \hat{\boldsymbol{\beta}})$. Then the Asymptotic Relative Efficiency (ARE) of a procedure with $(c_0, \mathbf{c})$ with respect to a procedure yielding estimate $(b_0, \boldsymbol{b})$ is

$$\mathrm{Eff}_p = \frac{\mathrm{AER}(b_0, \boldsymbol{b})}{\mathrm{AER}(c_0, \boldsymbol{c})}. \tag{2.7}$$

The computation of AER for a learning scheme needs the variance-covariance matrix $\boldsymbol{M}$, which is often computed for MLE by computing the information matrix and inverting it. Efron [15] has computed the information matrix for the perfectly supervised case. It is as follows:

$$\boldsymbol{I}_c = \pi_0\pi_1 \begin{bmatrix} \boldsymbol{H} & 0 \\ 0 & (1+\Delta^2\pi_0\pi_1)^{-1} I_{p-1} \end{bmatrix} \tag{2.8}$$

where

$$\boldsymbol{H}^{-1} = \begin{bmatrix} 1 + \frac{\Delta^2}{4} & -(\pi_0 - \pi_1)\frac{\Delta}{2} \\ -(\pi_0 - \pi_1)\frac{\Delta}{2} & 1 + 2\pi_0\pi_1\Delta^2 \end{bmatrix}.$$

2.2.4 Efficiency of logistic regression

As pointed out earlier, logistic regression and the linear discriminant function are two parametric methods of obtaining a discriminant function. It was also pointed out that the efficiency of a learning scheme depends on the estimators used for the parameters. The multinormal model describes the feature vector in terms of multinormal distributions in each class with a common covariance matrix and obtains the linear discriminant function as functions of the parameters of these distributions and the prior probabilities. On the other hand, the logistic model directly describes a model for the posterior probabilities in terms of parameters of a linear regression. Both result in linear discriminant functions. However, the estimation procedures of the parameters are different and the two procedures use the data in different ways. Then the question arises as to which of the two procedures is better. The logistic regression is a more general model and hence is more robust than the normal model. However, the logistic regression is not very efficient compared to the linear discriminant function, in the sense of the above definition of ARE, when the multinormal model actually holds. This is the theme of Efron [15]. Thus, when the validity of the multinormal model is questionable it is safer to use the logistic regression and when the multinormal model is likely to be valid it is more efficient to use the linear discriminant function.

Efron [15] derived a formula for the ARE of logistic regression relative to normal discrimination. The actual values of ARE depend on the Mahalanobis distance Δ between the two classes and the prior probabilities of the two classes, π_0, π_1 $(\pi_0 + \pi_1 = 1)$. Efron's result is as follows: Let

$$A_i(\pi_1, \Delta) = \int_{-\infty}^{\infty} \frac{e^{-\Delta^2/2} x^i \phi(x)}{\pi_1 e^{\Delta x/2} + \pi_0 e^{-\Delta x/2}}$$

$$Q_1 = \left(1, \tfrac{\lambda}{\Delta}\right) \begin{bmatrix} 1 + \frac{\Delta^2}{4} & (\pi_0 - \pi_1)\frac{\Delta}{2} \\ (\pi_0 - \pi_1)\frac{\Delta}{2} & 1 + 2\pi_0\pi_1\Delta^2 \end{bmatrix} \begin{pmatrix} 1 \\ \frac{\lambda}{\Delta} \end{pmatrix}$$

$$Q_2 = 1 + \pi_1\pi_0\Delta^2$$

$$Q_3 = \left(1, \tfrac{\lambda}{\Delta}\right) \frac{1}{A_0A_2 - A_1^2} \begin{bmatrix} A_2 & A_1 \\ A_1 & A_0 \end{bmatrix} \begin{pmatrix} 1 \\ \frac{\lambda}{\Delta} \end{pmatrix}$$

$$Q_4 = \frac{1}{A_0}$$

$$\mathrm{Eff}_1(\lambda, \Delta) = Q_1/Q_3; \quad \text{and} \quad \mathrm{Eff}_\infty = Q_2/Q_4$$

$$q(\lambda, \Delta) = Q_3/Q_4.$$

Then the relative efficiency of logistic regression to normal discrimination is

$$\mathrm{Eff}_p(\lambda, \Delta) = \frac{q(\lambda, \Delta)\mathrm{Eff}_1(\lambda, \Delta) + (p-1)\mathrm{Eff}_\infty(\lambda, \Delta)}{q(\lambda, \Delta) + (p-1)}.$$

Thus Efficiency is a convex combination of asymptotic efficiency as $p \to \infty$ and when $p = 1$. The terms Eff_∞ and Eff_1 can also be interpreted as the efficiency for estimating the slope and intercept respectively of the discriminant function. In Table 2.2.4, numerical values of these efficiencies as well as the weights required to compute Eff_p for any p are presented for a few parameter values. Efron [15] has a table for more parameter values.

From this it is seen that the logistic regression is about one-half to two-thirds as efficient as the normal discrimination when the normality assumptions actually hold. The relative efficiency of logistic regression decreases as the

Table 2.1 Relative Efficiencies of Logistic Regression to Normal Discrimination

π_1 or π_0	Δ	Eff_∞	Eff_1	$q(\lambda, \Delta)$
0.5	2	0.899	0.899	1
0.667	2	0.879	0.913	1.070
0.9	2	0.801	0.804	1.697
0.5	3	0.641	0.641	1
0.667	3	0.618	0.662	1.023
0.9	3	0.511	0.667	1.225
0.5	4	0.343	0.343	1
0.667	4	0.328	0.358	1.009
0.9	4	0.252	0.416	1.094

two classes become more separated and as one of the classes becomes more dominant in the mixture.

We follow the foregoing method of analysis for making comparisons between various learning schemes. Moreover, logistic regression is also a useful tool for such comparisons with other learning schemes as we shall see in the sequel.

2.3 Unsupervised learning

2.3.1 Unsupervised learning and mixture resolution

In many applications of pattern recognition, classifying a training sample is expensive and difficult. So one might want to use an unclassified training sample, thereby attempt to learn without a supervisor. Such unsupervised learning is possible only under certain circumstances. The problem of possibility of unsupervised learning is related to the problem of *identifiability of mixtures* [45, 46], which means that a mixture has a unique resolution in terms of elements of $\mathcal{F}$. Yakowitz [52] has shown that if $\mathcal{F}$ is a parametric family associated with an unsupervised learning problem, then the problem has a solution if and only if $\mathcal{F}$ is identifiable. Thus it is meaningful to carry out unsupervised learning only if the models for the class-conditional distributions for the feature vector belong to an identifiable family. We shall be mostly concerned with the multinormal distribution , in which case the identifiability property holds and it makes sense to attempt to carry out unsupervised learning.

Books [16, 37, 39, 49] deal with the problem of mixture distributions.

2.3.2 Maximum likelihood estimation

Let us have an unsupervised training sample $x_1, x_2, \ldots, x_n$. An algorithm for estimating the parameters was given by Day [11] . The likelihood of the sample is

$$L(\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\mu}_0, \Sigma) = (2\pi)^{-np/2}|\Sigma|^{-n/2}\prod_{j=1}^{n}(\pi_1 e_{1j} + \pi_0 e_{0j})$$

where

$$e_{ij} = \exp\{-\frac{1}{2}(\boldsymbol{x}_j - \boldsymbol{\mu}_i)'\Sigma^{-1}(\boldsymbol{x}_j - \boldsymbol{\mu}_i)\}.$$

Further, let

$$\boldsymbol{u} = \pi_1\boldsymbol{\mu}_1 + \pi_0\boldsymbol{\mu}_0$$

$$\boldsymbol{R} = \Sigma + \pi_0\pi_1(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)'.$$

Let $\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\mu}_0, \Sigma$ be reparameterized in terms of $\boldsymbol{u}, \boldsymbol{R}, \beta_0, \boldsymbol{\beta}$; this clearly is possible in view of the one-one nature of the relationship between the two parameterizations. Day [11] derives the MLE to be

$$\widehat{\boldsymbol{u}} = \sum_{j=1}^{n} \boldsymbol{x}_j/n \tag{2.9}$$

$$\widehat{\boldsymbol{R}} = \frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{x}_j - \bar{\boldsymbol{x}})(\boldsymbol{x}_j - \bar{\boldsymbol{x}})' \tag{2.10}$$

$$\widehat{\boldsymbol{\beta}} = \frac{\widehat{\boldsymbol{R}}^{-1}(\widehat{\boldsymbol{\mu}}_1 - \widehat{\boldsymbol{\mu}}_0)}{1 - \widehat{\pi_0\pi_1}(\widehat{\boldsymbol{\mu}}_1 - \widehat{\boldsymbol{\mu}}_0)'\widehat{\boldsymbol{R}}^{-1}(\widehat{\boldsymbol{\mu}}_1 - \widehat{\boldsymbol{\mu}}_0)} \tag{2.11}$$

$$\widehat{\beta}_0 = \log(\widehat{\pi}_1/\widehat{\pi}_0) - \widehat{\boldsymbol{\beta}}'(\widehat{\boldsymbol{\mu}}_1 + \widehat{\boldsymbol{\mu}}_0). \tag{2.12}$$

Equations (2.9) and (2.10) are solved directly. However, (2.11)–(2.12) and (2.13)–(2.15) below are interdependent. Using the expression (2.5) and the expressions (2.11)–(2.12) along with (2.10), expressions for estimators for $\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\mu}_0, \Sigma$ may be obtained, if required, as follows:

$$\widehat{\pi}_{1j} = 1 - \widehat{\pi}_{0j} = \frac{\exp(\widehat{\beta}_0 + \widehat{\boldsymbol{\beta}}' \boldsymbol{x})}{\{1 + \exp(\widehat{\beta}_0 + \widehat{\boldsymbol{\beta}}' \boldsymbol{x})\}} \tag{2.13}$$

$$\widehat{\pi}_1 = 1 - \widehat{\pi}_0 = \sum_{j=1}^{n} \frac{\widehat{\pi}_{1j}}{n} \tag{2.14}$$

$$\widehat{\boldsymbol{\mu}}_i = \frac{\sum_{j=1}^{n} \boldsymbol{x}_j \widehat{\pi}_{ij}}{\sum_{j=1}^{n} \widehat{\pi}_{ij}}, \quad i = 1, 0. \tag{2.15}$$

$$\widehat{\Sigma} = \widehat{\boldsymbol{R}} - \widehat{\pi}_0 \widehat{\pi}_1 (\widehat{\boldsymbol{\mu}}_1 - \widehat{\boldsymbol{\mu}}_0)(\widehat{\boldsymbol{\mu}}_1 - \widehat{\boldsymbol{\mu}}_0)'.$$

These equations can be solved by standard iterative methods such as Newton-Raphson method or Fisher's scoring method. However, an algorithm in the lines of what we now call the EM algorithm is very effective and was already suggested by Day [11]. Suppose we have initial values $\beta_{00}, \boldsymbol{\beta}_0$ for $\beta_0, \boldsymbol{\beta}$ respectively. Substituting these in (2.13), (2.14) and (2.15) and using (2.11) and (2.12) along with (2.10), we get revised estimates of β_0 and $\boldsymbol{\beta}$. This procedure can be repeated until convergence. These together with the use of (2.10) give the required MLE of the parameters $\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\mu}_0, \Sigma$ as well as those of the discriminant function coefficients β_0 and $\boldsymbol{\beta}$. For details of the EM algorithm, see [38].

2.3.3 Covariance matrix of MLE

Now let us derive the covariance matrix of these estimates and the ARE of the unsupervised learning scheme. We follow O'Neill's [41] treatment. Let $\ell(\boldsymbol{x}, y), \ell(\boldsymbol{x})$ and $\ell(y|\boldsymbol{x})$ denote respectively the full log-likelihood of $(\boldsymbol{x}, y)$, the marginal log-likelihood of $\boldsymbol{x}$ and the conditional log-likelihood of $y|\boldsymbol{x}$. These correspond to the (perfectly) supervised, unsupervised and logistic regression learning schemes respectively. Let $f_1(\boldsymbol{x})$ and $f_0(\boldsymbol{x})$ respectively denote the multinormal density with mean vector $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_0$ and a common covariance matrix Σ. Then

$$\ell(\boldsymbol{x}, y) = \log\{(\pi_1 f_1(\boldsymbol{x}))^y \ (\pi_0 f_0(\boldsymbol{x}))^{1-y}\}$$
$$\ell(\boldsymbol{x}) = \log\{(\pi_1 f_1(\boldsymbol{x})) + (\pi_0 f_0(\boldsymbol{x}))\}$$
$$\ell(y|\boldsymbol{x}) = \log\{\pi_1(\boldsymbol{x})^y \pi_0(\boldsymbol{x})^{1-y}\}$$

where $\pi_1(\boldsymbol{x})$ and $\pi_0(\boldsymbol{x})$ are as defined in (2.5) defining the logistic regression. Let $\boldsymbol{I_C}, \boldsymbol{I_{UC}}$ and $\boldsymbol{I_{LR}}$ respectively denote the full, marginal and conditional Fisher information matrices of $\beta_0, \boldsymbol{\beta}$ arising from these log-likelihoods; Fisher information matrix is the matrix of expected values of second partial derivatives of log-likelihood with respect to pairs of parameters, or equivalently, the matrix of expected values of products of partial derivatives of log-likelihood with respect to pairs of parameters. That is,

$$\boldsymbol{I} = ((E(-\frac{\partial^2 \ell}{\partial\theta_i \partial\theta_j})) = ((E(\frac{\partial \ell}{\partial\theta_i} \frac{\partial \ell}{\partial\theta_j})).$$

The inverse of this matrix gives the asymptotic covariance matrix of the parameters. It is easily seen that since $f(\boldsymbol{x}, y) = f(y|\boldsymbol{x})f(\boldsymbol{x})$ for density f, it follows that

$$\ell(\boldsymbol{x}, y) = \ell(\boldsymbol{x}) + \ell(y|\boldsymbol{x}). \tag{2.16}$$

While the covariance matrix of estimates of a subset of parameters is the corresponding submatrix of the full covariance matrix and the full covariance matrix is the inverse of the full information matrix, the information matrix of a subset of parameters is *not* the inverse of the corresponding submatrix of the covariance matrix—it is the corresponding submatrix of the inverse of the entire covariance matrix. In order to invoke the formula (2.7) for ARE, we need to compute the covariance matrix of the estimators of $\beta_0, \boldsymbol{\beta}$. Let $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ be the information matrices of the parameters $\boldsymbol{u}, \boldsymbol{R}, \beta_0, \boldsymbol{\beta}$ from the full, marginal and conditional likelihoods respectively, where $\boldsymbol{R}$ is written as

$$\boldsymbol{R}' = (r_{11}, r_{12}, \ldots, r_{1p}, r_{21}, r_{22}, \ldots, r_{2p}, \ldots, \ldots, r_{p1}, r_{p2}, \ldots, r_{pp}).$$

It follows from (2.16) that $\boldsymbol{A} = \boldsymbol{B} + \boldsymbol{C}$. Let $\boldsymbol{A}$ be partitioned as

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix}$$

corresponding to $(\boldsymbol{u}', \boldsymbol{R}')$ and $(\beta_0, \boldsymbol{\beta}')$. Let us use analogous notations for conformal partitions of $\boldsymbol{B}, \boldsymbol{C}$ and their inverses. Let us make a partition of

A^{-1} conformally as follows:

$$A^{-1} = \begin{pmatrix} A^{11} & A^{12} \\ A^{21} & A^{22} \end{pmatrix}$$

and B^{-1} similarly.

Since $\ell(y|x)$ does not involve (u, R), C_{11}, C_{12} and C_{22} are null matrices. The asymptotic covariance matrices of $\beta_0, \boldsymbol{\beta}$ for the supervised, unsupervised and logistic regression learning schemes are respectively A^{22}, B^{22} and C_{22}^{-1}, with corresponding inverses being the Fisher information matrices of $\beta_0, \boldsymbol{\beta}$. The matrices C_{11}, C_{12}, C_{21} being null, the relation $A = B + C$ easily gives rise to the following, using a formula for inverse of partitioned matrices (see [43], p. 33, for instance)

$$\begin{aligned}(A^{22})^{-1} &= A_{22} - A_{21}A_{11}^{-1}A_{12} \\ &= B_{22} + C_{22} - B_{21}B_{11}^{-1}B_{12} \\ &= (B^{22})^{-1} + C_{22}.\end{aligned}$$

This gives rise to the equation $I_{UC} = I_C - I_{LR}$. Thus the asymptotic covariance matrix of estimates $b_0, \boldsymbol{b}$ of $\beta_0, \boldsymbol{\beta}$ by the unsupervised method is $(I_C - I_{LR})^{-1}$, of c_0, c by the supervised method is I_C^{-1}.

2.3.4 Efficiency of unsupervised learning

Using the formulae (2.8) and (2.6) for I_C and I_{LR} respectively, we obtain

$$\begin{aligned}&\mathrm{Eff}_p(\lambda, \Delta) \\ &= \frac{(1, -\lambda/\Delta)H^{-1}(1, -\lambda/\Delta)' + (p-1)(1 + \pi_0\pi_1\Delta^2)}{(1, -\lambda/\Delta)(H - A)^{-1}(1, -\lambda/\Delta)' + (p-1)\frac{1+\pi_0\pi_1\Delta^2}{1-a_0(1+\pi_0\pi_1\Delta^2)}} \\ &= \frac{q(\lambda, \Delta)\mathrm{Eff}_1(\lambda, \Delta) + (p-1)\mathrm{Eff}_\infty(\lambda, \Delta)}{q(\lambda, \Delta) + (p-1)}\end{aligned}$$

where

$$q(\lambda, \Delta) = \frac{(1, -\lambda/\Delta)(H - A)^{-1}(1, -\lambda/\Delta)'[1 - a_0(1 + \pi_0\pi_1\Delta^2)]}{1 + \pi_0\pi_1\Delta^2}$$

Table 2.2 Asymptotic Relative Efficiency of Unsupervised Learning

π_1	Δ	$\mathrm{Eff}_1(\lambda, \Delta)$	$\mathrm{Eff}_\infty(\lambda, \Delta)$	$q(\lambda, \Delta)$
0.5	2	0.100	0.100	1
	3	0.360	0.360	1
	4	0.655	0.655	1
0.667	2	0.085	0.122	1.61
	3	0.338	0.382	1.24
	4	0.642	0.672	1.15
0.9	2	0.058	0.199	5.82
	3	0.254	0.490	3.08
	4	0.558	0.749	2.42

$$\mathrm{Eff}_1(\lambda, \Delta) = \frac{(1 - \lambda/\Delta)\boldsymbol{H}^{-1}(1 - \lambda/\Delta)'}{(1 - \lambda/\Delta)(\boldsymbol{H} - \boldsymbol{A})^{-1}(1 - \lambda/\Delta)'}$$
$$\mathrm{Eff}_\infty(\lambda, \Delta) = 1 - a_0(1 + \pi_0\pi_1\Delta^2).$$

Thus the efficiency is a convex combination of Eff_1 and Eff_∞, and this convex combination can be computed by computing q. Eff_1 and Eff_∞ themselves are of interest as intercept and slope efficiencies as well as for the limiting situations of the dimension of the feature vector being 1 and ∞ respectively.

A tabulation of these efficiencies for a useful range of parameter values is given in Table 2.2. A more detailed table can be found in [41], p. 824.

From this table one can see that with increasing distance between the classes, the efficiency of unsupervised learning increases. Again, with increasing dominance of one class in the mixture, the efficiency of unsupervised learning decreases. Further, if the two classes are not of equal proportion in the mixture, then efficiency increases with increasing dimension of the feature vector, the slope of the discriminant function being estimated with greater efficiency than the intercept. Information available in an unsupervised sample in this range of parameter values is about a fifth to two-thirds of that in a supervised sample. Thus unsupervised learning will be profitable in situations where a large number of unsupervised sample units can be generated inexpensively compared to a supervised sample.

Some useful references for this topic are [11, 18, 35, 51].

2.3.5 Combined learning

Let us have a supervised training sample $(y_1, \boldsymbol{x}_1), (y_2, \boldsymbol{x}_2), \ldots, (y_m, \boldsymbol{x}_m)$ together with an unsupervised sample $\boldsymbol{x}_{m+1}, \boldsymbol{x}_{m+2}, \ldots, \boldsymbol{x}_n$. An algorithm for estimating the parameters analogous to the foregoing algorithm of Day [11] is easy to devise.

The likelihood of the sample is

$$L(\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\mu}_0, \Sigma) = (2\pi)^{-np/2}|\Sigma|^{-n/2} \prod_{j=1}^{m} (e_{1j}^{y_j} e_{0j}^{1-y_j}) \prod_{j=m+1}^{n} (\pi_1 e_{1j} + \pi_0 e_{0j}),$$

giving rise to estimators very similar to the unsupervised case discussed above. We first reparametrize as before. Then the estimators of π_{1j} are given by

$$\begin{aligned} \widehat{\pi}_{1j} &= 1 - \widehat{\pi}_{0j} \\ &= y_j, \quad j = 1, 2, \ldots, m \\ &= \frac{\exp(\widehat{\boldsymbol{\beta}}_0 + \widehat{\boldsymbol{\beta}}' \boldsymbol{x})}{\{1 + \exp(\widehat{\boldsymbol{\beta}}_0 + \widehat{\boldsymbol{\beta}}' \boldsymbol{x})\}}, \quad j = m+1, 2, \ldots, n \end{aligned}$$

and the remaining formulae are the same as for unsupervised learning. The EM algorithm for this case is also very similar to the unsupervised case. The only difference is that the supervised observations continue to have the $\widehat{\pi}_{1j}$ values 1 or 0 in each cycle of iteration.

In order to derive the efficiency of this combined learning scheme, let γ be $\frac{n-m}{n}$, the proportion of unsupervised observations. Then the likelihood is a product of supervised and unsupervised likelihoods and the information matrix is the weighted sum of the unsupervised and supervised information matrices, leading to

$$(1-\gamma)\boldsymbol{I}_C + \gamma \boldsymbol{I}_{UC} = (\boldsymbol{I}_C - \gamma \boldsymbol{I}_{LR}).$$

Then computations similar to the unsupervised case lead to

$$\begin{aligned} \mathsf{Eff}_p(\lambda, \Delta, \gamma) &= \frac{(1, -\lambda/\Delta)\boldsymbol{H}^{-1}(1, -\lambda/\Delta)' + (p-1)(1 + \pi_0\pi_1\Delta^2)}{(1, -\lambda/\Delta)(\boldsymbol{H} - \gamma\boldsymbol{A})^{-1}(1, -\lambda/\Delta)' + (p-1)\frac{1+\pi_0\pi_1\Delta^2}{1-\gamma a_0(1+\pi_0\pi_1\Delta^2)}} \\ &= \frac{q(\lambda, \Delta, \gamma)\mathsf{Eff}_1(\lambda, \Delta, \gamma) + (p-1)\mathsf{Eff}_\infty(\lambda, \Delta, \gamma)}{q(\lambda, \Delta, \gamma) + (p-1)} \end{aligned}$$

Table 2.3 Asymptotic Relative Efficiency of Combined Learning (for equal proportions of supervised and unsupervised samples)

π_1	Δ	$\mathrm{Eff}_1(\lambda,\Delta)$	$\mathrm{Eff}_\infty(\lambda,\Delta)$	$q(\lambda,\Delta,0.5)$
0.5	2	0.550	0.550	1
	3	0.680	0.680	1
	4	0.828	0.828	1
0.667	2	0.543	0.561	1.15
	3	0.669	0.691	1.13
	4	0.821	0.836	1.12
0.9	2	0.573	0.600	1.78
	3	0.636	0.745	1.87
	4	0.780	0.874	2.03

where

$$q(\lambda,\Delta,\gamma) \;=\; \frac{(1,-\lambda/\Delta)(\boldsymbol{H}-\gamma\boldsymbol{A})^{-1}(1,-\lambda/\Delta)'[1-\gamma a_0(1+\pi_0\pi_1\Delta^2)]}{1+\pi_0\pi_1\Delta^2}$$

with formulae for Eff_1 and Eff_∞ obtained by replacing $\boldsymbol{A}$ by $\gamma\boldsymbol{A}$ and a_0 by γa_0. Using these formulae, the efficiency of combined learning for $\gamma = 0.5$ has been tabulated in Table 2.3. Indeed, the presence of supervised samples improves the efficiency considerably.

Some useful references for this topic are [9, 10, 33, 34, 41, 47].

2.4 Models for imperfect supervision

2.4.1 Types of supervision models

In order to study various aspects of imperfect supervision such as its effect on the conventional discriminant function, estimation of the discriminant function coefficients, and consideration of efficiency of learning schemes, it is necessary to understand how, in a given context, the supervision is carried out. In other words, we have to be able to describe the mechanism by which the supervisor allocates a training sample unit to a class and how mislabeling occurs, and, if the supervision is stochastic, how the supervisor ascribes a probability distribution over classes to a training sample unit. In the statistical approach to pattern recognition, this description has to be in terms of the probability distribution of the label or in the case of stochastic supervision, the probability distribution of the variable(s) denoting the supervisor assessment. Let us denote by y, the class to which a training sample unit actually belongs and by z the class to which the supervisor allocates the unit. In the case of deterministic supervision, z takes values over the set of classes, that is, $1, 2, \ldots, c$. In the case of stochastic supervision, z is c-vector valued, $z = (z_1, z_2, \ldots, z_c)$, z_j denoting the probability assigned by the supervisor for the training sample unit to belong to class j; thus z_j's satisfy the condition that $z_j \geq 0$ and $\sum_{j=1}^{c} z_j = 1$. Thus when there are two classes, *i.e.*, $c = 2$, z can be scalar-valued and $0 \leq z \leq 1$, representing the probability that the unit belongs to class 1, say. We review in this section various models proposed and used in the literature, although we use only a few of them in the sequel.

A statistical model for supervisor imperfection could be devised by considering observed z as realizations of a random variable Z and specifying a probability model for Z. This probability distribution will usually depend upon the class y to which the unit actually belongs. Thus it appears that we need models for $P(z|y)$. If the supervision mechanism is based on features totally unassociated or uncorrelated with the feature vector x, then it may be reasonable to model $P(z|y)$ without taking into account x, that is, make $z|y$ statistically independent of x. Such models are called *random misallocation models* (in the case of deterministic supervision) and *random stochastic supervision models* (in the case of stochastic supervision).

However, in practice, units belonging to the region of overlap between classes have a greater tendency to be misallocated and in the case of stochastic supervision, tend to get an uninformative distribution over classes. For instance, a unit whose feature vector is 'near' the centroid of the wrong class may have a larger chance of being misallocated compared to a unit whose feature vector is not so near the centroid of the wrong class, if the supervision mechanism is positively associated with the feature vector x. In such cases, the modeling of $P(z|y)$ is not good enough. We have to model $P(z|y, x)$. Such models are called *nonrandom misallocation models* (in the case of deterministic supervision) and *nonrandom stochastic supervision models* (in the case of stochastic supervision).

2.4.2 Random misallocation models

Many studies on various aspects of unreliable supervision have used random misallocation models, in view of its simplicity and as a first step in the understanding of the phenomenon of imperfect supervision, for example, [7, 8, 23, 24, 28, 31, 32, 40]. Further, it appears (see [6]) that random misallocation is the least favorable from the efficiency point of view of the estimators and hence is well worth studying . Although most of these authors have only used the model for the case of two classes ($c = 2$), it is just as easily formulated more generally, as follows. Here $\boldsymbol{X}$ denotes the random feature vector.

The model here is

$$\alpha_{ji} = P(Z = i|y = j, \boldsymbol{X} = \boldsymbol{x}) = P(Z = i|y = j), \quad i, j = 1, 2, \ldots, c.$$

Clearly, $\sum_{i=1}^{c} \alpha_{ji} = 1$. Lugosi [31] called it *binary memoryless channel* model. A particular case of this model where a certain symmetry over the classes is achieved, thereby reducing the number of parameters to 1 is to make $\alpha_{ii} = \alpha$ and $\alpha_{ij} = \frac{1-\alpha}{c-1}$. This model is very restrictive and unrealistic, but for the case $c = 2$, it does help in simplifying analysis and computations in order to get an initial understanding of imperfect supervision.

2.4.3 Nonrandom misallocation models

A supervisor is more likely to misallocate a training sample unit if the unit resembles units belonging to a wrong class. Thus, even if the supervisor allocation mechanism does not depend upon the feature vector used, if there is an association between this mechanism and the feature vector, the chances of supervisor misallocation will be related to the feature vector. The more similar the unit is to the wrong class, the greater are the chances of misallocation. Thus it will be realistic to formulate a misallocation model in terms of $P(z|y, \boldsymbol{x})$. One such model is where the probability of misallocation increases with the distance between $\boldsymbol{x}$ and the mean of the class y to which it actually belongs.

A class of such models was formulated by Chhikara and McKeon in [6], special cases of which include random misallocation models and a nonrandom misallocation model earlier introduced by Lachenbruch [29].

Lugosi in [31] formulated and used other nonrandom misallocation models which he called *models with misprints in the training sequence* and *consequently* *lying teacher models.*

2.4.4 Random stochastic supervision models

Only two types of models and that too for the case of two classes have been considered in the literature. These models are specified by the distribution of Z: Observable variable indicating the supervisor's assessment of the probability of a training sample unit belonging to class 1.

Distribution models for Z

Let $g_i(z)$ be density of Z in class i.
Beta model: (Krishnan and Nandy [25, 26]): This model assumes that the supervisor's assessment follows beta distributions when the training unit belongs to either class. For the sake of parsimony of parameter, a symmetric model is assumed, whereby the parameters of the beta distributions are m, n and n, m. The 'randomness' of the model will mean that Z is statistically independent of the feature vector $\boldsymbol{X}$. Various choices of m, n give a whole range from completely unsupervised ($m = n$) to supervised ($|m - n| \to \infty$) as seen in Fig. 2.1(i). For $m \neq n$, the supervisor assessment is probabilistically more on the correct side, the larger the $|m - n|$ the more correct, approaching perfect supervision as $|m - n| \to \infty$. Thus the beta model may be a reasonable way to describe stochastic supervision.

$$g_0(z) = z^{m-1}(1-z)^{n-1}, \quad 0 < z < 1;$$

$$g_1(z) = z^{n-1}(1-z)^{m-1}, \quad 0 < z < 1;$$

Also, the difference $|m - n|$ suitably normalized may be used as an *index of supervision*, the larger the index, the better the supervision. Thus we define supervision index to be $|m - n|/\sqrt{m+n}$.
Logistic-normal model: Titterington in [48] suggests an alternative to the beta model above, which is just as flexible, but is easier to analyze than the beta model. In this model, the logit of Z, that is, logarithm of the odds ratio, namely, $W = \log \frac{Z}{1-Z}$ is modeled rather than Z itself. The sample space for W is the real line and it is natural to assume that W is normally distributed with different means and conveniently with a common variance in the two classes. In the context of medical diagnosis Aitchison in [1] and Aitchison and Begg

in [2] used this model. Thus the model is:

$$\log \frac{Z}{1-Z} |0 \sim \mathcal{N}(-\delta, \sigma^2)$$
$$\log \frac{Z}{1-Z} |1 \sim \mathcal{N}(\delta, \sigma^2).$$

The flexibility of these distributions is demonstrated in Fig. 2.1(ii) from [48], where densities of different shapes produced by various of parameters choices are presented. Figs. 2.1(iii), (iv) and (v) are also from [48], where the logistic-normal parameters are chosen so that two moments—of order r_1 and r_2—of the beta are approximately equal to those of the logistic-normal, showing that the logistic-normal family is as rich as the beta family.

An obvious extension of the beta model to the multi-class case is the Dirichlet distribution. Let $Z = (Z_1, Z_2, \ldots, Z_c)$ represent the stochastic supervisor's probability assignment to a specimen, where $Z_j \geq 0$, $\sum_{j=1}^{c} Z_j = 1$. Then the Dirichlet model is

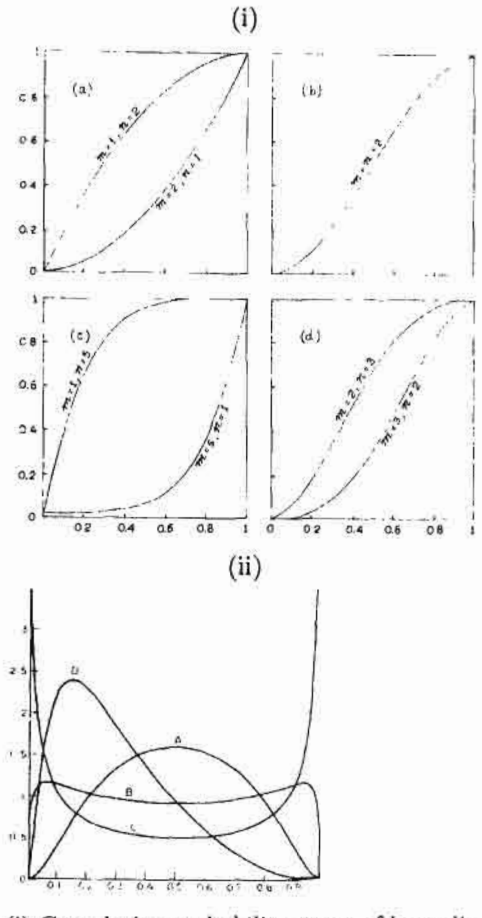
$$P(z_1, z_2, \ldots, z_c|i) = \text{constant} \times z_1^{m_{1i}} z_2^{m_{2i}} \cdots z_c^{m_{ci}},$$

where $m_{1i}, m_{2i}, \ldots, m_{ci}$ are parameters for class i, $i = 1, 2, \ldots, c$. Various simplifications can be imposed on this such as certain types of symmetry in order to reduce the number of parameters. Similarly, the logistic-normal model can be generalized by considering the $(c-1)$ log-odds-ratios $W_j = \log(Z_j/Z_1)$, $j = 2, 3, \ldots, c$ and formulating a $(c-1)$-dimensional normal distribution for $(W_2, W_3, \ldots, W_c|i)$. Simplifications can be made by the assumption of a common covariance matrix for all these multinormal distributions and by assuming certain symmetric structures for the mean vectors of these distributions.

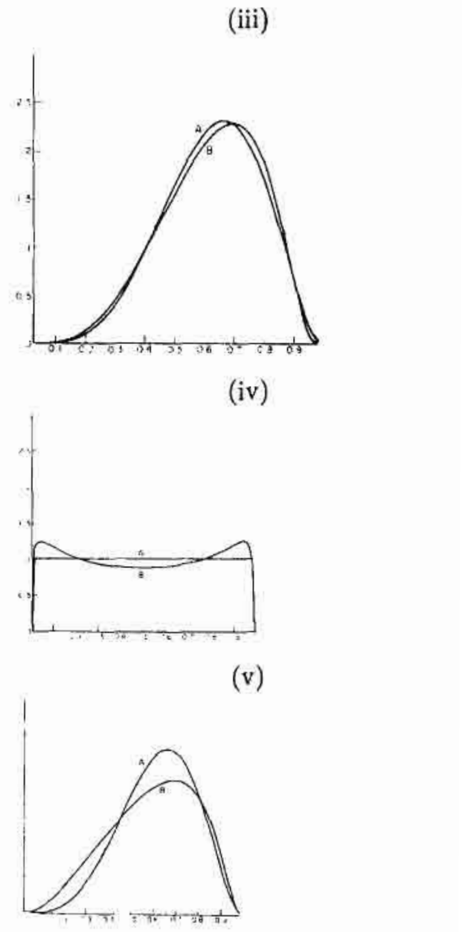
2.5 Effect of imperfect supervision

The effect of initial sample misallocation has been studied both analytically and by simulation techniques. A variety of criteria have been used in these studies. Some of these are: (1) asymptotic distribution of discrimination boundaries; (2) asymptotic mean and variance of the class-conditional and overall error rates R_0, R_1, R defined as:

$$R_0 = Pr\{\widehat{\Lambda}(\boldsymbol{x}) \geq 0 \mid \boldsymbol{x} \in C_0\}; \quad R_1 = Pr\{\widehat{\Lambda}(\boldsymbol{x}) < 0 \mid \boldsymbol{x} \in C_1\}$$



(i) Cumulative probability curve of beta distribution for various parameter values (ii) Density functions of various logistic-normal distributions (A) $\delta = 0, \sigma^1 = 1$ (B) $\delta = 0, \sigma^2 = 3$ (C) $\delta = 0, \sigma^2 = 10$ (D) $\delta = -1, \sigma^2 = 1$.



Density functions of beta (A) and c sponding logistic-normal (B) for (iii) bet 3); $r_1 = 1, r_2 = 2$ (iv) beta(1, 1); r $0.05, r_2 = 0.10$ (v) beta(5, 3); $r_1 = 0.05$, 0.10.

Fig. 2.1 Various density functions

and the overall error rate: $R = \pi_0 R_0 + \pi_1 R_1$.
(3) asymptotic bias and efficiency of discriminant function estimates.
Under the random misallocation model, Lachenbruch [28] used Monte Carlo

methods, McLachlan [32] used asymptotic expansions for misclassification probabilities and Michalek and Tripathi [40] used Efron's ARE for the case of two-class multinormal distributions with a common covariance matrix. If α_0 and α_1, the class-conditional supervisor misclassification probability of elements in class C_0, C_1 respectively, are small and equal, no appreciable increase in misclassification probabilities occurs. Let P_i be the misclassification probability of an element from class i, by the estimated discriminant function, $i = 1, 0$. Then,

$$P_1 = \Phi(-\frac{\Delta}{2}(1+\alpha_0-\alpha_1)),\ P_0 = \Phi(-\frac{\Delta}{2}(1+\alpha_1-\alpha_0)), \quad \text{if} \quad \alpha_0+\alpha_1 < 1.$$

If $\alpha_0 = \alpha_1$, these are the same as for perfect supervision. Michalek and Tripathi [40] found that for $0 < \Delta < 1.5, 0 < \alpha_0 < \alpha_1 < 0.5$, if Δ_e^2 is the distance between the misclassified populations, the real distance lies between

$$\frac{\Delta_e^2(\alpha_0+\alpha_1-1)^2}{1+\alpha_0(1-\alpha_0)\Delta_e^2} \quad \text{and} \quad \frac{\Delta_e^2(\alpha_0+\alpha_1-1)^2}{1+\alpha_1(1-\alpha_1)\Delta_e^2}$$

and the distance between misclassified populations decreases as α_0, α_1 increase. The effect of misclassification is to make the estimators of β_0, β biased, increase error rates (owing to decreased distance), make maximum likelihood estimates converge to false values and to decrease efficiency of the discriminant function. Lachenbruch [29] used a truncated model in which observations closer to the mean of the 'wrong' class had a larger chance of misallocation and by means of Monte Carlo methods found that actual error rates were only marginally affected, the apparent error rates drastically affected and Δ^2 greatly inflated. Duda and Singleton [14] studied the case of binary independent observation vectors and showed that the linearly separable discriminant function converges to the one under perfect supervision.

Lugosi [31] investigated the asymptotic behavior of the error probability of the Bayes rule in the general case in the nonparametric set-up, and the nearest neighbor classification rule for the two-class case using his models of imperfect teachers. Lachenbruch [30] considered the effect of initial sample misclassification on the quadratic discriminant function. We do not discuss these here.

2.6 Learning with an unreliable supervisor

The statistical estimation problem in the case of learning with imperfect supervision is the problem of resolution of a mixture, where the mixture model includes the supervision variable as well. An efficient method of maximum likelihood estimation for such problems is the EM algorithm, discussed for instance in [38]. The EM algorithm has been worked out for the case of random model with symmetric misallocation between groups in [23]. Often, a combination of supervision schemes is used, depending upon availability of various types of supervisors. The EM algorithms for such situations are easily developed along lines in the references cited above. References [7, 19, 21, 22, 44] discuss various other methods of learning under imperfect supervision.

We now discuss the ARE of learning under an unreliable supervisor based on the model with a constant α of supervisor misallocation of a training sample unit, introduced in Section 2.4.2. We follow the treatment in [24]. We use the same technique for working out efficiency formulae as in Section 2.3. Denoting by z the supervisor's class and by y the actual class, and regarding the observations (z_j, x_j), $j = 1, 2, \ldots, n$ as a random sample from the mixture

$$\pi_0 f_0(z, x) + \pi_1 f_1(z, x)$$

where

$$\begin{aligned} f_0(z, x) &= f_0(x)\alpha^z(1-\alpha)^{1-z} \\ f_1(z, x) &= f_1(x)\alpha^{1-z}(1-\alpha)^z \end{aligned}$$

f_0, f_1 being multinormal densities of the feature vector x in the two classes. Using the notations

$$\delta = \log[\alpha/(1-\alpha)]; \quad w = 1 - 2z$$

we easily get by methods similar to those in Section 2.3

$$\begin{aligned} \log \ell(y|z, x) &= \frac{1}{1 + \frac{(1-\alpha)^z\alpha^{1-z}}{\alpha^z(1-\alpha)^{1-z}} e^{\beta_0 + \boldsymbol{\beta}' x}} \\ &= \frac{1}{1 + e^{\beta_0 + \boldsymbol{\beta}' x + \delta w}} \end{aligned}$$

The formulae for efficiency are obtained in a manner similar to those in Section 2.3, with the additional complication due to the extra parameter δ. We give below expressions leading up to the efficiency formulae, which now depend on α also besides λ and Δ. Quantities not defined here have the same meaning as in Section 2.3.

$$A_i(\pi_1, \Delta) = \int_{-\infty}^{\infty} \frac{e^{-\Delta^2/8} x^i \phi(x)}{\pi_1 e^{\Delta x/2} + \pi_0 e^{-\Delta x/2}} dx, \quad i = 0, 1, 2$$

$$a_0 = \pi_0(1-\alpha) + \pi_1\alpha; \quad a_1 = \pi_0\alpha + \pi_1(1-\alpha)$$

$$p_0 = \pi_1\alpha/a_0; p_1 = \pi_0(1-\alpha)/a_1$$

$$F_i = [A_i(p_0, \Delta)/a_0] + [A_i(p_1, \Delta)/a_1], \quad i = 0, 1, 2$$

$$B_i = [A_i(p_0, \Delta)/a_0] - [A_i(p_1, \Delta)/a_1], \quad i = 0, 1$$

$$\boldsymbol{D} = \alpha(1-\alpha)\times$$

$$\begin{bmatrix} F_0 + (\pi_0\pi_1 B_0^2/(1-\pi_0\pi_1 F_0) & F_1 + (\pi_0\pi_1 B_0 B_1/(1-\pi_0\pi_1 F_0)) \\ F_1 + (\pi_0\pi_1 B_0 B_1/(1-\pi_0\pi_1 F_0)) & F_2 + (\pi_0\pi_1 B_1^2/(1-\pi_0\pi_1 F_0)) \end{bmatrix}.$$

Using these formulae, we write $\mathsf{Eff}_1, \mathsf{Eff}_\infty$ and Eff_p which have the same meaning and use as before.

$$\mathsf{Eff}_p(\lambda, \Delta, \alpha) = \frac{q(\lambda, \Delta, \alpha)\mathsf{Eff}_1(\lambda, \Delta, \alpha) + (p-1)\mathsf{Eff}_\infty(\lambda, \Delta, \alpha)}{q(\lambda, \Delta, \alpha) + (p-1)}$$

where

$$q(\lambda, \Delta, \alpha) = (1, -\lambda/\Delta)(\boldsymbol{H} - \boldsymbol{D})^{-1}(1, -\lambda/\Delta)'$$

$$\times[1 - \alpha(1-\alpha)F_0(1 + \pi_0\pi_1\Delta^2)]/(1 + \pi_0\pi_1\Delta^2).$$

$$\mathsf{Eff}_1(\lambda, \Delta, \alpha) = \frac{(1, -\lambda/\Delta)\boldsymbol{H}^{-1}(1, -\lambda/\Delta)'}{(1, -\lambda/\Delta)(\boldsymbol{H} - \boldsymbol{D})^{-1}(1, -\lambda/\Delta)'}$$

$$\mathsf{Eff}_\infty(\lambda, \Delta, \alpha) = 1 - \alpha(1-\alpha)F_0(1 + \pi_0\pi_1\Delta^2).$$

Thus as before the efficiency is a convex combination of Eff_1 and Eff_∞, with the same interpretation.

If $\alpha = 0$, then there is no supervision error and the efficiency is that of perfect supervision. For cases in between, the efficiency is between 0 and 1

Table 2.4 Asymptotic Relative Efficiency of Unreliable Supervisor

π_1	Δ	α	$\mathrm{Eff}_1(\lambda, \Delta)$	equiv γ	$\mathrm{Eff}_\infty(\lambda, \Delta)$	equiv γ
0.5	2	0.01	0.933	0.07	0.933	0.07
	3	0.01	0.934	0.10	0.934	0.10
	4	0.01	0.956	0.13	0.956	0.13
0.667	2	0.01	0.935	0.07	0.910	0.10
	3	0.01	0.933	0.10	0.932	0.11
	4	0.01	0.955	0.30	0.958	0.13
0.9	2	0.01	0.944	0.07	0.845	0.19
	3	0.01	0.921	0.11	0.927	0.13
	4	0.01	0.942	0.13	0.964	0.14
0.5	2	0.20	0.347	0.73	0.347	0.73
	3	0.20	0.518	0.75	0.518	0.75
	4	0.20	0.737	0.77	0.737	0.77
0.667	2	0.20	0.339	0.72	0.353	0.74
	3	0.20	0.502	0.75	0.533	0.76
	4	0.20	0.726	0.77	0.748	0.77
0.9	2	0.20	0.342	0.75	0.370	0.79
	3	0.20	0.438	0.76	0.606	0.77
	4	0.20	0.660	0.77	0.805	0.78
0.5	2	0.50	0.109	1.00	0.109	1.00
	3	0.50	0.359	1.00	0.359	1.00
	4	0.50	0.657	1.00	0.657	1.00
0.667	2	0.50	0.085	1.00	0.121	1.00
	3	0.50	0.338	1.00	0.382	1.00
	4	0.50	0.642	1.00	0.672	1.00
0.9	2	0.50	0.059	1.00	0.199	1.00
	3	0.50	0.254	1.00	0.489	1.00
	4	0.50	0.558	1.00	0.749	1.00

compared to perfect supervision. Thus, a certain proportion of supervision error is equivalent to, in terms of efficiency, a certain proportion of samples being unclassified, and so we can strike an equivalence between α of the error-prone model and γ, the proportion of unclassified observations in the combined model which result in the same level of efficiency. A tabulation of efficiencies for error-prone supervision along with equivalent γ in combined model for a useful range of parameter values is given in Table 2.4. A more detailed table can be found in [24].

As a summary, for Δ in the range 2.5–4, the efficiency range for various levels of α and their equivalent percent samples unclassified are as given in Table 2.5.

Table 2.5 Summary of efficiency of Error-Prone and Combined Learning

α	Efficiency range (%)	γ
0.05	74–90	30–45
0.10	45–75	40–60
0.20	35–80	74–79
0.35	18–70	94–95
0.50	13–75	100

2.7 Learning with a stochastic supervisor

2.7.1 Beta model

The statistical estimation problem in the case of learning with a stochastic supervisor is also a problem of resolution of a mixture, where the mixture model includes the supervision variable as well. Again, an efficient method of maximum likelihood estimation for such problems is the EM algorithm, discussed for instance in [38]. The EM algorithm has been worked out for the case of symmetric random beta model of stochastic supervision in [25], and for the logistic-normal model of stochastic supervision in [48]. We do not discuss this here but restrict our attention to efficiency issues. We follow the treatments in [26, 27].

To work out formulae for ARE of learning under stochastic supervision, we use the same technique as in Section 2.3; the formulae are a little more complicated because of the additional supervision error model parameters. De-

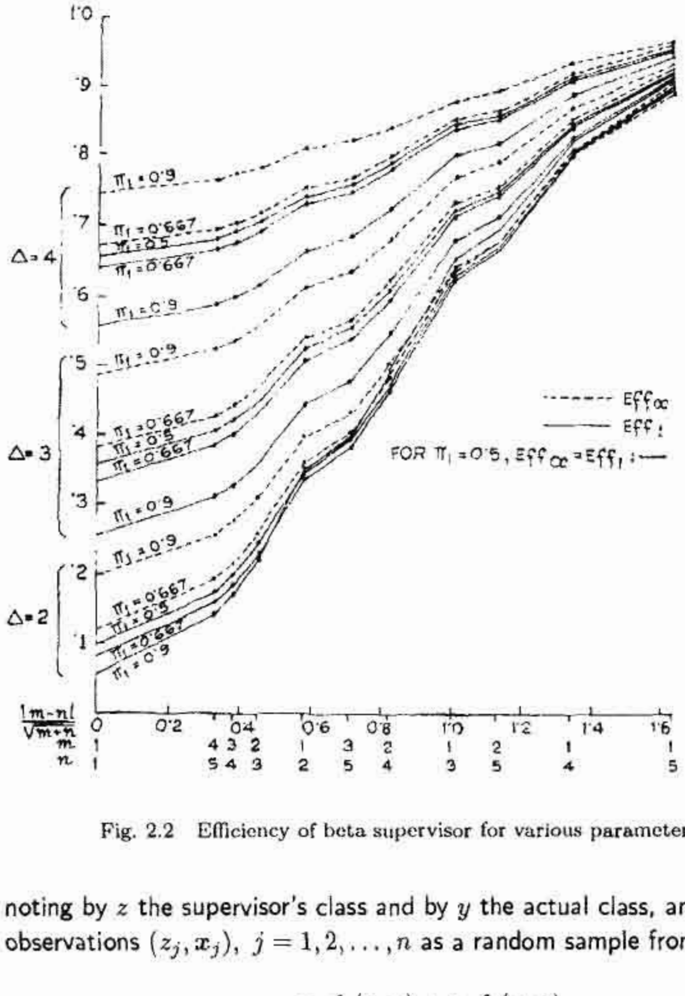


Fig. 2.2 Efficiency of beta supervisor for various parameter values.

noting by z the supervisor's class and by y the actual class, and regarding the observations $(z_j, \boldsymbol{x}_j),\ j = 1, 2, \ldots, n$ as a random sample from the mixture

$$\pi_0 f_0(z, \boldsymbol{x}) + \pi_1 f_1(z, \boldsymbol{x})$$

where

$$f_0(z, \boldsymbol{x}) = f_0(\boldsymbol{x}) z^{m-1}(1-z)^{n-1}/\mathsf{Beta}(m, n)$$
$$f_1(z, \boldsymbol{x}) = f_1(\boldsymbol{x}) z^{n-1}(1-z)^{m-1}/\mathsf{Beta}(n, m)$$

f_0, f_1 being multinormal densities of the feature vector x in the two classes, Beta being the complete beta integral. Using the notation $v = \log[z/(1-z)]$

we easily get by methods similar to those in Section 2.3

$$\log \ell(y|z, \boldsymbol{x}) = \frac{1}{1 + e^{\beta_0 + \boldsymbol{\beta}'\boldsymbol{x} + (n-m)v}}.$$

We give below expressions leading up to the efficiency formulae, which now depend on m, n also, besides λ and Δ. Quantities not defined here have the same meaning as in Section 2.3.

$$a_{ij} = \int_0^1 (\log z)^i (\log(1-z))^j z^{m-1}(1-z)^{n-1} dz, \quad j = 0, 1, 2$$

$$\boldsymbol{M} = ((M_{ij})) = \begin{bmatrix} \frac{a_{20}}{a_{00}} - \frac{a_{10}^2}{a_{00}^2} & \frac{a_{11}}{a_{00}} - \frac{a_{01}a_{10}}{a_{00}^2} \\ \frac{a_{11}}{a_{00}} - \frac{a_{01}a_{10}}{a_{00}^2} & \frac{a_{02}}{a_{00}} - \frac{a_{01}^2}{a_{00}^2} \end{bmatrix}$$

$$D_{ij} = \exp(-\Delta^2/8)/\sqrt{2\pi}\mathsf{Beta}(m,n)\times$$

$$\int_0^1 v^j z^{m+n-2}(1-z)^{m+n-2}\times$$

$$\int_{-\infty}^{\infty} \frac{x^i \exp(-x^2/2)}{\pi_1 z^{n-1}(1-z)^{m-1}\exp(\Delta x/2) + \pi_0 z^{m-1}(1-z)^{n-1}\exp(-\Delta x/2)} dx dz,$$

$$i, j = 0, 1, 2$$

$$d = M_{11}/[(M_{22} - D_{02})M_{11} - M_{12}^2]$$

$$\boldsymbol{F} = \begin{bmatrix} D_{00} + dD_{01}^2 & D_{10} + dD_{01}D_{11} \\ D_{10} + dD_{01}D_{11} & D_{20} + dD_{11}^2 \end{bmatrix}$$

$$\mathsf{Eff}_p(\pi_1, \Delta, m, n)$$

$$= \frac{q(\pi_1, \Delta, m, n)\mathsf{Eff}_1(\pi_1, \Delta, m, n) + (p-1)\mathsf{Eff}_\infty(\pi_1, \Delta, m, n)}{q(\pi_1, \Delta, m, n) + (p-1)}$$

where

$$q(\pi_1, \Delta, m, n) = \frac{(1, -\frac{\lambda}{\Delta})[\boldsymbol{H} - \boldsymbol{F}]^{-1}(1, -\frac{\lambda}{\Delta})'}{1 + \pi_0\pi_1\Delta^2}$$

$$\mathsf{Eff}_1(\pi_1, \Delta, m, n) = \frac{(1, -\frac{\lambda}{\Delta})\boldsymbol{H}^{-1}(1, -\frac{\lambda}{\Delta})'}{(1, -\frac{\lambda}{\Delta})[\boldsymbol{H} - \boldsymbol{F}]^{-1}(1, -\frac{\lambda}{\Delta})'}$$

$$\mathsf{Eff}_\infty(\pi_1, \Delta, m, n) = 1 - D_{00}(1 + \pi_0\pi_1\Delta^2).$$

Thus as before efficiency is a convex combination of Eff_1 and Eff_∞, with the same interpretation.

A tabulation of these efficiencies for a useful range of parameter values is given in Table 2.7.1. This table also gives the proportion γ of unclassified observations in combined learning which will result in the same level of efficiency. A more detailed table can be found in [26].

In Fig. 2.2 we present a chart of efficiencies for various parameter values. If $m = n$, then the supervisor uses exactly the same formula for classifying a training sample when it belongs to classes 0 and 1, and hence supervision has no information; this is equivalent to the unsupervised case. When the difference between m and n is large, there is plenty of information in the supervision variable v and the limiting case of $\mid m - n \mid \rightarrow \infty$ is the perfectly supervised case. In Fig. 2.2 we use the supervision index as defined earlier, rather than values of m and n. In terms of efficiency, a certain value of supervision index in stochastic learning is equivalent to a certain value of α for error-prone supervision, for given parameter values. For Δ in the range 2.5-4, the efficiency range for various values of the supervision index and the equivalent α in the error-prone case are given in Table 2.7.

2.7.2 Logistic-normal supervisor

Titterington [48] works out the EM algorithm for this case. We present the efficiency formulae and results here. In the case of random logistic-normal supervisor, it is assumed that $W = \log(\frac{Z}{1-Z})$ is normally distributed with means $-\eta$ and η in the two groups with a common variance σ^2 and independent of the feature vector $\boldsymbol{X}$. This makes the distribution of $(W, \boldsymbol{X})$ multinormal in the two groups with a common covariance matrix. The problem then is simply a problem of unsupervised learning with $(p+1)$-dimensional multinormal distributions with a common covariance matrix in the two groups and we can simply invoke the formulae for unsupervised learning schemes with the modification that the new Mahalanobis distance ∇ is given by $\nabla^2 = \Delta^2 + \delta^2$, where $\delta^2 = \frac{4\eta^2}{\sigma^2}$.

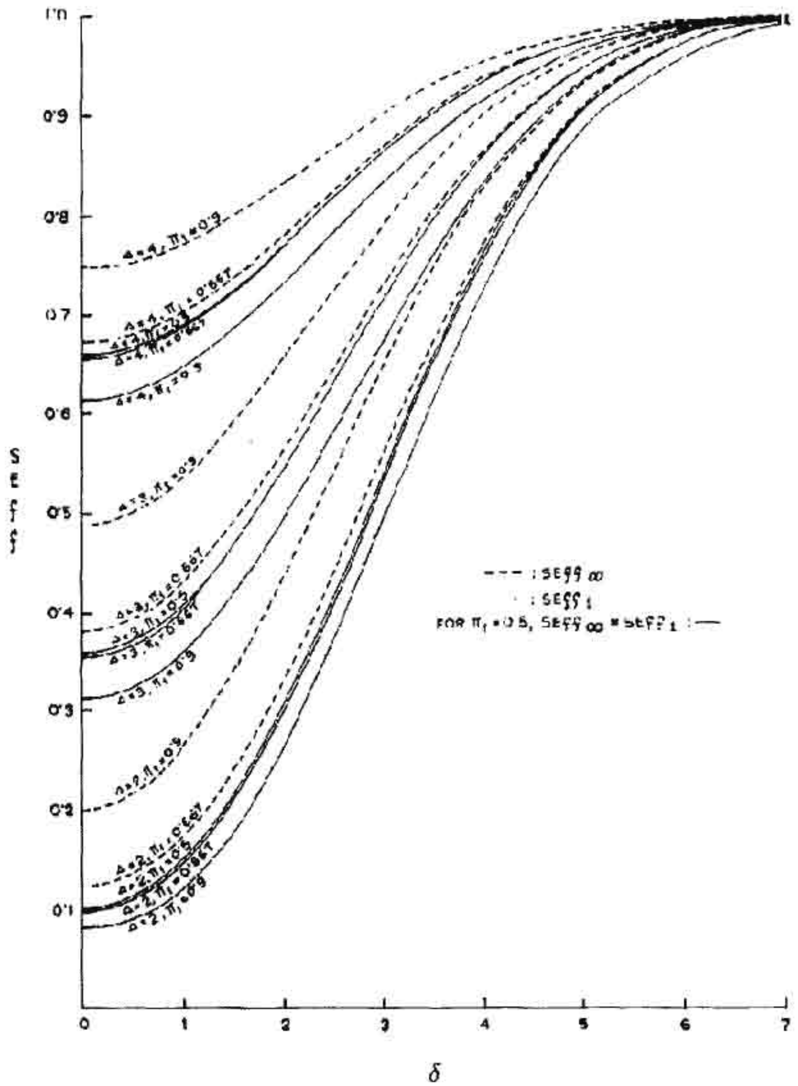


Fig. 2.3 Efficiency of logistic-normal supervisor for various parameter values.

We present these efficiency values in Table 2.8 and Fig. 2.3 (where efficiency is denoted by SEff). Results for the two models are similar although it is difficult to strike a correspondence between the beta supervision parameters

Table 2.6 Asymptotic Relative Efficiency of Beta Supervisor

			$\Delta = 2$		$\Delta = 3$		$\Delta = 4$	
π_1	m	n	Eff_1	Eff_∞	Eff_1	Eff_∞	Eff_1	Eff_∞
0.5	1	5	0.8980	0.8980	0.9194	0.9194	0.9536	0.9536
	2	4	0.4762	0.4762	0.6094	0.6094	0.7852	0.7852
	3	3	0.1016	0.1016	0.3590	0.3590	0.6570	0.6570
0.667	1	5	0.8972	0.8988	0.9171	0.9217	0.9518	0.9554
	2	4	0.4694	0.4844	0.5972	0.6223	0.7762	0.7943
	3	3	0.0847	0.1217	0.3375	0.3820	0.6422	0.6719
0.9	1	5	0.9086	0.8958	0.9097	0.9312	0.9400	0.9647
	2	4	0.4927	0.5026	0.5475	0.6801	0.7222	0.8403
	3	3	0.0595	0.1996	0.2537	0.4892	0.5580	0.7483

Table 2.7 Summary of Efficiency for Stochastic and Error-Prone Supervision

	equiv	Efficiency
$\frac{\lvert m-n\rvert}{\sqrt{m+n}}$	α	range (%)
1.67	0.01	90–95
1.42	0.05	74–90
0.40	0.20	35–80
0.30	0.35	18–70
0.00	0.5	13–75

and the logistic-normal supervision parameters. Clearly, efficiency increases with increasing supervision (larger δ or $|m-n|$) and with larger Δ, decreases with π_1 away from $\frac{1}{2}$. Values given in these tables can be interpreted like in the example below: From Table 2.8, we see that for $\Delta = 4$, $\delta = 3$, $\pi_1 = 0.667$, Eff is between 0.8630 and 0.8752; this means that in this case of stochastic supervision, 100 sample units are needed to achieve the same level of efficiency as 86 perfectly supervised samples.

Table 2.8 Asymptotic Relative Efficiency of Logistic-Normal Supervisor

		$\Delta = 2$		$\Delta = 3$		$\Delta = 4$	
π_1	δ	Eff_1	Eff_∞	Eff_1	Eff_∞	Eff_1	Eff_∞
0.5	0	0.1008	0.1008	0.3594	0.3594	0.6589	0.6589
	2	0.3072	0.3072	0.5467	0.5467	0.7716	0.7716
	3	0.5467	0.5467	0.7201	0.7201	0.8652	0.8652
	4	0.7716	0.7716	0.8652	0.8652	0.9376	0.9376
0.667	1	0.0989	0.1211	0.3572	0.3819	0.6561	0.6722
	2	0.3051	0.3305	0.5441	0.5642	0.7690	0.7805
	3	0.5441	0.5642	0.7174	0.7311	0.8630	0.8702
	4	0.7690	0.7805	0.8630	0.8702	0.9362	0.9397
0.9	1	0.0798	0.1994	0.3120	0.4896	0.6134	0.7481
	2	0.2624	0.4384	0.4976	0.6570	0.7334	0.8345
	3	0.4976	0.6570	0.6781	0.7957	0.8364	0.9031
	4	0.7334	0.8345	0.8364	0.9031	0.9199	0.9548

In general,

1. Efficiency increases with Δ.
2. Efficiency increases with better supervision (that is, for smaller proportion of unsupervised samples, for smaller α, for larger supervision index of δ or $|m - n|$).
3. For $\pi_1 = 0.5$, Eff_1 and Eff_∞ are the same; in the neighborhood of $\pi_1 = 0.5$, the difference between Eff_1 and Eff_∞ is small and the dimension of the feature vector is immaterial.
4. Unsymmetric groups, that is, π_1 near 0 or 1, need a larger p, that is, a larger-dimensional feature vector to attain the same efficiency, other parameter values remaining the same.
5. For fixed Δ, Eff_1 increases, but Eff_∞ decreases as π_1 increases for better supervision, and the other way around for worse supervision.

From these results, it is clear that imperfect supervision can also be used for learning with appropriate models and techniques, albeit with less efficiency. Its actual use will of course depend upon the choices of supervision available and the cost of these supervisors. If an imperfect supervisor is available with considerably lower cost than a perfect supervisor but who (which) is not terribly imperfect, then it seems worthwhile learning from them.

References

[1] J. Aitchison, *The Statistical Analysis of Compositional Data*. London: Chapman and Hall, 1986.

[2] J. Aitchison and C.B. Begg, "Statistical diagnosis when the basic cases are not classified with certainty," *Biometrika*, vol. 63, pp. 1–12, 1976.

[3] T.W. Anderson, *An Introduction to Multivariate Statistical Analysis*. New York: Wiley, 1984.

[4] R.S. Chhikara, "Effects of mixed (boundary) pixels on crop proportion estimation," *Remote Sensing of Environment*, vol. 14, pp. 207–218, 1984.

[5] R.S. Chhikara, "Error analysis of crop acreage estimation using satellite data," *Technometrics*, vol. 28, pp. 73–80, 1986.

[6] R.S. Chhikara and J. McKeon, "Linear discriminant analysis with misallocation in training samples," *Journal of the American Statistical Association*, vol. 79, pp. 899–906, 1984.

[7] C.B. Chittineni, "Learning with imperfectly labeled patterns," *Pattern Recognition*, vol. 12, pp. 271–281, 1980.

[8] C.B. Chittineni, "Estimation of probabilities of label imperfections and correction of mislabels," *Pattern Recognition*, vol. 13, pp. 257–268, 1981.

[9] D.B. Cooper, "When should a learning machine ask for help?" *IEEE Transactions on Information Theory*, vol. 20, pp. 455–471, 1974.

[10] D.B. Cooper and J.H. Freeman, "On the asymptotic improvement in the outcome of supervised learning provided by additional nonsupervised learning," *IEEE Transactions on Computers*, vol. 19, pp. 1055–1063, 1970.

[11] N.E. Day, "Estimating the components of a mixture of normal distributions," *Biometrika*, vol. 56, pp. 463–474, 1969.

[12] A.P. Dempster, N.M. Laird and D.B. Rubin, "Maximum likelihood from incomplete data *via* the EM algorithm," *Journal of the Royal Statistical Society*, vol. B 39, pp. 1–38, 1977.

[13] R.A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, N.J.: Prentice-Hall, 1982.

[14] R.O. Duda and R.C. Singleton, "Training a threshold logic unit with

imperfectly classified patterns," WESCON Convention, Los Angeles, 1964.

[15] B. Efron, "The efficiency of logistic regression compared to normal discriminant analysis," *Journal of the American Statistical Association*, vol. 70, pp. 892–898, 1975.

[16] B.S. Everitt and D.J. Hand, *Finite Mixture Distributions*. London: Chapman & Hall, 1981.

[17] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic Press, 1972.

[18] S. Ganesalingam and G.J. McLachlan, "The efficiency of a linear discriminant function based on unclassified initial samples," *Biometrika*, vol. 65, pp. 658–662, 1978.

[19] D.R. Gimlin and D.R. Farell, "A k–k error correcting procedure for nonparametric imperfectly supervised learning," *IEEE Transactions on Systems, Cybernetics and Man*, vol. 4, pp. 304–306.

[20] T. Imai and M. Shimura, "Learning with probabilistic labeling," *Pattern Recognition*, vol. 8, 5–10, 1976.

[21] R.L. Kashyap, "Algorithms for pattern classification," in *Adaptive Learning and Pattern Recognition Systems: Theory and Applications*, J.M. Mendel and K.S. Fu, Eds., New York: Academic, 1970, pp. 81–113.

[22] R.L. Kashyap and C.C. Blaydon, "Recovery of functions from noisy measurements taken at randomly selected points," *Proceedings of IEEE*, vol. 54, pp. 1127–1128, 1966.

[23] U.A. Katre and T. Krishnan, "Pattern recognition with imperfect supervision," *Pattern Recognition*, vol. 22, pp. 423–431, 1989.

[24] T. Krishnan, "Efficiency of learning with imperfect supervision," *Pattern Recognition*, vol. 21, pp. 183–188, 1988.

[25] T. Krishnan and S.C. Nandy, "Discriminant analysis with a stochastic supervisor," *Pattern Recognition*, vol. 20, pp. 379–384, 1987.

[26] T. Krishnan and S.C. Nandy, "Efficiency of discriminant analysis when initial samples are classified stochastically," *Pattern Recognition*, vol. 23, 529–537, 1990.

[27] T. Krishnan and S.C. Nandy, "Efficiency of logistic-normal supervision," *Pattern Recognition*, vol. 23, pp. 1275–1279, 1990.

[28] P.A. Lachenbruch, "Discriminant analysis when initial samples are misclassified," *Technometrics*, vol. 8, 657–662, 1966.

[29] P.A. Lachenbruch, "Discriminant analysis when the initial samples are

misclassified II: Nonrandom misclassification models," *Technometrics*, vol. 16, pp. 419–424, 1974.

[30] P.A. Lachenbruch, "Note on initial misclassification effects on the quadratic discriminant function," *Technometrics*, vol. 21, pp. 129–132, 1979.

[31] G. Lugosi, "Learning with an unreliable teacher," *Pattern Recognition*, vol. 25, pp. 79–88, 1992.

[32] G.J. McLachlan, "Asymptotic results for discriminant analysis when initial samples are misclassified," *Technometrics*, vol. 14, pp. 415–422, 1972.

[33] G.J. McLachlan, "Iterative reclassification procedure for constructing an asymptotically optimum rule of allocation in discriminant analysis," *Journal of the American Statistical Association*, vol. 70, pp. 365–369, 1975.

[34] G.J. McLachlan, "Estimating the linear discriminant function from initial samples containing a small number of unclassified observations," *Journal of the American Statistical Association*, vol. 72, pp. 403–406, 1977 .

[35] G.J. McLachlan, "The classification and mixture maximum likelihood approaches to cluster analysis," in *Handbook of Statistics, Vol. 2*, P.R. Krishnaiah and L.N. Kanal, Eds. Amsterdam: North-Holland, 1982, pp. 199–208.

[36] G.J. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley, 1992.

[37] G.J. McLachlan and K.E. Basford, *Mixture Models: Inference and Applications to Clustering*. New York: Marcel Dekker, 1988.

[38] G.J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*. New York: Wiley, 1977.

[39] G.J. McLachlan and D. Peel, *Finite Mixture Models*. New York: Wiley, 2000.

[40] J.E. Michalek and R.C. Tripathi, "The effect of errors in diagnosis and measurement on the estimation of probability of an event," *Journal of the American Statistical Association*, vol. 75, pp. 154–160, 1980.

[41] T.J. O'Neill, "Normal discrimination with unclassified observations," *Journal of the American Statistical Association*, vol. 73, pp. 821–826, 1978.

[42] M. Okamoto, "An asymptotic expansion for the distribution of the linear discriminant function," *Annals of Mathematical Statistics*, vol.

34, pp. 1286–1301, 1963.

[43] C.R. Rao, *Linear Statistical Inference and its Applications*. New York: Wiley, 1973.

[44] K. Shanmugam and A.M. Breipohl, "An error-correcting procedure for learning with an imperfect teacher," *IEEE Transactions on Systems, Cybernetics and Man*, vol. 1, pp. 223–229.

[45] H. Teicher, "Identifiability of mixtures," *Annals of Mathematical Statistics*, vol. 32, pp. 244–248, 1961.

[46] H. Teicher, "Identifiability of finite mixtures," *Annals of Mathematical Statistics*, vol. 34, pp. 1265–1269, 1963.

[47] D.M. Titterington, "Updating a diagnostic system using unconfirmed cases," *Applied Statistics*, vol. 25, pp. 238–247, 1976.

[48] D.M. Titterington, "An alternative stochastic supervisor in discriminant analysis," *Pattern Recognition*, vol. 22, pp. 91–95, 1989.

[49] D.M. Titterington, A.F.M. Smith and U.E. Makov, *Statistical Analysis of Finite Mixture Distributions*. London: Wiley, 1985.

[50] S. Watanabe, *Pattern Recognition: Human and Mechanical*. New York: Wiley, 1985.

[51] J.H. Wolfe, "A Monte Carlo study of the sampling distribution of the likelihood ratio for mixtures of multinormal distributions," Technical Bulletin, STB 72-2, National Personnel & Training Research Laboratories, San Diego, 1971.

[52] S.J. Yakowitz, "Unsupervised learning and the identification of finite mixtures," *IEEE Transactions on Information Theory*, vol. 16, pp. 330–338, 1970.

Chapter 3

ADAPTIVE STOCHASTIC ALGORITHMS FOR PATTERN CLASSIFICATION

M. A. L. Thathachar and P. S. Sastry

Department of Electrical Engineering
Indian Institute of Science
Bangalore 560012, INDIA
e-mail: *{malt,sastry}@ee.iisc.ernet.in*

Abstract

Adaptive stochastic algorithms for learning the decision rule that maximizes the probability of correct classification are discussed in this chapter. The class-conditional densities are assumed to be unknown and only samples with noisy class labels are provided for learning the decision rule. The algorithms are based on the concept of Learning Automata (LA) which are adaptive devices for learning the optimal action out of a given set of actions through repeated interactions with a random environment. Each action corresponds to the value of a parameter in the decision rule. A common payoff game of LA and a 3-layer network of LA are considered for learning the parameters of the optimum decision rule. Algorithms which converge to the global optimum are also described. Modules of LA which enable learning algorithms to run in parallel and consequently improve the speed of learning almost linearly with the number of units in the module, are discussed.

3.1 Introduction

In this chapter we address the problem of learning optimal decision rules for classifying patterns. We discuss some adaptive stochastic algorithms for finding the decision rule that maximizes probability of correct classification. All these algorithms are based on Learning Automata (LA) models [12]. The LA algorithms that we consider are all essentially optimization algorithms for finding a maximum or a minimum of a regression functional based on noisy function measurements. Such an optimization is an important component of all learning problems (for example, see the discussion in [8]). The LA algorithms that we discuss here are useful both in pattern recognition (PR) [28, 30] and in learning concepts in the form of Boolean expressions [17, 18, 22].

In any pattern recognition (PR) problem, the objective is to classify any given input pattern into one of finitely many classes [13, 7]. The nature of input pattern as well as the classification labels to be output by the system depend on the specific application. For example, in OCR applications, the input pattern is a two-dimensional (grey-scale) image of a character symbol and the output label (also called the class label or class) is the name (or some other representation) of the character present in the image. In speech recognition systems, the input pattern is a (sampled version of the) time-varying voltage signal from a microphone and the output class label is the identity of the word (or any other speech unit) that compose the speech utterance. In a fingerprint-based identity verification system, input is the image of the fingerprint pattern of a person (along with some other claim of identity such as a name) and the output class label is binary specifying whether or not the person is who he claims to be. While the first two examples represent multiclass problems, the last example is a two class problem.

In this chapter, for simplicity of exposition, we will concentrate only on the two-class problem. Any two-class technique can be extended to tackle multi-class problems [7, 13, 30].

Any general PR system can be viewed as an implementation of a two-step procedure: feature extraction and classification. In the first step, the system extracts or determines some salient characteristics, called *features*, from the input pattern. In most cases the features are real (or integer) numbers. Thus, though the input pattern may be in some arbitrary representation (such as an image), after feature extraction each pattern is represented by a vector of real numbers, called the *feature vector.** Now the classification step is to assign

*In some applications it may be desirable that not all features are numeric-valued. For example, in concept learning problems in AI some of the features may be nonnumeric. While it is possible to arbitrarily encode such features (*e.g.*, color of an object) into a class label to each feature vector. Since we are considering only 2-class problems here, the classifier is a function that maps the set of feature vectors to the set $\{0,1\}$. Any such function is called a decision rule. A good classifier or a decision rule is one that maps a feature vector to a class label that most often corresponds to the class that the pattern represented by the feature vector belongs to.

The design of a PR system now involves two steps: design of a feature extractor and design of a classifier. Feature extraction is very much problem-specific. For example, the features that we would like to measure from a fingerprint image so as to be able to make the correct identification decision would, in general, be different from those needed for correctly identifying a character from the image of a printed page. However, after fixing the set of features, design of a classifier admits some general procedures. The techniques we discuss in this chapter are all meant for designing the classifier. Hence we start with formulating the problem of classifier design. For this, we shall be considering the pattern recognition (PR) problem in the statistical framework [7]. That is, we assume that the variations in the feature vectors corresponding to patterns of one class can be captured in terms of probability distributions which can be used for discriminating between feature vectors of different classes.

Let $\mathcal{X}$ be the feature space, that is, set of all feature vectors. If we are using n real-valued features then $\mathcal{X}$ would be $\Re^n$, the n-dimensional real Euclidean space. Let $\mathcal{Y} = \{0,1\}$ be the set of possible outputs of our classifier. Every classifier or decision rule is a function from $\mathcal{X}$ to $\mathcal{Y}$. Let $\mathcal{H}$ be the set of all possible classifiers or decision rules of interest. For a pattern or feature vector, $X \in \mathcal{X}$, let $y(X)$ denote its 'true' class label. It may be noted here that, in general, $y(X)$ would be a random variable. This is because the set of all possible feature vectors obtained from all patterns of one class is not necessarily disjoint with the set of feature vectors of the other class. Similar feature vectors can come from patterns of different classes (with different probabilities) and thus we can only talk about the probability of a feature vector belonging to one class or the other. Also note that this is more due to the specific set of features chosen in a given system than an inherent property of the underlying pattern spaces. For any $h \in \mathcal{H}$, $h(X)$ denotes the class label assigned to X by the classifier h. We can represent the goodness of the classifier h on pattern

some numbers, it may not be a satisfactory solution. The LA based algorithms that we describe in this chapter would be useful in such applications also [18]. Due to space limitations, we would not be discussing these aspects of the LA algorithms.

X using $\ell(h(X), y(X))$ where $\ell(\cdot,\cdot)$ is called a *loss function*. A simple loss function is given by: $\ell(x,y) = 0$ when $x = y$ and $\ell(x,y) = 1$ when $x \neq y$. This is called a 0–1 loss function. Since $\ell(h(X), y(X))$ denotes the loss suffered by h on pattern X, an overall figure of merit for any classifier h can be given by a functional, $F(\cdot)$, defined on $\mathcal{H}$ by

$$F(h) = E\,(1 - \ell(h(X), y(X))), \quad h \in \mathcal{H} \tag{3.1}$$

where E denotes expectation with respect to the joint probability distribution of X and $y(X)$. A good classifier would have high value for F. If we are using the 0–1 loss function, $F(h)$ would be the probability that $h(X) = y(X)$. That is

$$\begin{aligned} F(h) &= E\, I_{\{h(X)=y(X)\}} \\ &= \mathrm{Prob}[h(X) = y(X)], \end{aligned} \tag{3.2}$$

where I_A represents the indicator function of event A. Thus a h which results in the maximum value of $F(h)$ would be a classifier with maximum probability of correct classification, or, equivalently, minimum probability of misclassification. Though $F(h)$ gives the probability of correct classification with h, only when we use 0–1 loss function, the definition of $F(h)$ given by (3.1) is reasonable for many other types of loss functions.[†] We will discuss some of these later on in this section.

The next question is, how do we find a h to maximize $F(h)$? Define, $Q_0(X) = \mathrm{Prob}[y(X) = 0|X]$ and $Q_1(X) = 1 - Q_0(X)$. These are called posterior probabilities of classes. Define a classifier h by

$$\begin{aligned} h(X) &= 0 \ \text{ if } \ Q_0(X) > Q_1(X) \\ h(X) &= 1 \ \text{ if } \ Q_0(X) \leq Q_1(X). \end{aligned} \tag{3.3}$$

It is easy to see that this h is optimal in the sense that it would maximize F given by (3.2) and thus would obtain minimum probability of misclassification. Let $f_0(X)$ denote the probability density function of feature vectors belonging to class 0 and similarly $f_1(X)$. These are called class-conditional densities.

[†]In (3.1), we have implicitly assumed that $0 \leq \ell(\cdot,\cdot) \leq 1$, which is true for the 0–1 loss function. In general, if the loss function used does not satisfy this restriction, F as defined would not be the proper criterion to maximize. However, this general structure is applicable for any reasonable loss function. For example, if the loss function is positive but is not necessarily bounded above by unity, we could attempt to minimize expected value of loss function.

Define $p_0 = \text{Prob}[y(X) = 0]$ and $p_1 = 1 - p_0$. These are referred to as prior probabilities. Since these are unconditional probabilities, p_0 denotes the probability that a random pattern belongs to class 0 and thus indicates the fraction of the total number of patterns that are in class 0. Now, using Bayes theorem of conditional probabilities, we have $Q_0(X) = f_0(X)p_0$ and similarly for Q_1. Define $g_B(X) = f_0(X)p_0 - f_1(X)p_1$. Now we can rewrite (3.3) as

$$\begin{aligned} h(X) &= 0 \text{ if } g_B(X) > 0 \\ h(X) &= 1 \text{ otherwise.} \end{aligned} \tag{3.4}$$

The classifier given by (3.3) or equivalently by (3.4) is called the Bayes optimal classifier. In a real problem it would be very difficult to actually derive the Bayes optimal classifier because we do not know the class conditional densities or posterior probabilities. The main reason for this is that we generally have no information about the statistics of pattern classes (under the chosen feature vector representation). However, what we have are some examples or the so called training samples. A training sample is a set $\{(X_1, y_1), \ldots, (X_m, y_m)\}$ where each $X_i \in \mathcal{X}$ is a feature vector obtained from some pattern and y_i is the class label of that pattern. It may be noted here that the y_i is not obtained using some *special* classifier function on X_i. (If we have available such a function, we do not need to build a PR system for the problem!). It is obtained through a teacher (usually a human expert) classifying the example patterns. For example, in OCR application, we take some known characters (so that we can assign the y_i) and then obtain the corresponding feature vectors (X_i) by applying our feature extraction programs. Hence, depending on the selected set of features, the training sample may be noisy in the sense that two arbitrarily close X_i may have different y_i as labels. In addition, in complicated PR problems the human expert acting as the teacher may himself make mistakes occasionally, thus introducing additional noise. Now the problem of classifier design is to build a procedure that infers or learns 'optimal' decision rules for classification using only these training samples.

One approach to this problem is to assume that the form of the class-conditional probability density functions is known. For example we can assume that these are multidimensional Gaussians with unknown mean vectors and covariance matrices. Then, using some statistical estimation techniques, the unknown parameters of these density functions can be estimated from the given training samples [7]. Once the class-conditional densities and prior probabilities are estimated, we can use the estimated quantities in the function g_B and the resulting classifier given by (3.4) would be an approximation to the Bayes classifier. This is a method that is often employed in many applications though there are some problems with this approach. The class of density functions that can be efficiently estimated is somewhat restricted. In addition, it is often difficult to relate errors in density estimation to probability of error in classification, which is a more important performance measure.

In cases where we cannot easily guess even the form of the density function or when it is difficult to do efficient density estimation for any other reason, we need an alternative method. A popular approach is to use what is known as a discriminant function. Here we consider a family of classifiers given by

$$\begin{aligned} h(X) &= 0 \text{ if } g(W,X) > 0 \\ h(X) &= 1 \text{ otherwise.} \end{aligned} \tag{3.5}$$

where $g(W,X)$ is called a discriminant function which is parameterized by a parameter vector W. By choosing different values for W we get different classifiers in this family. Since each classifier h in this family is specified by a value of parameter vector W, we can denote the decision made on X by a generic classifier in this family by $h(W,X)$ which is same as $h(X)$ in (3.5). The design of a classifier in this approach proceeds as follows. We first fix a form for the discriminant function g parameterized by a suitable W. Then we obtain the optimal classifier in the family given by (3.5) by searching for an optimal W with respect to some criterion.

Many choices are possible for the form of a discriminant function. If the feature vector has n components given by $X = (x_1, \ldots, x_n)$ then we can choose $W = (w_0, w_1, \ldots, w_n)$ and use a linear discriminant function given by $g(W,X) = w_0 + \sum_{i=1}^{n} w_i x_i$. Another choice is to use a neural network as a discriminant function when X would be input to the neural network and W would correspond to all the weights in the network.

Now the next question is to specify the criterion function used to define our 'optimal' classifier. The obvious choice would be to use the figure of merit F defined by (3.1). Since each classifier h in our case is specified by the parameter vector W, we can think of F as a function of W as

$$F(W) = E\,(1 - \ell(h(W,X), y(X))), \tag{3.6}$$

ℓ being a suitable loss function.

Obtaining the optimal parameter vector, say, W^*, that maximizes F defined by (3.6) is not a standard optimization problem. Recall that E in (3.6) denotes expectation with respect to the *unknown* probability distributions of the feature vectors of different classes. Hence given a W we cannot even calculate $F(W)$! We have to somehow use the given training samples to solve this optimization problem. One way of doing this is to approximate the F in (3.6) by

$$\bar{F}(W) = \frac{1}{m} \sum_{i=1}^{m} (1 - \ell(h(W, X_i), y_i)), \tag{3.7}$$

where $\{(X_i, y_i), i = 1, \ldots, m\}$ is the set of training samples. If we can assume that the training samples are obtained in an *independent and identically distributed* (*i.i.d.*) manner and if the discriminant function g used in computing $h(W, X)$ and the loss function ℓ are well-behaved, then, using results from statistical learning theory [33], it can be shown that for large m, the maximizer of $\bar{F}$ would be close to the maximizer of F. The requirement that training samples are *i.i.d.* is a formalization of our intuitive notion that the training samples should be *representative* of the kind of patterns that our PR system is likely to encounter in practice. The conditions on g and ℓ needed to use the results of statistical learning theory are also fairly mild. Hence learning an optimal W by maximizing the function defined by (3.7) is a very popular method used in designing pattern classifiers.

There is another (but essentially equivalent) way of finding W that maximizes F given by (3.6). Suppose we have an infinite sequence of *i.i.d.* samples $\{(X_i, y_i), i = 1, 2, \ldots\}$. Then we can find maximizer of F given by (3.6) using only observed values of $\ell(X_i, y_i)$ through stochastic approximation type algorithms [4, 9]. However, in practice we have only a finite number of training samples. We can construct an infinite sequence out of this by uniformly sampling from the finite training set. In such a case, using stochastic approximation algorithms amounts to maximizing $\bar{F}$ given by (3.7) [32].

The next question that needs to be addressed is: what kind of optimization algorithms would be suitable for finding the optimal parameter vector, W^*? A simple technique which is used often is a gradient ascent algorithm using the gradient of the criterion function with respect to W.[‡] This would be an iterative algorithm to estimate the optimal parameter vector, W^*. Let $W(k)$

[‡]A crucial question here is whether the criterion function is differentiable with respect to W. We shall be considering this issue shortly.

denote this estimate at iteration k. Then the algorithm is specified by

$$\begin{aligned} W(k+1) &= W(k) \;+\; \eta\, \nabla_W \bar{F}(W(k)) \\ &= W(k) \;+\; \eta\, \frac{1}{m} \sum_{i=1}^{m} \nabla_W \ell(h(W(k), X_i), y_i) \end{aligned} \tag{3.8}$$

where $\nabla_W \bar{F}$ and $\nabla_W \ell$ are gradients of the criterion function $\bar{F}$ and loss function ℓ with respect to W respectively, and they are evaluated at the arguments shown. Here η is the step size for the optimization algorithm. The above is a simple deterministic algorithm that can find a local maximum of the criterion function $\bar{F}$.

As mentioned earlier, we could have also used a stochastic approximation-based algorithm. One such algorithm is

$$W(k+1) = W(k) \;+\; \eta_k\, \nabla_W \ell(h(W(k), X(k)), y(k)) \tag{3.9}$$

where, now, $(X(k), y(k))$ is the random example at iteration k. This is a stochastic algorithm because now $W(k+1)$ depends on the *random* example at iteration k. The step size η_k satisfies $\eta_k \geq 0$, $\sum \eta_k = \infty$ and $\sum \eta_k^2 < \infty$. This algorithm is called Robbins-Munro algorithm and there are many similar methods which are useful in finding a maximizer of F based on observed values of $\ell(h(W,X), y)$ on random (*i.i.d.*) samples (X, y) [4, 9].

To use either of the algorithms given by (3.8) and (3.9), we need $\bar{F}$ and hence ℓ to be differentiable. However, if we use the 0–1 loss function mentioned earlier, then ℓ is not differentiable. We could use other loss functions such as $\ell(a,b) = (a-b)^2$. However, in addition to ℓ being differentiable we also need $h(W, X)$ to be differentiable with respect to W. Since h represents a decision rule and hence is binary-valued, it would not be differentiable. Hence to use such gradient-based techniques, normally one uses a loss function given by $\ell(a,b) = (a-b)^2$ and defines the criterion function as $\bar{F}(W) = \frac{1}{m} \sum (1 - \ell(\bar{h}(W, X_i), y_i))$ where $\bar{h}$ is a continuous function of W. If we choose $\bar{h}$ to be a continuous function with range $[0,1]$, then the maximizer of this criterion function (using *i.i.d.* samples) can be shown to be a least mean square estimate of the posterior probability. That is, if W^* is the (global) maximum of this $\bar{F}$, then $\bar{h}(W, X)$ would be a good estimate of $Q_1(X)$ defined earlier. This is the approach that is followed in most neural network based algorithms for designing pattern classifiers.

However, the above method of obtaining W^* using a squared loss function and continuous $\bar{h}$ does not give us a classifier that minimizes the probability of misclassification [25]. As explained earlier, the maximizer of F defined by (3.6) (or its approximation, $\bar{F}$ given by (3.7)) with a 0–1 loss function would correspond to a classifier with minimum probability of misclassification. From the viewpoint of the users of a PR system, often probability of misclassification is a more meaningful performance measure. If we want to obtain classifiers with minimum probability of misclassification, then we need to have techniques for maximizing F (given by (3.6) with 0–1 loss function) without needing to calculate or estimate the gradient.

Such an alternative approach for optimization is provided by the Learning Automata models. These algorithms maintain, at each instant k, a probability distribution, say, $\mathbf{p}(k)$ over the parameter space. The parameter vector at instant k, $W(k)$, is a random realization of this distribution. The (noisy) value of the criterion function at this parameter value is then obtained and is used to update $\mathbf{p}(k)$ into $\mathbf{p}(k+1)$ by employing a learning algorithm. The objective of the learning algorithm is to make the process converge to a distribution that chooses the optimal parameter vector with arbitrarily high probability. Thus, here the search is over the space of probability distributions. The random choice of $W(k)$ from $\mathbf{p}(k)$ allows sufficient exploration of the parameter space. Unlike stochastic approximations or other gradient descent methods, here we do not need to explicitly estimate any gradient information. In the rest of this chapter we present a few such learning algorithms for pattern classification using learning automata.

3.2 Learning automata

In this section we briefly explain Learning Automata (LA) models. The reader is referred to [12] for more details.

A Learning Automaton is an adaptive decision making device that learns the optimal action out of a set of actions through repeated interactions with a random environment. At each instant the automaton chooses an action, from its set of actions, at random based on its current action probability distribution. In response, it receives a stochastic *reinforcement* signal, also called the *reaction* from the environment which is a random evaluation of the automaton's action by the environment. The optimal action for the automaton is the one that elicits highest expected value of reinforcement from the environment. However, the automaton has no knowledge of the probability distributions using which the environment generates the reinforcement and hence of expected values of reinforcement for different actions. The automaton uses the stochastic reinforcement received from the environment to modify its action probability distribution through a learning algorithm. The objective of the learning algorithm is to converge to an action probability distribution that assigns a probability arbitrarily close to unity for the optimal action. Fig. 3.1 shows a block diagram of LA. Based on the cardinality of the action set two kinds of learning automata are distinguished: Finite Action set Learning Automata (FALA) and Continuous Action set Learning Automata (CALA).

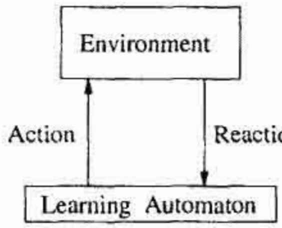


Fig. 3.1 A Learning Automaton in a random Environment

3.2.1 Finite action set learning automata

In a FALA the number of actions available is finite. Let $A = \{\alpha_1, \ldots, \alpha_r\}, r < \infty$, be the *set of actions* available. The automaton maintains a probability distribution over the action set. At each instant, k, the automaton chooses an *action* $\alpha_k \in A$, at random, based on its current *action probability distribution*, $\mathbf{p}(k) \in \Re^r$, $k = 0, 1, \ldots$. Thus, $\mathbf{p}(k) = [p_1(k) \ldots p_r(k)]^T \in \Re^r$, with $p_i(k) = \text{Prob}[\alpha(k) = \alpha_i], \forall k$. (It may be noted here that $\mathbf{p}(k)$ belongs to an r-dimensional *simplex* because $\sum_{i=1}^r p_i(k) = 1$, $\forall k$). For this choice of action the automaton receives from the environment a random *reaction* or *reinforcement*, $\beta(k)$. We have $\beta(k) \in R \subseteq [0, 1]$, $\forall k$, where R is the set of possible reactions from the environment. If $R = \{0, 1\}$ then the environment is called P-model; if $R = [0, 1]$ then it is called S-model; and if $R = [\beta_1, \ldots, \beta q\}$ then it is called Q-model. In all cases, higher values of the reinforcement signal are assumed more desirable. Let $\mathcal{F}_i$ be the distribution from which $\beta(k)$ is drawn when $\alpha(k) = \alpha_i$, $1 \leq i \leq r$. Let d_i denote the expected value of $\beta(k)$ given $\alpha(k) = \alpha_i$ (*i.e.*, the expected value of $\mathcal{F}_i$). Then d_i is called the *reward* *probability*[§] associated with action α_i, $1 \leq i \leq r$. Define the index m by $d_m = \max_i\{d_i\}$. Then the action α_m is called the optimal action.

In the above discussion, we have implicitly assumed that the distributions $\mathcal{F}_i$ and hence $d_i, 1 \leq i \leq r$, are not time varying and thus the identity of the optimal action is also not time varying. In this case the environment is said to be stationary. If the distribution of the random reaction from the environment for a given choice of action is time-varying then the environment is said to be nonstationary. In this section we consider only stationary environments. In the next two sections where LA systems are used for pattern classification we will be considering some nonstationary environments.

The learning automaton has no knowledge of the distributions $\mathcal{F}_i$ or of the reward probabilities. The objective for the automaton is to identify the optimal action; that is, to evolve to a state where the optimal action is chosen with probability arbitrarily close to unity. This is to be achieved through a learning algorithm that updates, at each instant k, the action probability distribution $\mathbf{p}(k)$ into $\mathbf{p}(k+1)$ using the most recent interaction with the environment, namely, the pair $(\alpha(k), \beta(k))$. Thus if T represents the learning algorithm, then, $\mathbf{p}(k+1) = T(\mathbf{p}(k), \alpha(k), \beta(k))$. We are interested in algorithms that make $p_m(k)$ converge to a value close to unity in some sense.

Definition 3.1 A learning algorithm is said to be *ϵ-optimal* if given any $\epsilon > 0$, we can choose parameters of the learning algorithm such that with probability greater than $1 - \epsilon$,

$$\liminf_{k \to \infty} \; p_m(k) > 1 - \epsilon.$$

■

We will be discussing ϵ-optimal learning algorithms here. We can characterize ϵ-optimality in an alternative way that captures the connection between learning and optimization. Define *average reward* at k, $G(k)$, by

$$\begin{aligned} G(k) &= E[\beta(k)|\mathbf{p}(k)] \\ &= \sum_i d_i p_i(k). \end{aligned} \tag{3.10}$$

[§]This name has its origin in P-model environments where d_i is the probability of getting a reward (*i.e.*, β=1) with action α_i.

Definition 3.2 A learning algorithm is said to be ϵ-optimal if, given any $\epsilon > 0$, it is possible to choose parameters of the algorithm so that

$$\operatorname*{Lim\ inf}_{k \to \infty} EG(k) > d_m - \epsilon$$

■

It is easily seen that the two definitions are equivalent. Thus, the objective of the learning scheme is to maximize the expected value of the reinforcement received from the environment. In the remaining part of this section we present some learning algorithms which are used later on.

3.2.1.1 *Linear reward inaction (L_{R-I}) algorithm*

This is one of the most popular algorithms used with LA models. This was originally described in mathematical psychology literature [5] but was later independently rediscovered and introduced with proper emphasis in [24].

Let the automaton choose action α_i at time k. Then $\mathbf{p}(k)$ is updated as:

$$\mathbf{p}(k+1) = \mathbf{p}(k) + \lambda(\mathbf{e}_i - \mathbf{p}(k))\beta(k) \tag{3.11}$$

where $0 < \lambda < 1$ is the step size parameter and $\mathbf{e}_i$ is a unit probability vector with i^{th} component unity and all others zero. To get an intuitive understanding of the algorithm, consider a P-model environment. When $\beta(k) = 1$, (*i.e.*, a *reward* from the environment), we move $\mathbf{p}(k)$ a little towards $\mathbf{e}_i$ when α_i is the chosen action, thus incrementing the probability of choosing that action and decrementing all others. When $\beta(k) = 0$, (*i.e.*, a *penalty* from the environment), the probabilities are left unchanged. Hence the name of the algorithm.

L_{R-I} is known to be ϵ-optimal in all stationary random environments [12]. L_{R-I} is very simple to implement and it results in decentralized learning in systems consisting of many automata [21]. However, in such cases it can find only local maxima (see the next section) and it may converge rather slowly.

3.2.1.2 *Pursuit algorithm*

This belongs to the class of estimator algorithms [16, 29] that were originally proposed to improve the speed of convergence. This algorithm typically converges about 10 to 50 times faster than L_{R-I}.

This improvement in speed is bought at the expense of additional computation and memory requirements. Here, the automaton maintains, in addition to the action probabilities, two more vectors, $\mathbf{Z}(k) = [Z_1(k), \ldots, Z_r(k)]^T$ and $\mathbf{B}(k) = [B_1(k), \ldots, B_r(k)]^T$. $Z_i(k)$ and $B_i(k)$ represent, respectively, the number of times action α_i is chosen till k and the total amount of reinforcement obtained with α_i till k, $1 \leq i \leq r$. Then, a natural estimate of the reward probability of i^{th} action, d_i, is $\hat{d}_i(k) = B_i(k)/Z_i(k)$, which is used in the algorithm to update the action probabilities. The algorithm is specified below.

Let $\alpha(k) = \alpha_i$ and let $\beta(k)$ be the reinforcement at k. Then,

$$\begin{aligned} B_i(k) &= B_i(k-1) + \beta(k) \\ Z_i(k) &= Z_i(k-1) + 1 \end{aligned} \tag{3.12}$$

$$\begin{aligned} B_j(k) &= B_j(k-1)\ \forall j \neq i \\ Z_j(k) &= Z_j(k-1),\ \forall j \neq i, \end{aligned} \tag{3.13}$$

$$\hat{d}_i(k) = \frac{B_i(k)}{Z_i(k)}, \ \ i = 1, \ldots, r$$

Let the random index H be defined by

$$\hat{d}_H(k) = \max_i \{\hat{d}_i(k)\}$$

Then,

$$\mathbf{p}(k+1) = \mathbf{p}(k) + \lambda(\mathbf{e}_H - \mathbf{p}(k)) \tag{3.14}$$

where λ $(0 < \lambda < 1)$ is the step-size parameter and $\mathbf{e}_H$ is the unit probability vector with H^{th} component unity and all others zero. By the definition of the random index H, α_H is the current estimated best action and (3.14) biases $\mathbf{p}(k+1)$ more in favor of that action. Since the index H keeps changing as the estimation proceeds, the algorithm keeps pursuing the current estimated best action. A special feature of the algorithm is that the actual reinforcement, $\beta(k)$ does not appear in the updating of $\mathbf{p}(k)$. Hence $\beta(k)$ can take values in any bounded set unlike the case of L_{R-I} where $\beta(k)$ has to be in [0,1] to ensure that $\mathbf{p}(k+1)$ is a probability vector (see (3.11)). The pursuit algorithm and the other estimator algorithms are ϵ-optimal in all stationary environments.

3.2.2 Continuous action set learning automata

So far we have considered the LA model where the set of actions is finite. Here we consider LA whose action set is the entire real line. To motivate the model, consider the problem of finding the maximum of a function $f : \Re \rightarrow \Re$, given that we have access only to noisy function values at any chosen point. We can think of f as the probability of misclassification with a single parameter discriminant function. To use the LA model for this problem, we can discretize the domain of f into finitely many intervals and take one point from each interval to form the action set of the automaton [30] (see Section 3.3.2 below). We can supply the noisy function value (normalized if necessary) as the reinforcement. This can solve the optimization problem but only at a level of resolution which may be poor based on the coarseness of the discretization. Also, if we employ too fine a level of discretization, the resulting LA will have too many actions and the convergence rate will be poor.

A more satisfying solution would be to employ an LA model where the action set can be continuous. Such a model, called Continuous Action Set Learning Automaton(CALA) will be discussed in this subsection.

The action set of CALA is the real line. The action probability distribution at k is $N(\mu(k), \sigma(k))$, the normal distribution with mean $\mu(k)$ and standard deviation $\sigma(k)$. At each instant, the CALA updates its action probability distribution (based on its interaction with the environment) by updating $\mu(k)$ and $\sigma(k)$, which is analogous to updating the action probabilities by the FALA. As before, let $\alpha(k) \in \Re$ be the action chosen at k and let $\beta(k)$ be the reinforcement at k. Here, instead of reward probabilities for various actions, we now have a reward function, $f : \Re \rightarrow \Re$, defined by

$$f(x) = E[\beta(k) \mid \alpha(k) = x].$$

We shall denote the reinforcement in response to action x as β_x and thus $f(x) = E\beta_x$.

The objective for CALA is to learn value of x at which f attains a maximum. That is, we want the action probability distribution, $N(\mu(k), \sigma(k))$ to converge to $N(x_o, 0)$ where x_o is a maximum of f. However, we do not let $\sigma(k)$ to converge to zero to ensure that the algorithm does not get stuck at a nonoptimal point. So, we use another parameter, $\sigma_\ell > 0$, and keep the objective of learning as $\sigma(k)$ converging to σ_ℓ and $\mu(k)$ converging to a maximum of f. By choosing σ_ℓ sufficiently small, asymptotically CALA will choose actions sufficiently close to the maximum with probability sufficiently close to unity.

The learning algorithm for CALA is described below. Since the updating given for $\sigma(k)$ does not automatically guarantee that $\sigma(k) > \sigma_\ell$, we always use a projected version of $\sigma(k)$, denoted by $\phi(\sigma(k))$, while choosing actions. Also, CALA interacts with the environment through choice of two actions at each instant.

At each instant k, CALA chooses an $x(k) \in \Re$ at random from the normal distribution $N(\mu(k), \phi(\sigma(k)))$ where ϕ is the function specified below. Then it gets the reinforcement from the environment for the two actions: $\mu(k)$ and $x(k)$. Let these reinforcements be β_μ and β_x. Then the distribution is updated as follows:

$$\begin{aligned}
\mu(k+1) &= \mu(k) + \lambda \frac{(\beta_x - \beta_\mu)}{\phi(\sigma(k))} \; \frac{(x(k) - \mu(k))}{\phi(\sigma(k))} \\
\sigma(k+1) &= \sigma(k) + \lambda \frac{(\beta_x - \beta_\mu)}{\phi(\sigma(k))} \left[\left(\frac{(x(k) - \mu(k))}{\phi(\sigma(k))} \right)^2 - 1 \right] \\
&\quad + \lambda \{ C[\sigma_\ell - \sigma(k)] \}
\end{aligned} \tag{3.15}$$

where

$$\begin{aligned}
\phi(\sigma) &= \sigma_\ell \text{ for } \sigma \le \sigma_\ell \\
&= \sigma \text{ for } \sigma > \sigma_\ell
\end{aligned} \tag{3.16}$$

and

- λ is the step size parameter for learning $(0 < \lambda < 1)$,
- C is a large positive constant, and
- σ_ℓ is the lower bound on standard deviation as explained earlier.

As explained at the beginning of this subsection, this CALA can be used as an optimization technique without discretizing the parameter space. It is similar to stochastic approximation algorithms [4, 9] though here the randomness in choosing the next parameter value makes the algorithm explore better search directions. For this algorithm it is proved that with arbitrarily large probability, $\mu(k)$ will converge close to a maximum of $f(\cdot)$ and $\phi(\sigma(k))$ will converge close to σ_ℓ, if we choose λ and σ_ℓ sufficiently small [19, 20].

3.3 A common payoff game of automata for pattern classification

In this section and the next we will present several learning automata algorithms for pattern classification. Recall from Section 3.1 that we pose the pattern classification problem as follows. Let $g(W, X)$, where X is the feature vector and W is the parameter vector, be the discriminant function. We classify a pattern using the classification rule given by (3.5). The form of $g(\cdot, \cdot)$ is assumed known (chosen by the designer). The optimal value for the parameter vector is to be determined by making use of a set of (possibly noisy) *i.i.d.* samples patterns which are preclassified. We call these samples the training set. We are interested in learning W that maximizes

$$F(W) = E\, I_{\{y(X)=h(W,X)\}} \tag{3.17}$$

where E denotes expectation with respect to the joint distribution of X and $y(X)$, and h is a classifier (which is completely specified by the parameter vector W) as defined in (3.5). As explained in Section 3.1, $F(W)$ is the probability of correct classification with classifier W. F is defined over $\Re^n$ if there are n parameters and we are interested in finding a W that globally maximizes F.

Recall, from Section 3.1, that the relevant probability distributions are unknown and hence the expectation in (3.17) cannot be evaluated. We need to find the maximizer of F using only a training set of patterns that are available. The training set consists of pairs $(X, \bar{y}(X))$ where X is a feature vector and $\bar{y}(X)$ is the class label for X *as given* in the training set. Now define

$$\tilde{F}(W) = E\, I_{\{\bar{y}(X)=h(W,X)\}}. \tag{3.18}$$

Given the set of training patterns, we can evaluate $I_{\{\bar{y}(X)=h(W,X)\}}$ for any classifier W and any sample pattern X. This is what we can use for finding a maximizer of $\tilde{F}$. Since the samples are *i.i.d.*, $\tilde{F}(W)$ will be the probability of correct classification with classifier W (that is, it will be same as $F(W)$) if the training set is *noise free*, that is, if $y(X) = \bar{y}(X)\ \forall X$. When there is noise, that is, if there are random mistakes in the classification of training samples, then the class label given for a training pattern X, $\bar{y}(X)$, would not be same as $y(X)$. (Now, the random variable $\bar{y}(X)$ is such that $\bar{y}(X) = y(X)$ with probability ρ and $\bar{y}(X) = 1 - y(X)$ with probability $1 - \rho$ where ρ is the probability of correct classification by the teacher). In this case, the $\tilde{F}(W)$ defined by (3.18) will only give the probability that the classification of a random pattern by the system (that is, by the classifier W) agrees with that of the (noisy) teacher. But we want to actually maximize the probability of correct classification, $F(W)$. Let ρ be the probability of correct classification by the (noisy) teacher (which is assumed to be independent of the class to which the pattern belongs). Then,

$$\begin{aligned} F(W) &= \rho E\, I_{\{\bar{y}(X)=h(W,X)\}} + (1-\rho)E\,(1 - I_{\{\bar{y}(X)=h(W,X)\}}) \\ &= (2\rho - 1)E\, I_{\{\bar{y}(X)=h(W,X)\}} + (1-\rho) \end{aligned} \tag{3.19}$$

Thus, as long as $\rho > 0.5$, $\tilde{F}(\cdot)$ and $F(\cdot)$ have the same maxima and hence it is sufficient to maximize $\tilde{F}$.

In the above we have assumed a uniform classification noise. That is, the probability of the teacher correctly classifying a training set pattern is the same for all patterns. Some of the automata algorithms discussed here can also handle the more general case where the probability of teacher correctly classifying X is $\rho(X)$, as long as $\rho(X) > 0.5$, $\forall X$, for certain classes of discriminant functions [11].

3.3.1 Common payoff game of LA

As briefly outlined in Section 3.2.2, a single automaton is sufficient for learning the optimal value of one parameter. But for multidimensional optimization problems we need a system consisting of as many automata as there are parameters. Here we consider the case where these automata are involved in a cooperative game.

Let $A_1, \ldots, A_N$ be the automata involved in an N-player game. Each play of the game consists of each of the automata players choosing an action and then getting the *payoffs* (reinforcement) from the environment for this choice of actions by the team. The game we consider is a common payoff game and hence all players get the same payoff. Let $\mathbf{p}_1(k), \ldots, \mathbf{p}_N(k)$ be the action probability distributions of the N automata. Then, at each instant k, each of the automata, A_i, chooses an action, $\alpha^i(k)$, independently and at random according to $\mathbf{p}_i(k)$, $1 \le i \le N$. This set of N actions is input to the environment which responds with a random payoff, $\beta(k)$ which forms the common reinforcement to all automata. The objective for the team is to maximize the expected payoff.

If $A_1, \ldots, A_N$ have all finite action sets then we call the expected value of the common reinforcement or payoff for a specific choice of actions by the team, as the reward probability for that choice of actions. In this case we can represent the reward probabilities as a hyper-matrix $D = [d_{j_1 \ldots j_N}]$ of dimension

$r_1 \times \ldots \times r_N$, where

$$d_{j_1 \ldots j_N} = E[\beta(k) \mid \alpha^i(k) = \alpha^i_{j_i},\ 1 \le i \le N] \tag{3.20}$$

Here $\{\alpha^i_1, \ldots, \alpha^i_{r_i}\}$ is the set of actions of automaton, A_i, $1 \le i \le N$. D is called the reward probability matrix of the game and it is unknown to the automata. The automata are to evolve to the optimal set of actions through multiple interactions with the environment and updating of their action probability vectors using a learning algorithm. For this case of a game of finite action set automata, the action $\alpha^i_{m_i}$ is the optimal action of automaton A_i, $1 \le i \le N$, if

$$d_{m_1 \ldots m_N} = \max\{d_{j_1 \ldots j_N}\} \tag{3.21}$$

where the maximum is over all possible values of the indices. It may be noted here that for any single automaton in the team, the environment is non-stationary. This is because the reinforcement to any automaton depends also on the choice of actions by the other automata.

We can consider common payoff game played by CALA also. Then the action set of each automaton is the real line. Now the expected value of reinforcement for a specific choice of actions by the team can be represented by the function

$$d(x_1, \ldots, x_N) = E[\beta(k) \mid \alpha^i(k) = x_i,\ 1 \le i \le N] \tag{3.22}$$

Here, $d(\cdot, \ldots, \cdot)$ is a function from $\Re^N$ to $\Re$, and we are considering a maximization problem over N-dimensional real Euclidean space. The objective for the automata team again is to find a maximum of d using a learning algorithm. It may be noted again that the automata have no knowledge of the function d and all they get from the environment is the common reinforcement, β (whose expected value for a given choice of actions equals the value of function d at the corresponding parameter values).

3.3.2 Pattern classification with finite action set LA

As explained earlier, for learning the optimal classifier, we need to learn the optimal value of the parameter vector, $W = [w_1, \ldots, w_N] \in \Re^N$. Let $w_i \in V^i \subset \Re$. In any specific problem, knowledge of the parametric form chosen for the discriminant function and knowledge of the region in the feature space where the classes cluster, is to be utilized for deciding on the sets V^i. Since actions of automata are parameter values and each automaton is capable of learning its optimal action, we need a team of N automata for learning optimal values of N parameters. In this section we are concerned with FALA and each automaton can have only finitely many actions. Hence we need to discretize the sets V^i to come up with action sets for the automata. Partition each of the sets V^i into finitely many intervals V^i_j, $1 \leq j \leq r_i$. Choose one point, v^i_j, from each interval V^i_j, $1 \leq j \leq r_i$, $1 \leq i \leq N$. Let the N automata in the team be denoted by $A_1, \ldots, A_N$. The action set of i^{th} automaton will be $\{v^i_1, \ldots, v^i_{r_i}\}$. Thus the actions of i^{th} automaton are the possible values for the i^{th} parameter, which are finitely many due to the process of discretization.

Now consider the following common payoff game played by these N automata. At each instant k, each automaton A_i chooses an action $\alpha^i(k)$ independently and at random according to its action probabilities, $\mathbf{p}_i(k)$. Since actions of automata are possible values for parameters, this results in the choice of a specific parameter vector, say $W(k)$ by the automata team. The environment classifies the next sample pattern using this parameter vector, and the correctness or otherwise of this classification is supplied to the team as the common reinforcement, $\beta(k)$. Specifically

$$\begin{aligned}\beta(k) &= 1 \quad \text{if } h(W(k), X(k)) = \bar{y}(X(k)) \\ &= 0 \quad \text{otherwise}\end{aligned} \tag{3.23}$$

where $X(k)$ is the sample pattern at k.

It is easy to see from (3.18) and (3.23) that the expected value of the common payoff to the team at k is equal to $\tilde{F}(W(k))$ where $W(k)$ is the parameter vector chosen by the team at k. Now it follows from (3.19), (3.20), (3.21) and (3.23) that the optimal set of actions for the team (corresponding to the maximum element in the reward matrix) is the optimal parameter vector that maximizes the probability of correct classification. Now what we need is a learning algorithm for the team which will make each automaton in the team converge to its optimal action. We will see below that each of the algorithms for a single automaton specified in Section 3.2 can easily be adapted to the team problem.

Before proceeding further, it should be noted that this method (in the best case) would only converge to the classifier that is optimal from among the *finitely* many classifiers in the set $\prod_{i=1}^{N} V^i$. This can only be an approximation to *the* optimal classifier due to the inherent loss of resolution in the process of discretization. While this approximation can be improved by finer discretization, it can result in a large number of actions for each automaton and consequently, slow rate of convergence. One can also improve the precision in the learnt classifier by progressively finer discretization. That is, we can first learn a rough interval for the parameter and then can choose the V^i set as this interval and further subdivide it and so on. However, the method is most effective in practice mainly in two cases: when there is sufficient knowledge available regarding the unknown parameters so as to make the sets V^i small enough intervals or when it is sufficient to learn the parameter values to a small degree of precision. Since we impose no restrictions on the form of the discriminant function $g(\cdot,\cdot)$, we may be able to choose the discriminant function so as to have some knowledge of the sets V^i. In Section 3.3.3 we will employ a team of CALA for solving this problem where no discretization of parameter ranges would be necessary.

3.3.2.1 L_{R-I} *algorithm for the team*

The Linear Reward Inaction algorithm presented in Section 3.2.1.1 is directly applicable to the automata team. Each automaton in the team uses the reinforcement that is supplied to it to update its action probabilities using (3.11). This will be a decentralized learning technique for the team. No automaton needs to know the actions selected by other automata or their action probabilities. In fact each automaton is not even aware that it is part of a team because it is updating its action probabilities as if it were interacting alone with the environment. However, since the reinforcement supplied by the environment depends also on the actions selected by others, each automaton experiences a non-stationary environment.

In a common payoff game, if each automaton uses an L_{R-I} algorithm with sufficiently small step size, then the team will converge with arbitrarily high probability to a set of actions that is a *mode* of the reward matrix. The concept of a mode is defined below.

Definition 3.3 The set of actions, $\alpha^i_{j_i}$, $1 \leq i \leq N$, is called a mode of the reward matrix if the following inequalities hold simultaneously.

$$\begin{aligned}
d_{j_1 \ldots j_N} &\geq \max_t \{d_{t\, j_2 \ldots j_N}\} \\
d_{j_1 \ldots j_N} &\geq \max_t \{d_{j_1\, t \ldots j_N}\} \\
&\;\;\vdots \\
d_{j_1 \ldots j_N} &\geq \max_t \{d_{j_1 \ldots j_{N-1}\, t}\}
\end{aligned} \tag{3.24}$$

where the maximum is over all possible values for the index, t. ∎

The mode is a Nash equilibrium in the common payoff game. In our case, from the point of view of optimization, it amounts to a local maximum. If the reward matrix of the game is unimodal then the automata team using the L_{R-I} algorithm will converge to the optimal classifier. For example, if the class conditional densities are normal and if the discriminant function is linear, then the game matrix would be unimodal. Another example where the automata team with L_{R-I} algorithm is similarly effective is that of learning simple conjunctive concepts [17, 22]. However, in general, with this algorithm the team can converge only to a local maximum of $F(\cdot)$ defined by (3.17) and, depending on the specific application, it may or may not be acceptable [21].

3.3.2.2 *Pursuit algorithm for the team*

We can adopt the Pursuit Algorithm presented in Section 3.2.1.2 for the automata team problem. However, if each automaton simply uses its reinforcement to estimate its effective reward probabilities, the algorithm will not work because each automaton experiences a nonstationary environment. We need to keep an estimated reward probability matrix from which each automaton can derive a quantity which is analogous to the $\hat{d}_i$ used in the pursuit algorithm in Section 3.2.1.2. The complete algorithm is given below. We use hypermatrices B and Z for updating the estimated reward probability matrix. The vector $\hat{E}^i$ here serves for automaton A_i the same purpose as the vector $\hat{d}$ for the algorithm in Section 3.2.1.2.

Let $\alpha^i(k)$, the action chosen by i^{th} automaton at k, be $\alpha^i_{j_i}$, $1 \leq i \leq N$, and let $\beta(k)$ be the reinforcement. Then

$$\begin{aligned} B_{j_1 \ldots j_N}(k) &= B_{j_1 \ldots j_N}(k-1) + \beta(k) \\ Z_{j_1 \ldots j_N}(k) &= Z_{j_1 \ldots j_N}(k-1) + 1 \end{aligned} \tag{3.25}$$

$$\begin{aligned} B_{i_1 \ldots i_N}(k) &= B_{i_1 \ldots i_N}(k-1),\ \forall (i_1 \ldots i_N) \neq (j_1 \ldots j_N) \\ Z_{i_1 \ldots i_N}(k) &= Z_{i_1 \ldots i_N}(k-1),\ \forall (i_1 \ldots i_N) \neq (j_1 \ldots j_N), \end{aligned} \tag{3.26}$$

$$\hat{d}_{i_1 \ldots i_N}(k) = \frac{B_{i_1 \ldots i_N}(k)}{Z_{i_1 \ldots i_N}(k)},\ \ \forall (i_1 \ldots i_N)$$

In the above equations the indices i_s range over 1 to r_s, $1 \leq s \leq N$. For $1 \leq i \leq N$, update the vectors E^i by

$$\hat{E}^i_\ell = \max_{t_s, 1 \leq s \leq N, s \neq i} \{\hat{d}_{t_1 \ldots t_{i-1}\, \ell\, t_{i+1} \ldots t_N}\},\ 1 \leq \ell \leq r_i.$$

Let the random indices $H(i)$, $1 \leq i \leq N$ be defined by

$$\hat{E}^i_{H(i)}(k) = \max_t \{\hat{E}^i_t(k)\}$$

Then, the action probabilities are updated as:

$$\mathbf{p}_i(k+1) = \mathbf{p}_i(k) + \lambda(\mathbf{e}_{H(i)} - \mathbf{p}_i(k)),\ \ 1 \leq i \leq N, \qquad (3.27)$$

where λ is the step size parameter $(0 < \lambda < 1)$ and $\mathbf{e}_{H(i)}$ is the unit vector with $H(i)^{th}$ component unity and all others zero.

It is proved in [30, 31] that the automata team employing this algorithm converges to the optimal set of actions even if the game matrix is not unimodal. Thus the automata team with pursuit algorithm learns the globally optimal classifier.

As is easy to see, this algorithm is not decentralized unlike the L_{R-I} algorithm. To maintain the estimated reward probability matrix, we need to know the actions chosen by all the automata at that instant. The algorithm is computationally not very intensive. Only one element of the estimated reward probability matrix changes at each instant and hence one can make obvious computational simplifications while updating the vectors $\hat{E}^i$. However, as the dimensionality of the problem increases, the memory overhead becomes severe due to the need to store the estimated reward probability matrix. However, this algorithm will make the team converge to the maximum element in the reward probability matrix in any general game with common payoff and thus ensure convergence to the global maximizer of probability of correct classification.

3.3.3 Pattern classification with CALA

We use a team of N continuous action set learning automata, $A_1, \ldots, A_N$, for learning an N-dimensional parameter vector. The actions of the automata will be possible values for the parameters. Since the action set of a CALA is the real line, we need not discretize the parameter space. Each automaton will be using a normal distribution for the action probability distribution as described in Section 3.2.2. The N automata will independently choose actions resulting in the choice of a parameter vector by the team. As in the previous section, we classify the next sample pattern with the classifier specified by the chosen parameter vector and supply a 1/0 reinforcement to the team depending on whether the classification agrees with that of the teacher or not. Each of the automata use the algorithm described in Section 3.2.2 to update the action probability distribution. This is once again a completely decentralized learning scheme for the team. For the algorithm described by (3.15) in Section 3.2.2, at each instant the automaton needs to interact with the environment twice. The same situation holds for the team also and thus we classify each pattern with two different parameter vectors to supply the two reinforcement signals needed.

It is proved that the team will converge (with arbitrarily large probability) to a parameter vector that is arbitrarily close to a local maximum of the function $F(\cdot)$ defined by (3.17) [19, 20].

3.3.4 Simulations

In this section we present results obtained with the automata team models presented earlier on some pattern classification problems. We present results with the Pursuit algorithm and with CALA. For the pursuit algorithm, we need to discretize the parameters and the algorithm finds the global maximum. In the case of CALA, convergence to only local maxima is assured but we need not discretize the parameters. As will be seen below, by proper choice of initial variance in the CALA algorithm we can obtain very good performance.

We present simulation results on only one problem. More details on empirical performance of these algorithms can be found in [23, 30]. Let $f_i(X), i = 0, 1$ denote the two class conditional densities. The prior probabilities of the two classes are assumed equal.

Example 3.1 The class conditional densities for the two classes are given by Gaussian distributions:

$$\begin{aligned}
f_0(X) &= N(\mathbf{m}_1, \Sigma_1) \\
\text{where } \mathbf{m}_1 &= [2.0\ ,\ 2.0]^T \\
\Sigma_1 &= \begin{bmatrix} 1.0 & -0.25 \\ -0.25 & 1.0 \end{bmatrix} \\
f_1(X) &= N(\mathbf{m}_2, \Sigma_2) \\
\text{where } \mathbf{m}_2 &= [4.0\ ,\ 4.0]^T \\
\Sigma_2 &= \begin{bmatrix} 1.5 & -0.25 \\ -0.25 & 1.5 \end{bmatrix}
\end{aligned}$$

For this problem, a quadratic discriminant function is considered. The form of the discriminant function is

$$\begin{aligned}
&g(W,X) \\
= \quad & \left[mx_2 + x_1 - (m^2+1)\left(x_0 - \tfrac{a}{(1+m^2)^{1/2}}\right)\right]^2 \tfrac{1}{1+m^2} \\
& - \left[x_1 - \left(x_0 + \tfrac{a}{(1+m^2)^{1/2}}\right)\right]^2 - \left[x_2 - m\ \left(x_0 + \tfrac{a}{(1+m^2)^{1/2}}\right)\right]^2
\end{aligned}$$

where $W = (m,\ x_0,\ a)$ is the parameter vector and $X = (x_1, x_2)$ is the feature vector. This is a parabola described by three parameters m, x_0 and a. A sketch of this parabola is shown in Fig. 3.2. This form of the equation chosen for the parabola makes it easier to visualize the parabola in terms of the three parameters. As mentioned earlier, in our method we can choose any form for the discriminant function. With the specific form chosen, it is easier to guess the ranges of the parameters based on some knowledge of where in the feature space the two classes cluster. It would be considerably more difficult to guess the parameter ranges if we had chosen a general quadratic expression for our discriminant function. It may be noted that the discriminant function is nonlinear in its parameters.

The parameters of the optimal discriminant function for this problem are: $m = 1.0$ $x_0 = 3.0$ $a = 10.0$. A sketch of the optimal discriminant function is shown in Fig. 3.3.

□

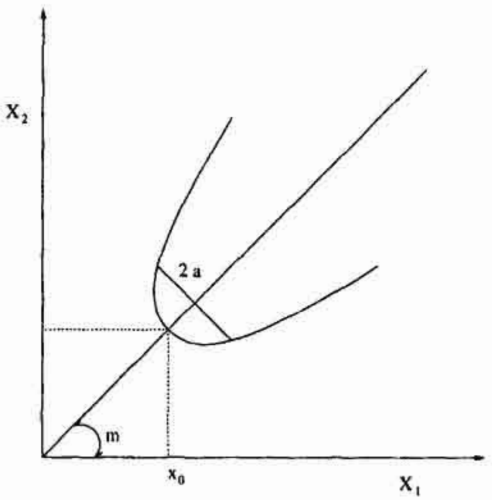


Fig. 3.2 The form of the discriminant function used in Example 3.1. It is a parabola specified by three parameters m, x_0 and a.

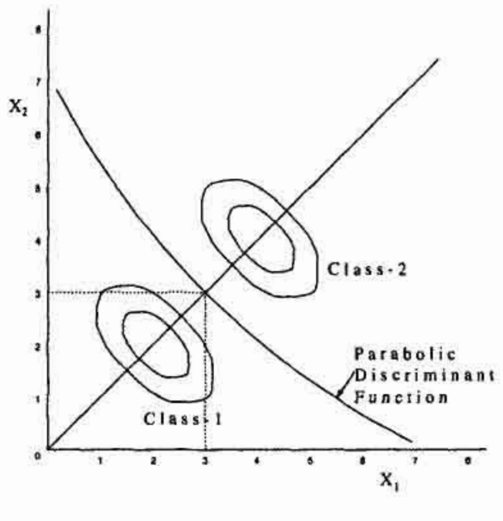


Fig. 3.3 Class conditional densities and the form of the Optimal discriminant function in Example 3.1.

3.3.4.1 *Learning automata team with pursuit algorithm*

For the simulation, 300 samples of each class are generated. At each instant one of the patterns from this set is selected at random and given to the learning system. The learning parameter, λ, was set at 0.1. A team of three automata was used. The ranges for the parameters, m, x_0 and a were taken to be [0.5, 1.5], [2, 6] and [1, 10] respectively. The range of each parameter was discretized into five levels. In seven out of ten experiments the team converged to the optimal parameters. In the other three runs, only one of the three automata converged to a wrong action. The average number of iterations needed for convergence was 1970.

3.3.4.2 *CALA team*

We used a *team of three continuous action set learning automata* (CALA). As earlier, 300 sample patterns are generated from each class. Also a test set consisting of 100 samples is generated from the two classes. In this problem it is difficult to analytically compute the minimum probability of misclassification. The number of misclassifications on the generated test set of patterns, with the parameter values set to those values corresponding to the optimal discriminant function (mentioned above), was found to be 11 (out of a total of 100 test patterns). The results of the simulations are provided in Table 3.1, for 3 different initial values. As can be seen from the results presented, with the initial values of the parameters the probability of misclassification is between 30% and 40%. However, after learning, the probability of misclassification is close to the best expected. Also, it may be noted that the team converges to a classifier whose probability of correct classification is close to global maximum, from many different starting points.

Table 3.1 Results obtained with CALA team on Example 3.1.

Initial Values								Final Value
μ_0^1	μ_0^2	μ_0^3	σ_0^1	σ_0^2	σ_0^3	% Error	# Iterations	% Error
1	0.4	6	6	6	4	32	3700	11
5	2	12	6	6	4	45	6400	12
4	2	12	6	6	4	42	3400	12

3.4 Three layer network consisting of teams of automata for pattern classification

In the previous section we have considered a common payoff game of automata and showed how it can be utilized for pattern classification. There we have assumed that the designer has decided on a parametric representation for the discriminant function, $g(W, X)$. The algorithm itself is independent of what this function is or how it is represented. For example, we could have represented it as an artificial neural network with parameters being the weights and then the algorithm would be converging to the 'optimal' set of weights. By making a specific choice for the discriminant function, we can configure the automata team more intelligently and this is illustrated in this section. We will only be considering teams of finite action learning automata here though the method can be extended to include CALA. The material in this section follows [14, 27, 28].

In a two-class PR problem, we are interested in finding a surface that appropriately divides the feature space which may be assumed to be a compact subset of $\Re^N$. Such a surface is well approximated by a piecewise linear function [10] and can be implemented using linear threshold units in a three-layer feedforward network. In this network the first layer units learn hyperplanes. Units in the second layer perform the *AND* operation on the outputs of *some selected* first layer units and thus learn convex sets with piecewise linear boundaries. The final layer performs an *OR* operation on the outputs of the second layer units. Thus this network, with appropriate choice of the internal parameters of the units and connections, can represent any subset of the feature space that is expressed as a union of convex sets with piecewise linear boundaries. The network structure is chosen because any compact subset of $\Re^N$ with piecewise linear boundary can be expressed as a union of such convex sets. We now describe how we can configure such a network with each unit being a team of automata.

Let the first layer consist of M units and let the second layer have L units. That means we can learn utmost M distinct hyperplanes and L distinct convex pieces. The final layer consists of a single unit. As before let $X(k) = [x_1(k), \ldots, x_N(k)]^T \in \Re^N$ be the feature vector.

Denote by U_i, $1 \leq i \leq M$, the units in the first layer each of which should learn a N-dimensional hyperplane. A hyperplane in $\Re^N$ can be represented by $(N + 1)$ parameters, namely, the normal vector and the distance from the origin to the hyperplane. Hence we will represent each unit, U_i, $1 \leq i \leq M$, by an $(N+1)$-member team of automata, A_{ij}, $0 \leq j \leq N$. The actions of automaton A_{ij} are the possible values of the j^{th} parameter of the i^{th} hyperplane being learnt. Since we are using finite action set automata here¶, as in Section 3.3.2, we discretize the ranges of parameters for making up the action sets of automata. Let $\tilde{A}_{ij}$ be the set of actions of automaton A_{ij} whose elements will be denoted by a_{ijs}, $1 \leq s \leq r_{ij}$, $0 \leq j \leq N$, $1 \leq i \leq M$. Let $\mathbf{p}_{ij}(k)$ be the action probability vector of A_{ij} with components p_{ijs} and

$$\text{Prob}[\alpha_{ij}(k) = a_{ijs}] = p_{ijs}(k)$$

where $\alpha_{ij}(k)$ is the action chosen by the automaton A_{ij} at time k. The output of unit U_i at k is $y_i(k)$ where

$$\begin{aligned} y_i(k) &= 1 \quad \text{if} \ \sum_j \alpha_{ij}(k) x_j(k) > 0 \\ &= 0 \ \text{otherwise} \end{aligned} \tag{3.28}$$

where $X(k) = [x_1(k) \ldots x_N(k)]$ is the feature vector of the sample pattern at iteration k.

Let V_i be the i^{th} second layer unit that has connections with $n(i)$ first layer units, $1 \leq i \leq L$. The $n(i)$ first layer units that are connected to V_i are prefixed. Thus V_i can learn a convex set bounded by *utmost* $n(i)$ hyperplanes. The unit V_i is composed of a team of $n(i)$ automata B_{ij}, $1 \leq j \leq n(i)$, each of which has two actions: 0 and 1. The action probability distribution of B_{ij} at k can be represented by a single real number $q_{ij}(k)$ where

$$\text{Prob}[z_{ij}(k) = 1] = 1 - \text{Prob}[z_{ij}(k) = 0] = q_{ij}(k)$$

where $z_{ij}(k)$ is the action selected by B_{ij} at k. Let $a_i(k)$ be the output of V_i at instant k. $a_i(k)$ is the AND of the outputs of all those first layer units which are connected to V_i and are activated, i.e., $z_{ij}(k) = 1$. More formally,

$$\begin{aligned} a_i(k) &= 1 \ \text{if} \ y_j(k) = 1 \ \forall\, j,\ 1 \leq j \leq n(i), \ \text{such that} \ z_{ij}(k) = 1 \\ &= 0 \quad \text{otherwise.} \end{aligned} \tag{3.29}$$

¶We could also use a team of CALA for each unit in the first layer. We use a FALA team here to demonstrate (in Section 3.4.2) how one can introduce a perturbation term to L_{R-I} type algorithms so as to converge to the global optimum.

The third layer contains only one unit whose output is a boolean OR of all the outputs of the second layer units. Since here we want to learn a union of convex sets, no learning is needed in the third layer.

This network of automata functions as follows. At each instant k, all the automata in all the first and second layer units choose an action at random based on their current action probability vector. That is, in each U_i, each of the automata A_{ij} chooses an action $\alpha_{ij}(k)$ from the set $\tilde{A}_{ij}$ at random based on the probability vector $\mathbf{p}_{ij}(k)$. This results in a specific parameter vector and hence a specific hyperplane being chosen by each U_i. Then, based on the next pattern vector, $X(k)$, each unit U_i calculates its output $y_i(k)$ using equation (3.28). In each second layer unit V_i, all the $n(i)$ automata B_{ij} choose an action $z_{ij}(k)$ at random based on the probability $q_{ij}(k)$. Using these $z_{ij}(k)$ and the outputs of first layer units, each V_i would calculate its output $a_i(k)$ using (3.29). Using the outputs of the second layer units, the unit in the final layer will calculate its output which is 1 if any $a_i(k)$ is 1; and 0 otherwise. Let $Y(k)$ denote the output of the final layer unit which is also the output of the network. $Y(k) = 1$ denotes that the pattern is classified as Class-1 and $Y(k) = 0$ denotes that the pattern is Class-0. For this classification, the environment supplies a reinforcement $\beta(k)$ as

$$\begin{aligned} \beta(k) &= 1 \quad \text{if } Y(k) = \bar{y}(X(k)) \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{3.30}$$

where $\bar{y}(X(k))$ is the classification supplied for the current pattern $X(k)$ in the training set. $\beta(k)$ is supplied as the common reinforcement to all the automata in all the units and then all the automata update their action probability vectors using the L_{R-I} algorithm as below.

For each i, j, $0 \leq j \leq N$, $1 \leq i \leq M$, the probability vectors $\mathbf{p}_{ij}$ are updated as

$$\begin{aligned} p_{ijs}(k+1) &= p_{ijs}(k) + \lambda\beta(k)(1 - p_{ijs}(k)) \quad \text{if } \alpha_{ij}(k) = a_{ijs} \\ &= p_{ijs}(k)(1 - \lambda\beta(k)) \qquad \text{otherwise} \end{aligned} \tag{3.31}$$

For each i, j, $1 \leq j \leq n(i)$, $1 \leq i \leq L$, the probabilities q_{ij} are updated as

$$\begin{aligned} q_{ij}(k+1) &= q_{ij}(k) + \lambda\beta(k)(1 - q_{ij}(k)) \quad \text{if } z_{ij}(k) = 1, \\ &= q_{ij}(k)(1 - \lambda\beta(k)) \qquad \text{otherwise} \end{aligned} \tag{3.32}$$

Let $\mathbf{P}(k)$ denote the internal state of the network. This includes the action probabilities of all the automata in all the units. That is, it includes all $\mathbf{p}_{ij}$ and all q_{ij}. Define

$$f(\mathbf{p}) = E[\beta(k) \mid \mathbf{P}(k) = \mathbf{p}] \tag{3.33}$$

For this network of teams of automata, the learning algorithm given by (3.31) and (3.32) will make $\mathbf{P}(k)$ converge to a local maximum of $f(\cdot)$. Thus the action probabilities of all the automata will converge to values that would (locally) maximize the expected value of the reinforcement.

It is clear from the earlier discussion in Section 3.3 and from (3.30), that maximizing the expected reinforcement will result in maximizing the probability of correct classification. Thus this network will learn a classifier which locally maximizes the probability of correct classification.

3.4.1 Simulations

We consider two examples here to illustrate the three layer network presented above. The first one is an artificial problem while the second one is on a real data set. For the first one we consider a problem where the region in the feature space in which the optimal decision is Class 0, is a convex set with linear boundaries. Once again we consider only 2-dimensional feature vectors because it is easier to visualize the problem. Both these examples are from [28].

Example 3.2 The feature vectors from the environment arrive uniformly from the set $[0,1]\times[0,1]$. The discriminant function to be learnt is shown in Fig. 3.4. Referring to the figure, the optimal decision in region A is Class 0 and that in region B is Class 1. In region A,

$$\text{Prob}[X \in \text{Class } 0] = 1 - \text{Prob}[X \in \text{Class } 1] = 0.9,$$

and in region B

$$\text{Prob}[X \in \text{Class } 0] = 1 - \text{Prob}[X \in \text{Class } 1] = 0.1.$$

The discriminant function to be learnt is

$$[2x_1 - x_2 > 0] \text{ AND } [-x_1 + 2x_2 > 0]$$

where $X = (x_1 \;\; x_2)^T$ is the feature vector.

Since we need to learn only one convex set, the network is made up of two first-layer units, U_1 and U_2, and one fixed second-layer unit. The second-layer unit performs AND operation on the outputs of the first-layer

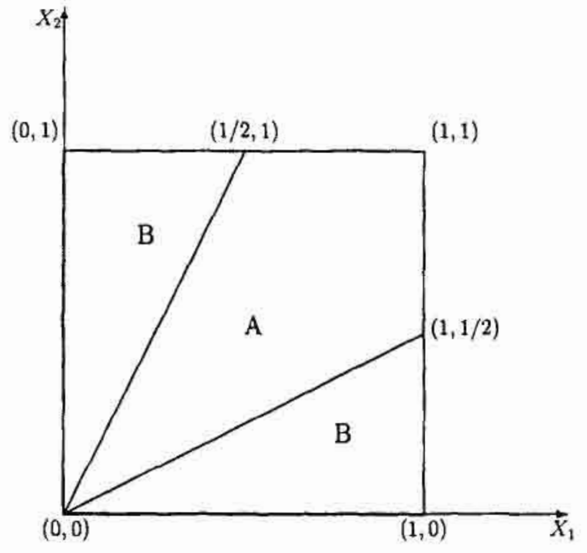


Fig. 3.4 The class regions in Example 3.2

units. Each first-layer unit has two automata. We used two rather than three because the hyperplanes to be learnt pass through origin. Each of the automata have four actions which are the possible values of the parameters to represent the hyperplanes. All four automata have the same action set given by {-2 -1 1 2}. In this problem, there are two sets of choices of actions by the four automata (or parameter vectors) given by (-1, 2, 2, -1) and (2, -1, -1, 2) at which the global optimum is attained. The learning parameter, λ, is fixed at 0.005 for all automata. The initial action probability distribution is uniform. That is, initial probability of each of the four actions is 0.25. The number of samples generated is 500 and at each instant one pattern chosen randomly from this set is presented to the network. Twenty simulation runs were conducted and the network converged to one of the two sets of optimal actions in every run. The average number of iterations needed for the probability of the optimal action to be greater than 0.98 for each automaton, is 10,922 steps. (Since the computations per iteration are very simple, the actual time taken is only a few seconds on a PC). □

Example 3.3 In this example, a 2-Class version of the Iris data [7] was considered. The data was obtained from the machine learning databases maintained at University of California, Irvine. This is a 3-class 4-feature problem. The three classes are *Iris setosa, Iris versicolor* and *Iris virginica.* Of these, *I. setosa* is linearly separable from the other two. Since we are considering only 2-Class problems here, *I. setosa* was ignored and the problem was reduced to that of classifying *I. versicolor* and *I. virginica.* The data used was 50 samples of each class with the correct classification.

The network consisted of 9 first layer units and 3 second layer units. Each first layer unit has 5 automata (since this is a 4-feature problem). Each automaton had 9 actions which were {-4,-3,-2,-1,0,1,2,3,4}. Uniform initial conditions were used. The learning parameters were 0.005 in the first layer and 0.002 in the second layer.

In this problem we do not know which are the optimal actions of the automata and hence we have to measure the performance based on the classification error on the data after learning.

For a comparison of the performance achieved by the automata network, we also simulated a standard feedforward neural network where we used backpropagation with momentum term (BPM) for the learning algorithm. The network has four input nodes (to take in the feature vector) and one output node. We tried two and three hidden layers. For the two hidden layer network we used 9 and 3 units in the hidden layers. For the three hidden layer network we used 8 nodes in each hidden layer. Initial weights for the network were generated randomly. In the learning algorithm the learning parameter for the momentum term was set at 0.9 and various values of the learning parameter for the gradient term were considered and the best results are reported here.

Simulations were conducted for perfect data (0% noise) and noisy cases. Noise was introduced by changing the known classification of the feature vector at each instant by a fixed probability. Noise levels of 20% and 40% were considered. With 40% noise, each sample has a probability of 0.4 of being misclassified.

The results obtained are summarized in Table 3.2. These are averages over 10 runs. The error reported in the table for the backpropagation algorithm is the root mean square error while that for the automata network is the probability of misclassification. While they cannot be directly compared, the performance was about the same at the values reported.

Table 3.2 Simulation Results for IRIS data. The entry in the fourth column refers to RMS error for BPM and probability of misclassification for L_{R-I}.

Algorithm	Structure	Noise(%)	Error	Steps
BPM	9 3 1	0	2.0	66,600
BPM	9 3 1	20	–	No Convergence
BPM	9 3 1	40	–	No Convergence
BPM	8 8 8 1	0	2.0	65,800
BPM	8 8 8 1	20	–	No Convergence
BPM	8 8 8 1	40	–	No Convergence
L_{R-I}	9 3 1	0	0.1	78,000
L_{R-I}	9 3 1	20	0.1	143,000
L_{R-I}	9 3 1	40	0.15	200,000

The results show that in the noise-free case, the backpropagation with momentum converges about 20% faster. However, this algorithm fails to converge even when only 20% noise is added. The learning automata network continues to converge even with 40% noise and there is only slight degradation of performance with noise.

□

3.4.2 A globally convergent algorithm for the network of automata

In the three-layer network of automata considered above, all automata use the L_{R-I} algorithm (*cf.* (3.31) and (3.32)). As stated earlier, one can establish only local convergence result for this algorithm. In this subsection we present a modified algorithm which leads to convergence to the global maximum.

One class of algorithms for automata team that result in convergence to global maximum are the estimator algorithms. As stated in Section 3.3.2.2, these algorithms have a large memory overhead. Here, we follow another approach, similar to the simulated annealing type algorithms for global optimization [1, 6], and impose a random perturbation in the update equations. However, unlike in simulated annealing type algorithms, here we keep the variance of perturbations constant and thus our algorithm would be similar to constant heat bath type algorithms. Since a FALA learning algorithm updates the action probabilities, introducing a random term directly in the updating equations is difficult due to two reasons. Firstly, it is not easy to ensure that the resulting vector after the updating remains a probability vector. Secondly, the resulting *diffusion* would be on a manifold rather than the entire space thus making the analysis difficult. To overcome such difficulties, the learning automaton is *parameterized* here. The automaton will now have an internal state vector, $\mathbf{u}$, of real numbers, which is not necessarily a probability vector. The probabilities of various actions are calculated based on the value of $\mathbf{u}$ using a *probability generating function*, $\bar{g}(\cdot,\cdot)$. The value of $\bar{g}(\mathbf{u},\alpha_i)$ will give the probability with which the i^{th} action is chosen by the automaton when the state vector is $\mathbf{u}$. Such learning automata are referred to as *Parameterized Learning Automata* (PLA) [14]. The probability generating function that we use is given by

$$p_i = \bar{g}(\mathbf{u},\alpha_i) = \frac{\exp(u_i)}{\sum_j \exp(u_j)} \tag{3.34}$$

where $\mathbf{u} = (u_1 \ldots u_r)^T$ is the state vector and $\mathbf{p} = (p_1 \ldots p_r)^T$ is the action probability vector.

We will now give the complete learning algorithm for the network of automata following the same notation that was used earlier. Thus p_{ijs} is the probability of s^{th} action, a_{ijs}, of automaton A_{ij} which is the j^{th} automaton in U_i, the i^{th} first layer unit and so on. The functioning of the network is same as before. However, the learning algorithm now updates the internal state of each automaton and the actual action probabilities are calculated using the probability generating function. Suppose $\mathbf{u}_{ij}$ is the state vector of automaton A_{ij} and has components u_{ijs}. Similarly, $\mathbf{v}_{ij}$ is the state vector of automaton B_{ij} and it has components v_{ij0} and v_{ij1}. Let $\bar{g}_{ij}(\cdot,\cdot)$ be the probability generating function for automaton A_{ij} and let $\tilde{g}_{ij}(\cdot,\cdot)$ be the probability generating function for automaton B_{ij}. As indicated by (3.34), the various action probabilities, p_{ijs} and q_{ij} are now given by

$$\begin{aligned} p_{ijs} &= \bar{g}_{ij}(\mathbf{u}_{ij}, a_{ijs}) = \tfrac{\exp(u_{ijs})}{\sum_s \exp(u_{ijs})} \\ q_{ij} &= \tilde{g}_{ij}(\mathbf{v}_{ij}, 1) \quad = \tfrac{\exp(v_{ij1})}{\exp(v_{ij1})+\exp(v_{ij0})} \end{aligned} \tag{3.35}$$

The algorithm given below specifies how the various state vectors should be updated. Unlike in (3.31) and (3.32), there is a single updating equation for all the components of the state vector. $\beta(k)$ is the reinforcement obtained at k, which is calculated as before by (3.30).

For each i, j, $0 \leq j \leq N$, $1 \leq i \leq M$, the state vectors $\mathbf{u}_{ij}$ are updated as

$$u_{ijs}(k+1) = u_{ijs}(k) + \lambda\beta(k)\frac{\partial \ln \bar{g}_{ij}}{\partial u_{ijs}} + \lambda h'(u_{ijs}(k)) + \sqrt{\lambda} s_{ij}(k) \quad (3.36)$$

For each i, j, $1 \leq j \leq n(i)$, $1 \leq i \leq L$, the state vectors $\mathbf{v}_{ij}$ are updated as

$$v_{ijs}(k+1) = v_{ijs}(k) + \lambda\beta(k)\frac{\partial \ln \tilde{g}_{ij}}{\partial v_{ijs}} + \lambda h'(v_{ijs}(k)) + \sqrt{\lambda} s_{ij}(k) \quad (3.37)$$

where

(i) The functions $\bar{g}_{ij}(\cdot,\cdot)$ and its partial derivatives are evaluated at

$$(\mathbf{u}_{ij}(k), \alpha_{ij}(k)),$$

the current state vector and the current action of A_{ij}. Similarly, the functions $\tilde{g}_{ij}(\cdot,\cdot)$ and its partial derivatives are evaluated at $(\mathbf{v}_{ij}(k), z_{ij}(k))$.

(ii) $h'(\cdot)$ is the derivative of $h(\cdot)$ which is defined by

$$\begin{aligned} h(x) &= -K(x - L_1)^{2n} \quad \text{for } x \geq L_1 \\ &= 0 \quad \text{for } |x| \leq L_1 \\ &= -K(x + L_1)^{2n} \quad \text{for } x \leq -L_1 \end{aligned} \quad (3.38)$$

where K and L_1 are real numbers and n is an integer all of which are parameters of the algorithm.

(iii) $\{s_{ij}(k)\}$ is a sequence of *i.i.d.* random variables (which also are independent of all the action probabilities, actions chosen etc.) with zero mean and variance σ^2. σ is a parameter of the algorithm.

In the updating equations given by (3.36) and (3.37), the $h'(\cdot)$ term on the right hand side (rhs) is essentially a *projection* term which ensures that the algorithm exhibits bounded behavior; and the s_{ij} term on the rhs adds a *random walk* to the updating. The $\beta(k)$ term on rhs is essentially the same updating as the L_{R-I} given earlier. To see this, it may be noted that

$$\begin{aligned} \frac{1}{\bar{g}_{ij}(\mathbf{u}_{ij}, a_{ijs})}\left[\frac{\partial \bar{g}}{\partial u_{ijs}}(\mathbf{u}_{ij}, a_{ijs})\right] &= 1 - p_{ijs} \\ \frac{1}{\bar{g}_{ij}(\mathbf{u}_{ij}, a_{ijs})}\left[\frac{\partial \bar{g}}{\partial u_{ijs'}}(\mathbf{u}_{ij}, a_{ijs})\right] &= -p_{ijs'} \end{aligned} \quad (3.39)$$

For this algorithm it is proved [27] that a continuous-time interpolated version of the state of the network, $\mathbf{U}$, given by the states of all automata, converges to a solution of the Langevin equation given by

$$d\mathbf{U} = \nabla\mathbf{H}(\mathbf{U}) + \sigma d\mathcal{W} \tag{3.40}$$

where

$$\mathbf{H}(\mathbf{U}) = E[\beta \mid \mathbf{U}] + \sum h(u_{ijs})$$

and $\mathcal{W}$ is the standard Brownian motion process of appropriate dimension.

As is well-known, the solutions of the Langevin equation concentrate on the global maximum of $\mathbf{H}$ as σ tends to zero. By the nature of the function $h(\cdot)$, this means the algorithm will converge to a state that globally maximizes the expected value of the reinforcement if the global maximum state vector is such that each component is less than L_1 in magnitude. Otherwise it will find a global maximum of $\mathbf{H}$ (which can be shown to be in a bounded region) and the expected value of reinforcement at this point would be greater than or equal to that inside the bounded region allowed for the algorithm. For more discussion and precise statement of this convergence result, the reader is referred to [27].

Simulations with the global algorithm

Here we briefly give results of simulations with this algorithm on one example considered earlier in Section 3.4.1, namely, Example 3.2.

In that example, we have seen that the global maximum is attained at two parameter vectors (-1,2,2,-1) and (2,-1,-1,2). One of the local maxima in that problem is given by (1,1,1,1). We have seen that the L_{R-I} algorithm converges to the global maximum when started with uniform initial conditions, that is, equal initial probabilities to all actions in all automata. Here we pick the initial conditions such that the effective probability of the parameter vector corresponding to the local maximum is greater than 0.98 and the rest of the probability is distributed among the other parameter vectors. With this much of bias, the L_{R-I} algorithm always converged to the local maximum.

We tried the globally convergent algorithm presented above with these initial conditions. The parameters in the $h(\cdot)$ function are set as L_1=3.0, K=1.0, and n=2. The learning parameter is set at 0.05. The value for σ is initially 10 and was reduced as

$$\sigma(k+1) = 0.999\sigma(k), \quad 0 \leq k \leq 5000.$$

and was kept constant thereafter. (Here $\sigma(k)$ is the value of σ used at iteration k).

As earlier the training set had 500 samples. Twenty simulations were done and each time the algorithm converged to one of the two global maxima. The average number of iterations needed for convergence was 33,425. This, of course, does not compare favorably with the time taken by L_{R-I}. In this algorithm, the computation time per iteration is also more than that of L_{R-I}. The extra time seems to be mainly due to the fact that the action probabilities are to be computed at each instant and not directly stored. The extra terms in the algorithm (the term for bounding the algorithm and the random term) do not seem to slow down the algorithm much. A different choice of the probability generating function may result in a faster algorithm. However, the higher computational effort and slower rates of convergence appear to be the price to be paid for convergence to global maximum in all annealing type algorithms.

3.5 Modules of learning automata

All the LA algorithms discussed so far in this chapter are essentially sequential in the sense that only one action is applied to the environment at a time and a single reaction or reinforcement elicited. (Though we use two actions at each instant in the CALA algorithm, the purpose there is to compare the reinforcement obtained for the randomly selected action with that obtained for the 'average' action). This sequential nature of the algorithm is one of the main reasons for slow rate of convergence of all LA algorithms. Hence one approach towards increasing speed of convergence is to use several LA in parallel in place of one LA and employ appropriate algorithms combining their operation. This essentially amounts to choosing several random actions at each instant, eliciting corresponding reinforcements from the environment and then combining all these reinforcements while updating the action probabilities. This strategy of choosing several actions at each time instant is feasible only in certain applications. It is, however, well suited for pattern classification. Here, as seen in the earlier sections, actions of automata are possible values for the parameter vector. Thus in the sequential algorithms discussed so far, we are obtaining a stochastic evaluation of only one classifier at each instant using the next pattern from the training set. In a parallel algorithm we would be evaluating several classifiers at each instant using the same training example.

Since the reinforcement from the environment is stochastic, a decision based on several responses would have less expected error in comparison with a decision based on a single response. The increments to the action probabilities effected by the learning algorithm would thus be more accurate and facilitate faster convergence. Such parallelization of LA learning algorithms is possible through the so called modules of LA [26]. In this section we briefly describe such modules of learning automata.

We first consider the parallel version of the L_{R-I} algorithm using a module of LA. Such a module of LA has the following structure.

- The action probability distribution is common to all LA members of the module.
- Each member of the module selects an action based on the common action probability distribution, independent of other members.
- The updating depends on actions selected by all members of the module, as well as the payoffs obtained by these actions.

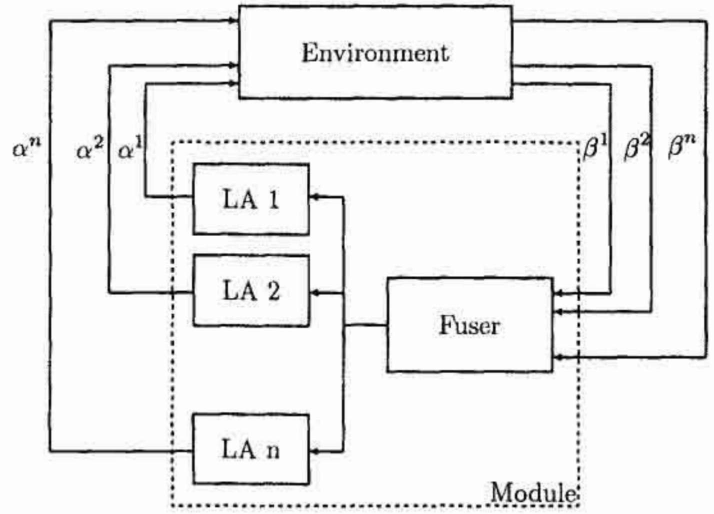


Fig. 3.5 Module of Learning Automata

The set up is shown in Fig. 3.5. (This figure may be compared with Fig. 3.1). It uses a module of n identical LA, with the common action set $\alpha \triangleq \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$. Each of them selects an action (independent of others) based on the common action probability distribution $\mathrm{p}(k) \triangleq [p_1(k), p_2(k), \ldots, p_r(k)]$. $\alpha^i(k)$ is the action selected by the ith module member at instant k, $\beta^i(k)$ denotes the corresponding payoff;

$$\alpha(k) \triangleq [\alpha^1(k), \alpha^2(k), \ldots, \alpha^n(k)]$$

denotes the action vector at k and

$$\beta(k) \triangleq [\beta^1(k), \beta^2(k), \ldots, \beta^n(k)]$$

denotes the corresponding payoff vector. It is assumed that $\beta^i(k) \in [0,1]\ \forall i, k$. The outputs of the environment are fed to a fuser that merges them suitably and supplies the required quantities to all the LA for updating the action probability distribution. The fuser at every instant computes

- Total payoff to α_i at k : $q_i(k) \triangleq \sum_{j=1}^{n} \beta^j(k) I_{\{\alpha^j(k)=\alpha_i\}}$.
- Total payoff at k : $q(k) \triangleq \sum_{j=1}^{n} \beta^j(k) = \sum_{i=1}^{r} q_i(k)$.

$\lambda \in (0,1]$ is the learning parameter for the algorithm and the quantity $\widetilde{\lambda} \triangleq \frac{\lambda}{n}$ is the normalized value of the learning parameter. I_A is the indicator function of even A which takes values 1 or 0. The actual algorithm is stated below.

$$p_i(k+1) = p_i(k) + \widetilde{\lambda}(q_i(k) - q(k)p_i(k)) \quad \forall i \in \{1, 2, \ldots, r\} \tag{3.41}$$

At any instant k, the expected fraction of choices of α_i is $p_i(k)$. The quantity $q_i(k)/q(k)$ could be considered as a figure of merit of the performance of α_i, and the update term could be regarded as moving $p_i(k)$ towards $q_i(k)/q(k)$.

The main advantage of the parallel algorithm (3.41) is that speed and accuracy can be independently controlled. Here, $\tilde{\lambda} = \frac{\lambda}{n}$ is the accuracy parameter which is chosen small enough to ensure, with a high probability, that the automaton converges to the optimal action. After choosing $\tilde{\lambda}$, n can be chosen large enough to get the required speed of convergence.

A common payoff game with several modules of LA can now be envisaged with one module replacing one LA. As in the case of common payoff game of a team of single LA considered earlier, one can show that for a common payoff game with unimodal reward matrix, the parallel version of the L_{R-I} algorithm is ϵ-optimal [26]. The speed of convergence can be increased by increasing n, the number of automata in each module. These results can be extended to networks of LA also and, in particular, to the three-layer network considered in Section 3.4.

To illustrate the improvement in speed of convergence through the use of parallel algorithms, we consider the iris data problem presented earlier (Example 3.3). We use the same data and the same 3-layer network structure used in Section 3.4.1 for Example 3.3. The results are presented in Fig. 3.6. We show the results obtained with no noise and when the training samples are corrupted by 10% classification noise. Modules with sizes 1,2 and 4 are used. Module with size 1 would correspond to the earlier algorithm. The figure shows the fractional error in classification averaged over ten runs as a function of the number of iterations. The algorithm was run in two phases; learning phase and the error computation phase. During the learning phase no error computation is performed. A sample pattern is chosen at random, and its classification is altered with probability 0.1 in the noisy case. Otherwise the given classification itself is maintained. The network of modules classifies the pattern using its internal action probability vectors and based on the payoffs obtained by each module, the probability vectors are updated. The error computation phase was performed once every 250 iterations. The probability vectors were not updated during this phase. A separate test set of 100 sample patterns was presented sequentially to the network. This was repeated over ten cycles and the fraction of patterns classified wrongly was computed. Repeated presentations are meant to average out the stochastic classification effects of the network to yield realistic error estimates. As in the learning phase, classification of the input pattern is altered with probability 0.1 in the noisy case.

The figures indicate the faster speed of convergence for the larger module sizes, with learning parameters chosen such that the limiting values of the error are approximately same for the noise free and noisy cases respectively. The chosen values for the limiting errors in the noise free and noisy cases were 0.1 and 0.2 respectively. The values of $(\widetilde{\lambda}_1, \widetilde{\lambda}_2)$, the learning parameters used in the first and second layer units respectively, are (0.005,0.002).

The basic idea behind the parallelization of LA algorithms is applicable for a large class of stochastic adaptive algorithms including CALA. More detailed discussion of these issues can be found in [26].

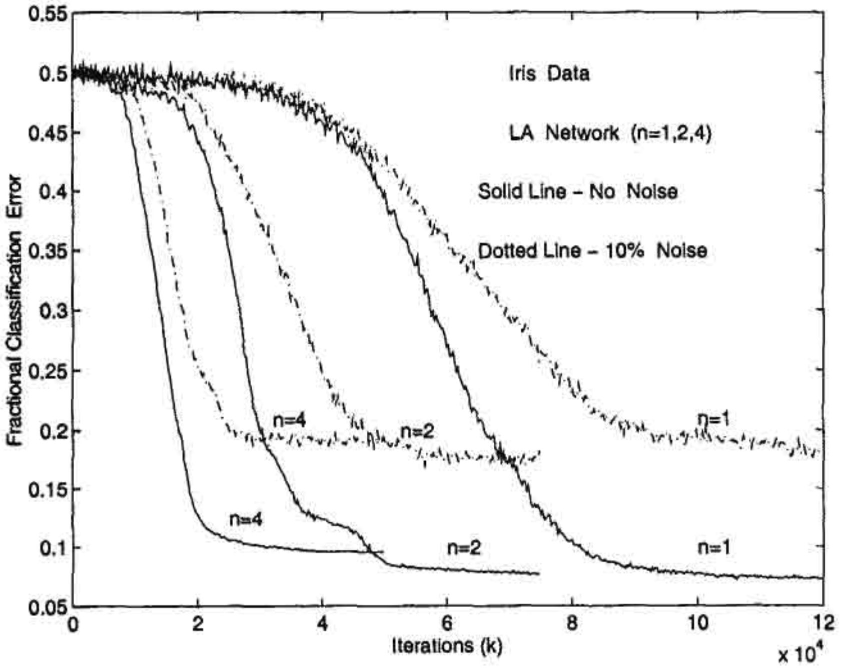


Fig. 3.6 Learning curves for classification of Iris data using modules of LA in a 3-layer network

3.6 Discussion

In this chapter, we have considered algorithms based on teams of learning automata for pattern classification. In all the algorithms, some parametric representation was chosen for the discriminant function and the objective was to learn the optimal values of parameters from the given set of preclassified training samples.

All the algorithms presented here can tackle only 2-class problems. The simplest method to solve an N-Class problem using 2-Class PR algorithms is to learn N 2-Class classifiers, each for distinguishing one class from the rest. Thus we make N training sets (each for a 2-class problem) from the given training set and successively learn the N classifiers. More detailed discussion of this issue can be found in [7, 30].

The criterion of optimality considered is maximizing probability of correct classification. As mentioned in Section 3.1, algorithms that minimize mean square error do not necessarily maximize probability of correct classification. The problem is to find a classifier, h, that maximizes $F(\cdot)$ given by (3.1) with a 0-1 loss function. Since the underlying probability distributions are unknown, this is a hard optimization problem. Given a classifier h and a random training sample (X, y), we can calculate $\ell(h(X), y)$; but we cannot calculate $F(h)$. Hence we have to solve a regression problem. However, there are two additional complications here. Even if h is parameterized by a real vector, $\ell(h(X), y)$ may not be differentiable. Hence algorithms (like stochastic approximations) that rely on estimating the gradient information by perturbation methods, are not likely to be robust. In addition, we may choose the structure of the classifier in such a way that it is not easy to define a gradient (*e.g.*, the three layer network of Section 3.4). The main strength of automata based algorithms (and other reinforcement learning methods) is that they do not explicitly estimate the gradient.

The essence of LA based methods is the following. Let $\mathcal{H}$ be the space of classifiers chosen. Then we construct an Automata system such that when each of the automata chooses an action from its action set, this tuple of actions corresponds to a unique classifier, say h, from $\mathcal{H}$. Then we give $1 - \ell(h(X), y)$ (which, for the 0-1 loss function, is simply correctness or otherwise of classifying the next training pattern with h) as the reinforcement. Since the automata algorithms guarantee to maximize expected reinforcement, with *i.i.d.* samples the system will converge to an h that maximizes $F(\cdot)$ given by (3.1). The automata system is such that its state, represented by the action probability distributions of all automata, defines a probability distribution over $\mathcal{H}$. It is this probability distribution that is effectively updated at each instant. Thus the automata techniques would be useful even in cases where $\mathcal{H}$ is not isomorphic to an Euclidean space (or when there is no simple algebraic structure on $\mathcal{H}$).

In the simplest case, if the classifier structure is a discriminant function determined by N real valued parameters then the actions of automata are possible values of the parameters and we employ a cooperating team of N automata involved in a common-payoff game. If we use the traditional (finite action set) learning automata then we have to discretize the parameter space, which may result in loss of precision. However, from the results obtained on the Iris data, it is easy to see that the automata algorithm, even with discretization of parameters, performs at a level comparable to other techniques such as feedforward neural nets in noise-free cases and outperforms such techniques when noise is present. We have also presented algorithms based on the recent model of continuous action set learning automata (CALA) where no discretization of parameters is needed.

The interesting feature of the automata models is that the actions of automata can be interpreted in many different ways leading to rich possibilities for representing classifiers. In the algorithms presented in Section 3.3, the actions of all automata are values of real-valued parameters. The discriminant functions (functions mapping $\Re^N$ to $\Re$) can be nonlinear in parameters also (as illustrated in the examples) since the form of the discriminant function does not affect the algorithm. In Section 3.4, another structure of automata was used to represent unions of convex sets. Here the actions of first level automata are real values denoting parameters to represent hyperplanes. The actions of second level automata are boolean decisions regarding which hyperplanes to pick to make convex sets. Here the discriminant function is essentially a boolean expression whose literals are simple linear inequalities. It is easy to see that a network structure like this will also be useful in learning decision tree classifiers. Another example of this flexibility is that the same models discussed in Section 3.3 can be used for concept learning where the features may be nonnumeric and the discriminant function is a logic expression [17, 18, 22].

All the automata algorithms presented here implement a probabilistic search over the space of classifiers. All the action probabilities of all the automata together determine a probability distribution over $\mathcal{H}$. At each iteration an $h \in \mathcal{H}$ is chosen (by each of the automata choosing an action) which is a random realization of this probability distribution. Then the reinforcement obtained is used, in effect, to tune this probability distribution over $\mathcal{H}$. This allows for a type of randomness in the search that helps the algorithms to generally converge to good parameter values even in presence of local optima. The three layer automata network delivers good performance on the Iris data even under 40% classification noise. The CALA algorithm also achieves good performance (see simulation results in Section 3.3.4) though theoretically only convergence to local maxima is assured. In the CALA algorithm, this is achieved by choosing a higher value of the initial variance for the action probability distribution which gives an initial randomness to the search process to explore the parameter space better.

We have also presented algorithms where convergence to global maximum is assured. The Pursuit algorithm allows a team of finite action set automata to converge to the global maximum of the reward matrix. However, this algorithm has a large memory overhead to estimate the reward matrix. One can trade such memory overhead for time overhead using a simulated annealing type algorithm. We presented automata algorithms that use a biased random walk in updating the action probability distributions (*cf.* Section 3.4.2) and here the automata team converges to the global maximum with a large probability. A similar modification is possible for the CALA algorithm also so that it can converge to the global maximum.

One of the main drawbacks of the LA based approach to pattern classification is the slow rate of convergence of automata algorithms. We have briefly indicated how we can increase speed of learning using the so called modules of LA. For many automata algorithms the speed of learning increases almost linearly with the number of units in the module.

There are other automata models that have been used for pattern classification. In all the models considered in this chapter, the actions of automata are possible values for the parameters. It is possible to envisage an alternative set-up where the actions of the automata are the class labels. However, in such a case, we need to allow for the pattern vector to be somehow input to the automata system. Hence we need to extend the automaton model to include an additional input which we shall call *context*. In the traditional model of learning automata (whether with finite action set or continuous action set), the automaton does not take any input other than the reinforcement feedback from the environment. Within this framework we talk of the optimal action of the automaton without reference to any *context*. For example, when actions of automata are possible values of parameters, it makes sense to ask which is the optimal action. However, when actions of automaton are class labels, one can talk of the optimal action only in the *context* of a pattern vector that is input. Here with different context inputs, different actions may be optimal and hence we should view the objective of the automaton as learning to associate the right action with each context. Such a problem has been called *associative reinforcement learning* and automata models with provision for a context input, called Generalized Learning Automata(GLA), are studied by many researchers [2, 3, 14, 34]. In contrast, the automaton considered in this chapter may be thought of as a model for nonassociative reinforcement learning. In a GLA, the action probabilities for various actions would also depend on the current context vector input. Thus, if X is the context input then the probability of GLA taking action y is given by $\bar{g}(X, W, y)$ where $\bar{g}(\cdot,\cdot,\cdot)$ is the action probability function of the GLA and W is a set of internal parameters. The learning algorithm updates the parameters W based on the reinforcement and the objective is to maximize the reinforcement over all context vectors. On account of the provision of the context input into the GLA, these automata can be connected together to form a network where outputs of some automata can form part of context input to other automata [14, 34]. There are learning algorithms for GLA that guarantee convergence to local as well as global maxima of the reinforcement function [14, 15]. These automata networks can be used for pattern classification very much like the models discussed in this chapter. All the automata algorithms appear to be particularly well-suited in situations with noisy features and also where class labels of training samples are noisy.

Acknowledgements

Some of the material in this chapter is reproduced from [30, 28, 23, 26] with permission from Indian Academy of Sciences and Institute of Electrical and Electronics Engineers (IEEE).

References

[1] F. Aluffi-Pentini, V. Parisi, and F. Zirilli, "Global optimization and stochastic differential equations," *Journal of Optimization Theory and Applications*, vol. 47, pp. 1–26, 1985.

[2] A. G. Barto and P. Anandan, "Pattern-recognizing stochastic learning automata," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 15, pp. 360–374, 1985.

[3] A. G. Barto, "Learning by statistical cooperation of self-interested neuron-like computing elements," COINS Tech. Rept. 81-11, Univ. of Massachusetts, Amherst, MA, Apr. 1985.

[4] B. Bharath and V. S. Borkar, "Stochastic approximation algorithms: Overview and recent trends," *Sadhana*, vol. 24, pp. 425–452, 1999.

[5] R. R. Bush and F. Mosteller, *Stochastic Models for Learning*. New York: John Wiley and Sons, 1958.

[6] T. Chiang, C. Hwang, and S. Sheu, "Diffusion for global optimization in $\Re^n$," *SIAM Journal of Control and Optimization*, vol. 25, pp. 737–753, 1987.

[7] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[8] D. L. Haussler, "Decision theoretic generalization of the PAC model for neural net and learning applications," *Information and Computation*,

vol. 100, pp. 78–150, 1992.

[9] H. J. Kushner and G. G. Yin, *Stochastic Approximation Algorithms and Applications*. New York: Springer-Verlag, 1997.

[10] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4–22, Apr. 1987.

[11] G. D. Nagendra, *PAC Learning with Noisy Samples*. ME thesis, Dept. of Electrical Engineering, Indian Institute of Science, Bangalore, India, Jan. 1997.

[12] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs: Prentice Hall, 1989.

[13] N. J. Nilsson, *Learning Machines*. New York: McGraw Hill, 1965.

[14] V. V. Phansalkar, *Learning Automata Algorithms for Connectionist systems - local and global convergence*. PhD thesis, Dept. of Electrical Engineering, Indian Institute of Science, 1991.

[15] V. V. Phansalkar and M. A. L. Thathachar, "Local and global optimization algorithms for generalized learning automata," *Neural Computation*, vol. 7, pp. 950–973, 1995.

[16] K. Rajaraman and P. S. Sastry, "Finite time analysis of pursuit algorithm for learning automata," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 26, pp. 590–599, 1996.

[17] K. Rajaraman and P. S. Sastry, "A parallel stochastic algorithm for learning logic expressions under noise," *Journal of the Indian Institute of Science*, vol. 77, pp. 15–45, 1996.

[18] K. Rajaraman and P. S. Sastry, "Stochastic optimization over continuous and discrete variables with applications to concept learning under noise," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 29, pp. 542–553, 1999.

[19] G. Santharam, *Distributed Learning with Connectionist Models for Optimization and Control*. PhD thesis, Dept. of Electrical Engineering, Indian Institute of Science, Bangalore, India, May 1994.

[20] G. Santharam, P. S. Sastry, and M. A. L. Thathachar, "Continuous action set learning automata for stochastic optimization," *Journal of the Franklin Institute*, vol. 331, pp. 607–628, 1994.

[21] P. S. Sastry, V. V. Phansalkar, and M. A. L. Thathachar, "Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, pp. 769–777, May 1994.

[22] P. S. Sastry, K. Rajaram, and S. R. Ranjan, "Learning optimal con-

junctive concepts using stochastic automata," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, pp. 1175–1184, 1993.

[23] P. S. Sastry and M. A. L. Thathachar, "Learning automata algorithms for pattern classification," *Sadhana*, vol. 24, pp. 261–292, 1999.

[24] I. J. Shapiro and K. S. Narendra, "Use of stochastic automata for parameter self optimization with multi-modal performance criteria," *IEEE Transactions on Systems Science and Cybernetics*, vol. , pp. 352–360, 1969.

[25] J. Sklansky and G. N. Wassel, *Pattern Classification and Trainable Machines*. New York: Springer-Verlag, 1981.

[26] M. A. L. Thathachar and M. T. Arvind, "Parallel algorithms for modules of learning automata," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, pp. 24–33, 1998.

[27] M. A. L. Thathachar and V. V. Phansalkar, "Learning the global maximum with parameterized learning automata," *IEEE Transactions on Neural Networks*, vol. 6, pp. 398–406, Mar. 1995.

[28] M. A. L. Thathachar and V. V. Phansalkar, "Convergence of teams and hierarchies of learning automata in connectionist systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, pp. 1459–1469, 1995.

[29] M. A. L. Thathachar and P. S. Sastry, "A new approach to the design of reinforcement schemes for learning automata," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 15, pp. 168–175, Jan. 1985.

[30] M. A. L. Thathachar and P. S. Sastry, "Learning optimal discriminant functions through a cooperative game of automata," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 17, pp. 73–85, 1987.

[31] M. A. L. Thathachar and P. S. Sastry, "Learning automata in stochastic games with incomplete information," in *Systems and Signal Processing* (R.N.Madan, N.Vishwanathan, and R.L.Kashyap, eds.), (New Delhi), pp. 417–434, Oxford and IBH, 1991.

[32] V. N. Vapnik, *Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1997.

[33] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, pp. 988–999, Sept. 1999.

[34] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

Chapter 4

UNSUPERVISED CLASSIFICATION: SOME BAYESIAN APPROACHES

A. Pal

Applied Statistics Unit
Indian Statistical Institute
Calcutta, INDIA
e-mail: *pamita@isical.ac.in*

Abstract

The utility of established methodology like clustering in the area of unsupervised classification is well-documented and well-acknowledged. However, if the problem of unsupervised classification is formulated as the statistical problem of resolution of finite mixtures of probability densities, then several Bayesian approaches become immediately relevant. Some of the more significant of these methodologies are surveyed here.

4.1 Introduction

It is a well-known fact that the ability of humans for recognizing patterns is mainly learnt from past experiences, though the procedure by which the human brain accomplishes this, is too complicated to be understood. Thus *learning* is an indispensable component of pattern recognizers, both human and mechanical. In the case of pattern recognition by machine, the equivalent of "past experiences" is a set of samples, called training or learning samples, from the pattern classes in question. The information contained in the training samples provides the basis for learning in pattern recognition systems. In some cases, learning is done with the help of a *teacher*, that is, an external agency of some sort that provides the correct *labels* or classification of the training samples provided for building the classifier. The training samples in such cases become representatives of the classes they belong to, and can be processed in a suitable manner so that the class-specific information they carry may be distilled from them. This is referred to as *supervised* pattern recognition. References [9, 8, 10, 33, 36, 37] are a few of the many books in which detailed information on this is available.

On the other hand, if no teacher is available for a pattern classification task, that is, the training samples are not *labeled*, then we have a case of *unsupervised* pattern recognition. This is true for many pattern recognition problems where it is difficult or expensive, perhaps even impossible, to obtain a reliable teacher for labeling the training samples correctly. In such cases, learning essentially means discovery of the natural groupings inherent in the training set. The most popular computational technique applicable to unsupervised classification is *cluster analysis*, for which there is no dearth of literature [2, 4, 5, 16, 18, 30].

Mixtures of probability distributions, particularly normal distributions, have been used widely as models in numerous practical problems where data can be visualized as arising from one of two or more populations mixed in varying proportions. In fact, they serve as very powerful and flexible tools for probabilistic modeling in a wide variety of problems. Titterington, Smith and Makov [35] give a comprehensive list of applications of such models, apart from discussing various aspects of their statistical analysis. It is not difficult to visualize the appropriateness of mixtures of probability distributions as models in the case of unsupervised classification. When an unsupervised classification problem is formulated in terms of a mixture model, issues of primary concern are the identifiability of the mixture, the determination of the number of components in case it is not known *a priori*, the estimation of the unknown parameters, and the goodness of the resulting fit.

The pattern recognition community has been aware of this approach for a long time, and has been quick to adopt techniques being developed by statisticians for analyzing mixture models. This area has been witnessing steady and significant research for the past few decades. McLachlan and Basford [19] give a detailed account of the progress made in this field till the mid-nineteen-eighties, beginning with the work of Karl Pearson in 1894 [22], in which he used the method of moments to estimate the parameters of the mixture of two univariate distributions with unequal variances. Pearson's work served to demonstrate the degree of difficulty involved in solving the problem of resolution of a finite mixture of distributions. Later, it was established by quite a few researchers that the method of moments is decidedly inferior to the method of maximum likelihood estimation for this particular problem. McLachlan and Basford [19] have highlighted the use of mixture models, fitted by the maximum likelihood approach, as a means of effective clustering of data in a variety of situations. Among other methods of estimation, they have also discussed the application of the Expectation Maximization (EM) algorithm, first formalized by Dempster, Laird and Rubin [6], for obtaining the maximum likelihood estimates of the parameters of a mixture model. This issue has also been discussed at length in a later book by McLachlan and Krishnan [20], which also gives a broad overview of the EM algorithm and its more efficient extensions.

It was not long before Bayesian statisticians got involved with the problem of analysis of finite mixtures. Research in this area gained momentum after Markov Chain Monte Carlo (MCMC) methods (see the Appendix) gained popularity in the nineteen-eighties. Among the significant efforts in this area are the approaches suggested by Phillips and Smith [23] based on the jump-diffusion methodology of Grenander and Miller [15], by Richardson and Green using the reversible jump methodology of Green [14], by Robert [28] on the basis of Bayes factors, and by Stephens [31] through the construction of appropriate continuous-time Markov birth-death processes. This chapter tries to provide an overview of these approaches with a view to making serious practitioners of pattern recognition aware of salient features of the more significant Bayesian techniques that are available for performing unsupervised classification.

4.2 Finite mixtures of probability distributions

Finite mixtures of probability distributions are typically used to model data where each (p-variate) observation $\boldsymbol{x}$ from a set $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ is assumed to have arisen from one of $k(\geq 2)$ groups, each group being modeled by a density from some parametric family

$$\mathcal{F} = \{f(\cdot|\boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\},$$

$\boldsymbol{\theta}$ being a q-variate parameter vector. That is, a finite mixture is represented as

$$g(\boldsymbol{x}|\boldsymbol{\phi}) = \sum_{i=1}^{k} \pi_i f(\boldsymbol{x}|\boldsymbol{\theta}_i), \tag{4.1}$$

where

$$\pi_i \geq 0,\ i = 1, 2, \ldots, k \quad \text{and} \quad \sum_{i=1}^{k} \pi_i = 1$$

and

$$\boldsymbol{\phi} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k, \boldsymbol{\pi}),$$

$\boldsymbol{\pi}$ being the k-variate vector $(\pi_1, \pi_2, \ldots, \pi_k)'$. The density of each group, $f(\boldsymbol{x}|\boldsymbol{\theta}_i)$, is referred to as a *component* of the mixture and $\pi_1, \pi_2, \ldots, \pi_k$ are called the *mixing* parameters.

4.2.1 Identifiability of finite mixtures

An important issue that has to be addressed in the context of estimation of the parameters $\boldsymbol{\phi}$ is their *identifiabilty.* In general, the parametric family $\mathcal{F}$ of probability density functions is said to be identifiable if distinct values of $\boldsymbol{\phi}$ determine distinct members of the family. This can be interpreted as follows in the case where $g(\boldsymbol{x}|\boldsymbol{\phi})$ defines a class of finite mixture densities according to (4.1). A class of finite mixtures is said to be identifiable for $\boldsymbol{\phi} \in \Phi$ if for any two members

$$g(\boldsymbol{x}|\boldsymbol{\phi}) = \sum_{i=1}^{k} \pi_i f(\boldsymbol{x}|\boldsymbol{\theta}_i) \tag{4.2}$$

and

$$g(\boldsymbol{x}|\boldsymbol{\phi}^*) = \sum_{i=1}^{k^*} \pi_i f(\boldsymbol{x}|\boldsymbol{\theta}_i{}^*), \tag{4.3}$$

then

$$g(\boldsymbol{x}|\boldsymbol{\phi}) \equiv g(\boldsymbol{x}|\boldsymbol{\phi}^*)$$

if and only if $k = k^*$ and we can permute the component labels so that

$$\pi_i = \pi_i^* \quad \text{and} \quad f(x|\theta_i) \equiv f(x|\theta_i^*),\ i = 1, 2, \ldots, k,$$

where $\equiv$ implies equality of the densities for almost all x relative to the underlying measure on $\mathbb{R}^p$ appropriate for $g(\cdot|\cdot)$. Titterington, Smith and Makov [35] provide a comprehensive account of the notion of identifiabilty of mixtures, with numerous examples and literature survey.

4.3 Bayesian approaches for mixture decomposition

As mentioned in Section 4.1, while classical statistical techniques like maximum likelihood estimation, did succeed in solving the problem of mixture decomposition, Bayesian statistics provided many more methodologies for attacking the same problem. This is largely due to the fact that in the last few years, *Markov Chain Monte Carlo (MCMC)* simulation techniques have made feasible the routine Bayesian analysis of many complex problems in higher dimensions. They have been applied successfully to the problem of estimation of parameters of mixture models. In the next few paragraphs, we shall be taking a closer look at some of more significant steps taken in this direction.

4.3.1 Use of jump-diffusion sampling

Phillips and Smith [23] have formulated the problem of decomposition of mixtures as a model choice problem. They have showed how the *jump-diffusion* methodology developed by Grenander and Miller [15] can be exploited to enable routine simulation-based analysis of general model choice problems, that is, problems involving the comparison of models of possibly different dimensions, when the essential difficulty is that of computing the high-dimensional integrals needed for calculating the normalization constants for posterior distributions under each model specification. Model uncertainty is accounted for by introducing a joint prior probability distribution over the set of possible models as well as the model parameters. Inference can be made by simulating realizations from the resulting posterior distribution using an iterative jump-diffusion sampling algorithm. The essential feature of this approach is that discrete transitions, or *jumps*, can be made between the models and, within a fixed model, the conditional posterior appropriate for that model is sampled. Model comparison or choice can then be based on the (simulated approximation to the marginal posterior distribution over the set of models. Details are provided in the Appendix.

4.3.1.1 *Mixture decomposition as a model choice problem*

Phillips and Smith [23] have formulated the problem of identifying the number of components in a univariate mixture distribution, together with the estimation of the parameters of the components densities, as a model choice problem (discussed in the following section), and suggest the application of jump-diffusion sampling proposed by Grenander and Miller [15] for solving it.

4.3.1.2 *The model choice problem*

Suppose we want to make inference on an unknown model which is assumed to come from a specified class of parametric models $\{\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots\}$ so that the overall parameter space can be written as a countable collection of subspaces,

$$\Theta = \bigcup_{k=0}^{\infty} \Theta_k$$

where Θ_k, a subspace of $R^{n(k)}$, denotes the $n(k)$-dimensional parameter space corresponding to the model $\mathcal{M}_k$.

Let $\boldsymbol{\theta} \in \Theta$, and $\boldsymbol{x}$ denote the model parameters and the observed data vector. Then the joint probability density π for both the model label and the model parameters can be written in the form

$$\pi(\boldsymbol{\theta}, k|\boldsymbol{x}) \;=\; \frac{\pi_k}{Z}\exp\{L_k(\boldsymbol{\theta}|\boldsymbol{x}) + P_k(\boldsymbol{\theta})\}\delta_k(\boldsymbol{\theta}),\;\; \boldsymbol{\theta} \in \Theta, k = 0, 1, \cdots$$

where $\delta_k(\boldsymbol{\theta}) = 1$ if $\boldsymbol{\theta} \in \Theta_k$ and zero otherwise. Here π_k is the prior probability for the model $\mathcal{M}_k$ so that $\sum_{k=0}^{\infty} \pi_k = 1$, and we assume that $\pi_k > 0$ for all k. The function $L_k(\boldsymbol{\theta}|\boldsymbol{x})$ is the conditional loglikelihood for $\boldsymbol{\theta}$ given $\boldsymbol{x}$ and the model $\mathcal{M}_k$ and $P_k(\boldsymbol{\theta})$ is the conditional log-prior for $\boldsymbol{\theta}$ given the model $\mathcal{M}_k$. For simplicity, we assume that both functions are identically zero on the subspace $\Theta - \Theta_k$,

$$i.e.,\ L_k(\boldsymbol{\theta}|\boldsymbol{x}) = P_k(\boldsymbol{\theta}) = 0 \text{ for } \boldsymbol{\theta} \notin \Theta_k.$$

The normalization constant Z is given by

$$Z = \sum_k \pi_k (\int \exp\{L_k(\boldsymbol{\theta}|\boldsymbol{x}) + P_k(\boldsymbol{\theta})\} d\boldsymbol{\theta}) = \sum \pi_k Z_k, \text{ say}$$

The marginal posterior densities have the following form

$$\begin{aligned}\pi(\boldsymbol{\theta}|\boldsymbol{x}) &= \sum \pi_k \exp\{L_k(\boldsymbol{\theta}|\boldsymbol{x}) + P_k(\boldsymbol{\theta})\}\delta_k(\boldsymbol{\theta})/Z \\ &= \sum \pi_k Z_k \pi_k(\boldsymbol{\theta}|\boldsymbol{x})/Z, \qquad \boldsymbol{\theta} \in \Theta\end{aligned}$$

where $\pi_k(\boldsymbol{\theta}|\boldsymbol{x})$ is the restriction of $\pi(\boldsymbol{\theta}|\boldsymbol{x})$ to Θ_k, and

$$\pi(k|\boldsymbol{x}) = \pi_k Z_k/Z, \quad k = 0, 1, \cdots$$

Hence, the density π has distribution μ given by

$$\mu(d\boldsymbol{\theta}) = \sum \pi_k Z_k \mu_k(d\boldsymbol{\theta})/Z = \sum p_k \mu_k(d\boldsymbol{\theta}), \text{ say },$$

where $\mu_k(\boldsymbol{\theta})$ is the distribution with density $\pi_k(\boldsymbol{\theta}|\boldsymbol{x})$.

4.3.1.3 *Jump-diffusion sampling*

Phillips and Smith [23] have shown how to construct an appropriate probability density over the joint sample space of the models and the parameters for the problem of decomposition of mixtures, where the model $\mathcal{M}_k$ corresponds to a mixture with k components, $k = 1, 2, \ldots$. As these densities are not amenable to analytic or numerical analysis in general, they propose the use of MCMC with target density $\pi(\boldsymbol{\theta}, k|\boldsymbol{x})$, denoted for the sake of brevity, by $\pi(\boldsymbol{\theta}, k)$. A Markov process $\{\boldsymbol{\theta}, k\}^{(t)}$ in continuous time t is constructed, which produces samples from the target density $\pi(\boldsymbol{\theta}, k)$. This stochastic process is generated from a jump component which makes discrete transitions between models at random times, and a diffusion component which samples values for the model-specific parameters between the jumps. Under appropriate conditions, the stationary distribution of $\{\boldsymbol{\theta}, k\}^{(t)}$ has density π, so that pseudo-samples from π may be produced by taking evenly-spaced states from a realization of the process, and ergodic averages may be used to estimate expectations, with respect to π, of integrable functions of $(\boldsymbol{\theta}, k)$. This particular MCMC methodology was proposed by Grenander and Miller [15], and is described briefly in the Appendix.

4.3.2 Bayesian estimation by Gibbs sampling

Robert [28] has proposed methodology for the estimation of parameters of finite mixture models based on Gibbs sampling. He has assumed, with little loss of generality, that the distributions $f(x|\boldsymbol{\theta})$ forming the functional basis, are all from the same exponential family

$$f(x|\boldsymbol{\theta}) \;=\; h(x)\exp[\boldsymbol{\theta}'x - \psi\boldsymbol{\theta}], \tag{4.4}$$

$\boldsymbol{\theta}$ being a real-valued vector. This family allows a conjugate prior for $\boldsymbol{\theta}$:

$$\pi(\boldsymbol{\theta}|\boldsymbol{y},\lambda) \;\propto\; \exp[\boldsymbol{\theta}'\boldsymbol{y} - \lambda\psi\boldsymbol{\theta}], \tag{4.5}$$

where $\boldsymbol{y}$ is a real-valued vector of constants of the same length as $\boldsymbol{\theta}$, and λ is a scalar constant. A conjugate prior for $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_k)'$ is the Dirichlet distribution $D(\alpha_1, \alpha_2, \ldots, \alpha_k)$ which has density

$$\pi^D(\pi_1, \pi_2, \ldots, \pi_k) \;\propto\; \pi_1^{\alpha_1-1}\pi_2^{\alpha_2-1}\ldots\pi_k^{\alpha_k-1},$$

where $\pi_1 + \pi_2 + \ldots + \pi_k = 1$. The Dirichlet distribution is a generalization of the beta distribution.

The posterior distribution of $\boldsymbol{\phi} = (\pi_1, \pi_2, \ldots, \pi_k, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k)$ can be written as

$$\begin{aligned}&[\pi_1, \pi_2, \ldots, \pi_k, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k | \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k] \\ &\propto\; \pi^D(\pi_1, \pi_2, \ldots, \pi_k)\prod_{i=1}^{k}\pi(\boldsymbol{\theta}_i|\boldsymbol{y}_i, \lambda_i)\prod_{j=1}^{n}\left(\sum_{i=1}^{k}\pi_i f(\boldsymbol{x}_j|\boldsymbol{\theta}_i)\right),\end{aligned} \tag{4.6}$$

where $[a|b]$ denotes the conditional distribution of a given b. Note that in (4.6), each component parameter $\boldsymbol{\theta}_i$ is assigned a distinct prior via unique constants y_i and λ_i in (4.5).

Equation (4.6) involves k^n terms of the form

$$\prod_{i=1}^{k}\pi^{\alpha_i+n_i-1}\pi(\boldsymbol{\theta}_i|\boldsymbol{y}_i + n_i\bar{\boldsymbol{x}}_i, \lambda_i + n_i),$$

where $n_1+n_2+\ldots+n_k = n$ and $\bar{\boldsymbol{x}}_i$ denotes the average of n_i of the n observations $\{\boldsymbol{x}_j\}$. While the posterior expectation of $(\pi_1, \pi_2, \ldots, \pi_k, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k)$ can be written in closed form, the computing time required to evaluate the $\boldsymbol{k_n}$ terms of (4.6) is far too large for the direct Bayesian approach to be implemented, even for problems that are only moderately large. However, Bayesian estimation of mixtures can be performed straightforwardly using Gibbs sampling. The methodology is described for a general exponential family setting, in the next few paragraphs.

Consider a sample $x_1, x_2, \ldots, x_n$ from the mixture probability density $g(x)$, given by (4.1), assuming $f(x|\theta_i)$ is from the exponential family, as in (4.4). It is possible to rewrite $x \sim g(x)$ under the hierarchical structure

$$x \sim f(x|\theta_z),$$

where z is an integer identifying the component generating observation x, taking value i with prior probability π_i, $1 \le i \le k$. The components of (4.1) may not be meaningful; it is a feature of many Gibbs implementations that artificial random variables are introduced to create conditional distributions that are convenient for sampling. The vector $z = (z_1, z_2, \ldots, z_n)'$ is the *missing data* part of the sample where, for $i = 1, 2, \ldots, n$, $z_i = j$ if the ith observation is from the jth class, $j = 1, 2, \ldots, k$. Had the missing data been available, it would have been much easier to estimate the mixture model. From (4.1),

$$[x_1, x_2, \ldots, x_n | z_1, z_2, \ldots, z_n] = \prod_{j:z_j=1} f(x_j|\theta_1) \ldots \prod_{j:z_j=k} f(x_j|\theta_k) \quad (4.7)$$

which implies that

$$\begin{aligned} &[\pi_1, \pi_2, \ldots, \pi_k, \theta_1, \theta_2, \ldots, \theta_k | x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_n] \\ &\propto \ \pi_1^{\alpha_1 = n_1 - 1} \pi_2^{\alpha_2 + n_2 - 1} \ldots \pi_k^{\alpha_k + n_k - 1} \\ &\quad \times \pi(\theta_1 | y_1 + n_1 \bar{x}_1, \lambda_1 + n_1) \ldots \pi(\theta_1 | y_k + n_k \bar{x}_k, \lambda_k + n_k), \end{aligned} \quad (4.8)$$

where now

$$n_i = \sum_j I(z_j = i) \quad \text{and} \quad n_i \bar{x}_i = \sum_{j:z_j=i} x_j,$$

and $I(\cdot)$ is the indicator function taking the value 1 when its argument is true, and 0 otherwise. This conditional decomposition implies that the conjugate structure is preserved for a given allocation/missing data structure $z_1, z_2, \ldots, z_n$ and thus, conditional on $z_1, z_2, \ldots, z_n$, the simulation is indeed possible

In the implementation of Gibbs sampling described below, samples for the missing data z_j and the parameters $\pi, \theta_1, \theta_2, \ldots, \theta_k$ are alternately generated, producing a missing-data chain and a parameter chain, both of which are Markov. The finite-state structure of the missing-data chain allows many convergence results to be easily established for it, and transferred automatically to the parameter chain.

4.3.2.1 *Gibbs sampling implementation*

Once a missing data structure is imposed as in (4.7) above, the direct implementation of Gibbs sampling is to run successive simulations from (4.8) for the parameters of the model, and from $[z_1, z_2, \ldots, z_n | \boldsymbol{\phi}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$ for the missing data. The simulation is straightforward if conjugate prior distributions are used for the parameters:

Algorithm 1

Step 1. Simulate

$$\boldsymbol{\theta}_i \sim \pi(\boldsymbol{\theta}_i | \boldsymbol{y}_i + n_i \bar{\boldsymbol{x}}_i, \lambda_i + n_i), \ (i = 1, 2, \ldots, k)$$

and

$$(\pi_1, \pi_2, \ldots, \pi_k) \sim D(\alpha_1 + n_1, \alpha_2 + n_2, \ldots, \alpha_k + n_k).$$

Step 2. Simulate, for $j = 1, 2, \ldots, n$,

$$[z_j | \boldsymbol{x}_j, \pi_1, \pi_2, \ldots, \pi_k, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k] = \sum_{i=1}^{k} p_{ij} I(z_j = i),$$

with

$$p_{ij} = \frac{\pi_i f(\boldsymbol{x}_j | \boldsymbol{\theta}_i)}{\sum_{\ell=1}^{k} \pi_\ell f(\boldsymbol{x}_j | \boldsymbol{\theta}_\ell)}, \ (i = 1, 2, \ldots, k).$$

Step 3. Update n_i and $\bar{x}_i$, $i = 1, 2, \ldots, k$.

This algorithm is a special case of the data augmentation algorithm of Tanner and Wong [32]. Robert [28] discusses the application of this approach to a mixture of normal densities, as well as other implementation-related issues.

4.3.3 Application of reversible jump Markov Chain Monte Carlo methods

Richardson and Green [25] proposed a fully Bayesian analysis of mixture models on the basis of the reversible jump Markov Chain Monte Carlo methods presented by Green in an earlier work [14]. These methods are capable of jumping between the parameter subspaces corresponding to different numbers of components in the mixture. A sample from the full joint distribution is thereby generated, and this can be used for a thorough presentation of many aspects of the posterior distribution. The number of components and the mixture component parameters are modeled jointly, and inference about them is based upon their posterior probabilities.

A hierarchical model is assumed within a Bayesian framework, the unknown parameters k, ϕ are regarded as drawn from appropriate prior distributions, and the joint distribution of all variables is written in general as

$$\begin{aligned} &p(k, \boldsymbol{\pi}, \boldsymbol{z}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k, \boldsymbol{X}) \\ &= p(k)p(\boldsymbol{\pi}|k)p(\boldsymbol{z}|\boldsymbol{\pi}, k)p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k|\boldsymbol{z}, \boldsymbol{\pi}, k)p(\boldsymbol{X}|\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k, \boldsymbol{z}, \boldsymbol{\pi}, k), \end{aligned} \tag{4.9}$$

where $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)$, $\boldsymbol{z} = (z_1, z_2, \dots, z_n)'$, z_i being the group (or class) label for the ith observation, treated as a latent allocation variable, with

$$P[z_i = j] = \pi_j, \ j = 1, 2, \dots, k,$$

and

$$\boldsymbol{x}_i|\boldsymbol{z} \sim f(\cdot|\boldsymbol{\theta}_{z_i})$$

independently for $i = 1, 2, \dots, k$. In (4.9), it is natural to impose further conditional independencies, so that

$$p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k|\boldsymbol{z}, \boldsymbol{\pi}, k) = p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k|k)$$

and

$$p(\boldsymbol{X}|\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k, \boldsymbol{z}, \boldsymbol{\pi}, k) = p(\boldsymbol{X}|\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k, \boldsymbol{z}).$$

These simplify the joint distribution (4.9) to

$$\begin{aligned} &p(k, \boldsymbol{\pi}, \boldsymbol{z}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k, \boldsymbol{x}) \\ &= p(k)p(\boldsymbol{\pi}|k)p(\boldsymbol{z}|\boldsymbol{\pi}, k)p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k|k)p(\boldsymbol{x}|\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k, \boldsymbol{z}), \end{aligned}$$

which is the Bayesian hierarchical model. For complete flexibility, an extra layer is added to the hierarchy, and the priors for k, π and $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k$ are allowed to depend on hyperparameters λ, δ and η respectively, drawn from independent hyperpriors. The joint distribution of all variables is then expressed as

$$\begin{aligned} & p(\lambda, \delta, \eta, k, \boldsymbol{\pi}, \boldsymbol{z}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k, \boldsymbol{X}) \\ = \; & p(\lambda)p(\delta)p(\eta)p(k|\lambda)p(\boldsymbol{\pi}|k, \delta)p(\boldsymbol{z}|\boldsymbol{\pi}, k)p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k|k, \eta) \qquad (4.10) \\ & \times p(\boldsymbol{X}|\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k, \boldsymbol{z}). \end{aligned}$$

The reversible jump MCMC methodology of Green [14] (see Appendix) is readily applicable to this model. Richardson and Green have presented the corresponding approach for the case of a mixture of normal distributions, and discussed many implementation-related issues like sensitivity to prior distributions of parameters, and so on.

4.3.4 Application of Markov birth-death processes

Stephens [31] has proposed an MCMC method for the Bayesian analysis of a finite mixture with an unknown number of components, as an alternative to the reversible jump approach of Richardson and Green [25]. His approach treats the parameters of the model as a *marked point process* and extends methods suggested by Ripley [26] to create a Markov birth-death process with an appropriate stationary distribution. Each point of the marked point process represents a component of the mixture. The MCMC scheme allows the number of components to vary by permitting new components to be "born", and existing components to "die". These births and deaths occur in continuous time, and the relative rates at which they occur determine the stationary distribution of the process. Stephens formalizes the relationship between these rates and the stationary distribution, and uses this to construct an easily simulated process, in which births occur at a constant rate from the prior, and deaths occur at a rate which is very low for components that are critical in explaining the data, and very high for components which do not help explain the data. The accept-reject mechanism of reversible jump is thus replaced by a mechanism which allows both "good" and "bad" births to occur, but reverses bad births very quickly through a very quick death. The method has been illustrated by fitting mixtures of normal (and t) distributions to univariate and bivariate data.

4.3.4.1 *Model*

The following finite mixture model is assumed, which is a special case of (4.1):

$$g(\boldsymbol{x}|\boldsymbol{\phi},\eta) = \sum_{i=1}^{k} \pi_i f(\boldsymbol{x}|\boldsymbol{\theta}_i,\eta), \tag{4.11}$$

where k is possibly unknown but finite, η is a (possibly vector) common parameter which is common to all components, other quantities having the same significance as in earlier sections. Here too, a missing data formulation of the model is made, and each observation is assumed to arise from a specific but unknown component z_j of the mixture. The model (4.11) can be written in terms of the *missing data* $z_1, z_2, \ldots, z_n$, assumed to be realizations of independent and identically distributed discrete random variables $Z_1, Z_2, \ldots, Z_n$ with probability mass function

$$\Pr(Z_j = i|\boldsymbol{\phi},\eta) = \pi_i, \ (j = 1, 2, \ldots, n; \, i = 1, 2, \ldots, k).$$

Conditional on the Z_j's, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ are assumed to be independent observations from the densities

$$p(\boldsymbol{x}_j|Z_j = i, \boldsymbol{\phi}, \eta) = f(\boldsymbol{x}_j|\boldsymbol{\theta}_i, \eta).$$

A hierarchical model is assumed for the prior on the parameters $(k, \boldsymbol{\phi}, \eta)$ with $(\pi_1, \boldsymbol{\theta}_1), (\pi_2, \boldsymbol{\theta}_2), \ldots, (\pi_k, \boldsymbol{\theta}_k)$ being exchangeable. Specifically, it is assumed that the prior distribution for $(k, \boldsymbol{\phi})$ given hyperparameters ω, and common component parameters η, has density $r(k, \boldsymbol{\phi}|\omega, \eta)$ with respect to an underlying symmetric measure defined in [31]. To ensure exchangeability, it is required that for any k, $r(\cdot)$ is invariant under relabeling of the components, in that

$$\begin{aligned} &r(k, (\pi_1, \pi_2, \ldots, \pi_k), (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k)) \\ &= r(k, (\pi_{e_1}, \pi_{e_2}, \ldots, \pi_{e_k}), (\boldsymbol{\theta}_{e_1}, \boldsymbol{\theta}_{e_2}, \ldots, \boldsymbol{\theta}_{e_k})) \end{aligned} \tag{4.12}$$

for all permutations $e_1, e_2, \ldots, e_k$ of $1, 2, \ldots, k$.

4.3.4.2 *Constructing Markov chains by simulating point processes*

As mentioned earlier, each component of the mixture is viewed as a point in parameter space, and the theory of simulation from point processes is adapted to construct a Markov chain that has the posterior distribution of the parameters as its stationary distribution. Since, for given k, the prior distribution of

$\boldsymbol{\phi}$ does not depend on the labeling of the components, and the likelihood

$$L(k,\boldsymbol{\phi},\eta) = \prod_{i=1}^{n} [\pi_1 f(\boldsymbol{x}_i|\boldsymbol{\theta}_1,\eta) + \pi_1 f(\boldsymbol{x}_i|\boldsymbol{\theta}_2,\eta) + \ldots + \pi_1 f(\boldsymbol{x}_i|\boldsymbol{\theta}_k,\eta)]$$

is also invariant under permutations of the component labels, the posterior distribution

$$p(k,\boldsymbol{\phi}|\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_n,\omega,\eta) \propto L(k,\boldsymbol{\phi},\eta)r(k,\boldsymbol{\phi})$$

will be similarly invariant. Fixing ω and η we can thus ignore the labeling of the components and can consider any set of k parameter values $\{(\pi_1,\boldsymbol{\theta_1}),(\pi_k,\boldsymbol{\theta_k}),\ldots,(\pi_k,\boldsymbol{\theta_k})\}$ as a set of k points in $[0,1]\times\Theta$, with the constraint that $\pi_1 + \pi_2 + \ldots + \pi_k = 1$. The posterior distribution can thus be seen as a *point process* on $[0,1]\times\Theta$. Equivalently, it can be looked upon as a *marked point process* in Θ, with each point $\boldsymbol{\theta}_i$ having an associated *mark* $\pi_i \in [0,1]$, the marks being constrained to sum to unity. This permits the use of realizations of a continuous-time Markov birth-death process with stationary distribution $p(k,\boldsymbol{\phi}|\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_n,\omega,\eta)$ as approximately random samples from the latter, keeping ω and η fixed. The procedure for this is described in the following paragraphs. Stephens has also described how to do this when ω and η are allowed to vary.

Let Ω_k denote the parameter space for the mixture model with k components, ignoring the labeling of the components, and let $\Omega = \cup_{k\geq 1}\Omega_k$. $y = \{(\pi_1,\boldsymbol{\theta_1}),(\pi_2,\boldsymbol{\theta_2}),\ldots,(\pi_k,\boldsymbol{\theta_k})\} \in \Omega_k$ will be used to represent the parameters of the mixture model represented by (4.11) keeping η fixed, so that we may write $(\pi_i,\boldsymbol{\theta}_i) \in y$ for $i = 1,2,\ldots,k$. Given ω and η, the invariance of $L(\cdot)$ and $r(\cdot)$ under permutation of the component labels, permits the definition of $L(y)$ and $r(y)$ in an obvious manner. Births and deaths on Ω are defined as follows:

Births. If at time t, the process is at $y = \{(\pi_1,\boldsymbol{\theta_1}),(\pi_2,\boldsymbol{\theta_2}),\ldots,(\pi_k,\boldsymbol{\theta_k})\} \in \Omega_k$, and a birth is said to occur at $(\pi,\boldsymbol{\theta}) \in [0,1]\times\Theta$, then the process jumps to

$$\begin{aligned} &y \cup (\pi,\boldsymbol{\theta}) \\ &:= \{(\pi_1(1-\pi),\boldsymbol{\theta_1}),(\pi_2(1-\pi),\boldsymbol{\theta_2}),\ldots,(\pi_k(1-\pi),\boldsymbol{\theta_k}),(\pi,\boldsymbol{\theta})\} \in \Omega_{k+1}. \end{aligned}$$

Deaths. If at time t the process is at $y = \{(\pi_1,\boldsymbol{\theta_1}),(\pi_2,\boldsymbol{\theta_2}),\ldots,(\pi_k,\boldsymbol{\theta_k})\} \in$

Ω_k and a death is said to occur at $(\pi_i, \boldsymbol{\theta}_i) \in y$, then the process jumps to

$$\begin{aligned} & y \backslash (\pi_i, \boldsymbol{\theta}_i) \\ := & \left\{ \left(\tfrac{\pi_1}{1-\pi_i}, \boldsymbol{\theta}_1\right), \ldots, \left(\tfrac{\pi_{i-1}}{1-\pi_i}, \boldsymbol{\theta}_{i-1}\right), \left(\tfrac{\pi_{i+1}}{1-\pi_i}, \boldsymbol{\theta}_{i+1}\right), \ldots, \left(\tfrac{\pi_k}{1-\pi_i}, \boldsymbol{\theta}_k\right) \right\} \in \Omega_{k-1}. \end{aligned}$$

Thus a birth increases the number of components by one, while a death decreases the number of components by one. The definitions above ensure that births and deaths are inverse operations to each other, and the constraint $\pi_1 + \pi_2 + \ldots + \pi_k = 1$ remains satisfied after a birth or death. With births and deaths so defined, let us consider the following continuous-time Markov birth-death process:

When the process is at $y \in \Omega_k$, let births and deaths occur as independent Poisson processes as follows:

Births: Births occur at overall rate $\beta(y)$, and when a birth occurs, it does so at a point $(\pi, \boldsymbol{\theta}) \in [0,1] \times \Theta$, chosen according to density $b(y; (\pi, \boldsymbol{\theta}))$ with respect to the product measure $\mathcal{U}^1 \times \nu$, where $\mathcal{U}^1$ is the uniform (Lebesgue) measure on $[0,1]$.

Deaths: When the process is at $y = \{(\pi_1, \boldsymbol{\theta}_1), (\pi_2, \boldsymbol{\theta}_2), \ldots, (\pi_k, \boldsymbol{\theta}_k)\}$, each point $(\pi_j, \boldsymbol{\theta}_j)$ dies independently of the others as a Poisson process with rate

$$\delta_j(y) = d(y \backslash (\pi_j, \boldsymbol{\theta}_j); (\pi, \boldsymbol{\theta}))$$

for some $d : \Omega \times ([0,1] \times \Theta) \to \mathbb{R}^+$. The overall death rate is given by $\delta(y) = \sum_j \delta_j(y)$.

The time to the next birth/death event is then exponentially distributed with mean $1/(\beta(y)+\delta(y))$, and it will be a birth with probability $\beta(y)/(\beta(y)+\delta(y))$, and a death of component j with probability $\delta_j(y)/(\beta(y)+\delta(y))$. Conditions are imposed on $b(\cdot)$ and $d(\cdot)$ to ensure that the birth-death process does not jump to regions with "zero" density. Stephens also gives algorithms that implement this theory for the case where ω and η are kept fixed, as well as for the case where they are allowed to vary.

4.3.5 Testing for number of components using Bayes factors

Raftery [24] has proposed ways of utilizing MCMC output (simulations) from the posterior of the mixture model (4.1) for the testing of hypotheses about the number of components, using Bayes factors. He discusses this as a special case of the more general problem of model selection. The Bayes factor comparing two competing models is the ratio of marginal likelihoods under the two models, the marginal likelihood of a model being the probability of the data with all the model parameters integrated out. He proposes two estimators, the Laplace-Metropolis estimator and the Candidate's estimator, for this purpose. He has also provided a review of methods for estimating the marginal likelihood of a model from posterior simulation output, like the harmonic mean estimators and other importance-sampling and related estimators.

4.3.5.1 *The Bayes factor*

The standard Bayesian solution to the hypothesis testing problem is to represent both the null and alternative hypotheses as parametric probability models, and to compute the Bayes factor for one against the other. The Bayes factor B_{10} for a model $\mathcal{M}_1$ against another model $\mathcal{M}_0$, given data $\boldsymbol{x}$, is the ratio of posterior to prior odds, namely,

$$B_{10} = \frac{P(\boldsymbol{x}|\mathcal{M}_1)}{P(\boldsymbol{x}|\mathcal{M}_0)}, \tag{4.13}$$

the ratio of the marginal likelihoods. In (4.13),

$$P(\boldsymbol{x}|\mathcal{M}_i) = \int P(\boldsymbol{x}|\boldsymbol{\theta}_i, \mathcal{M}_i)P(\boldsymbol{\theta}_i|\mathcal{M}_k)d\boldsymbol{\theta}_k, \tag{4.14}$$

where $\boldsymbol{\theta}_k$ is the vector of parameters of model $\mathcal{M}_i$, and $P(\boldsymbol{\theta}_k|\mathcal{M}_i)$ is the prior density, $i = 0, 1$.

The model selection problem (of which the problem of determination of the number of components in a mixture is a special case) arises when one initially considers several models, and wishes to select one of them that is most probable in light of the data. A Bayesian solution to this problem is to choose the model with the highest posterior probability. The marginal likelihoods yield posterior probabilities of all the models, as follows. If k models, $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$, are being considered, then by the Bayes theorem, the posterior probability of $\mathcal{M}_i$ is

$$P(\mathcal{M}_i|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|\mathcal{M}_i)P(\mathcal{M}_i)}{\sum_{j=1}^{k} P(\boldsymbol{x}|\mathcal{M}_j)P(\mathcal{M}_j)}. \tag{4.15}$$

4.3.5.2 *Marginal likelihood estimation by importance sampling*

The marginal likelihood $P(x|\mathcal{M}_i)$ being the key quantity needed for Bayesian model selection and related problems, Raftery [24] has outlined several estimators based on importance sampling and related concepts, that are constructed with a sample $\{\theta^{(t)}, t = 1, \dots, T\}$ from the posterior distribution of the parameter θ of the model. Such a sample can be generated by direct analytic simulation, MCMC, and so on.

Let $L(\theta)$ be the likelihood and $\pi(\theta)$ be the prior. Then the marginal likelihood is

$$P(x) = \int L(\theta)\pi(\theta)d\theta.$$

Let

$$||\chi||_h = \frac{1}{T}\sum_{t=1}^{T}\chi(\theta^{(t)}),$$

where $\chi(\cdot)$ is a function and $\{\theta^{(t)}\}$ is a sample of size T from the probability distribution density $h(\theta)/\int h(\phi)d\phi$, h being a positive function.

Importance sampling can be used to evaluate $P(x)$ as follows. suppose that we can sample from a density proportional to the positive function $g(\theta)$, say, the density $cg(\theta)$, where $c^{-1} = \int g(\theta)d\theta$. Then

$$\begin{aligned} P(x) &= \int L(\theta)\pi(\theta)d\theta \\ &= \int L(\theta)\left[\frac{\pi(\theta)}{cg(\theta)}\right]cg(\theta)d\theta. \end{aligned} \tag{4.16}$$

Given a sample $\{\theta^{(t)}, t = 1, \dots, T\}$ from the density $cg(\theta)$, then, as suggested by (4.17), a simulation-consistent estimator of $P(x)$ is

$$\begin{aligned} \hat{P}(x) &= \frac{1}{T}\sum_{t=1}^{T}\frac{L(\theta^{(t)})\pi(\theta^{(t)})}{cg(\theta^{(t)})} \\ &= \left|\left|\frac{L\pi}{cg}\right|\right|_g. \end{aligned}$$

If c cannot be found analytically, it remains only to estimate it from the MCMC output. A simulation-consistent estimator of c is $\hat{c} = ||\pi/g||_g$. This yields the general importance-sampling estimator of $P(x)$ with importance-sampling function $g(\cdot)$, namely,

$$\hat{P}_{IS} = \frac{\left|\left|\frac{L\pi}{g}\right|\right|_g}{\left|\left|\frac{\pi}{g}\right|\right|_g}. \tag{4.17}$$

Here g is a positive function. If it is normalized to be a probability density, then (4.17) becomes $\hat{P}_{IS} = ||L\pi/g||_g$. Raftery [24] considers several special cases.

4.3.5.3 *Marginal likelihood estimation by maximum likelihood*

(1) *The Laplace-Metropolis estimator* The Laplace method for approximating integrals is based on a Taylor series expansion of the real-valued function $f(u)$ of the d-dimensional vector $\boldsymbol{u}$, and yields the approximation

$$\int E^{f(\boldsymbol{u})} d\boldsymbol{u} \approx (2\pi)^{d/2} |\boldsymbol{A}|^{\frac{1}{2}} \exp\{f(\boldsymbol{u}^*)\},$$

where $\boldsymbol{u}^*$ is the value of $\boldsymbol{u}$ at which f attains its maximum, and $\boldsymbol{A}$ is the negative of the inverse Hessian (second-derivative matrix) of f evaluated at $\boldsymbol{u}^*$. When applied to (4.17), it yields

$$p(\boldsymbol{x}) \approx (2\pi)^{d/2} |\boldsymbol{\Psi}|^{\frac{1}{2}} P(\boldsymbol{x}|\tilde{\boldsymbol{\theta}}) P(\tilde{\boldsymbol{\theta}}), \tag{4.18}$$

where $\tilde{\boldsymbol{\theta}}$ is the posterior mode of $h(\boldsymbol{\theta}) = \log\{P(\boldsymbol{x}|\boldsymbol{\theta})P(\boldsymbol{\theta})\}$, $\boldsymbol{\Psi}$ is the negative of the inverse Hessian of $h(\boldsymbol{\theta})$ evaluated at $\boldsymbol{\theta} = \tilde{\boldsymbol{\theta}}$, and d is the dimension of $\boldsymbol{\theta}$. Thus the Laplace method requires $\tilde{\boldsymbol{\theta}}$ and $\boldsymbol{\Psi}$. The simplest method of estimating $\tilde{\boldsymbol{\theta}}$ is from posterior simulation output. The matrix $\boldsymbol{\Psi}$ is the asymptotic posterior covariance matrix, so it seems natural to approximate it by the estimated posterior covariance matrix from the posterior simulation output.

(2) *Candidate's estimator* From the Bayes theorem it follows that

$$P(\boldsymbol{x}) = P(\boldsymbol{x}|\boldsymbol{\theta}) \left(\frac{P(\boldsymbol{\theta})}{P(\boldsymbol{\theta}|\boldsymbol{x})} \right), \tag{4.19}$$

which is also related to the "Candidate's formula" of Besag [3]. Hence the name of the estimator based on it. Typically, $P(\boldsymbol{x}|\boldsymbol{\theta})$ and $P(\boldsymbol{\theta})$ can be calculated directly. If $P(\boldsymbol{\theta}|\boldsymbol{x})$ is also available in closed form, then (4.19) permits the computation of $P(\boldsymbol{x})$ analytically without integration. However, this is not usually the case, but (4.19) can still be used if one has a sample from the posterior $P(\boldsymbol{\theta}|\boldsymbol{x})$, in which case one can simply estimate it from the posterior sample using nonparametric density estimation techniques. An important consideration is the choice of $\boldsymbol{\theta}$. Though (4.19) holds for all values of $\boldsymbol{\theta}$, the most precise estimate of $P(\boldsymbol{\theta}|\boldsymbol{x})$ would usually result from using the posterior mode or a value very close to it.

(3) *The data-augmentation estimator* MCMC methods often involve the introduction of latent data $\boldsymbol{z}$ such that, when $\boldsymbol{z}$ is known, then the "complete data likelihood" $P(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ has a simple form. Rosenkranz [29] has shown that for calculating marginal likelihoods, it can be decidedly advantageous to integrate over the latent data directly, particularly if there are conditional independence properties that may be exploited to reduce the dimensionality of the integrals involved. Raftery [24] has discussed a few strategies for doing this.

4.4 Discussion

Problems of unsupervised pattern classification are most often solved by means the myriad clustering techniques that are available, and generally, for every such problem, it is possible to find a few that are quite successful at doing what they are expected to do. However, the PR community has also long been aware of the fact that within a probabilistic framework, it is possible to formulate the same problem as that of estimation of the parameters of a finite identifiable mixture of probability densities. Maximum likelihood solutions to many specialized forms of this problem have been around for quite some time. However, in the last couple of decades, Bayesian statistics has been witnessing vigorous research activity in general, and in the area of decomposition of mixtures, in particular. This is mainly because the Bayesian community discovered that computational problems, thought to be intractable till then, could easily be solved by simulation-based approaches collectively referred to as Markov chain Monte Carlo methods. The problem of estimation of parameters of mixtures of probability densities, could also be attacked in various ways, using this general idea. In this chapter, a survey of some of the significant efforts in this direction has been provided, with a view to demonstrating their utility in the context of unsupervised classification.

References

[1] Y. Amit, U. Grenander, and M. I. Miller, "Ergodic properties of jump-diffusion processes," Tech. Rep. 361, Department of Statistics, University of Chicago, 1993.

[2] M. R. Anderberg, *Cluster Analysis for Applications.* New York: Academic Press, 1973.

[3] J. E. Besag, "A candidate's formula: a curious result in Bayesian prediction," *Biometrika*, vol. 76, p. 183, 1989.

[4] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Functions.* New York: Plenum Press, 1981.

[5] J. C. Bezdek, J. Keller, R. Krishnapuram, and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing.* Boston: Kluwer Academic, 1999.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm (with discussion)," *Journal of the Royal Statistical Society, Series B*, vol. 39, pp. 1–38, 1977.

[7] L. Devroye, *Non-Uniform Random Variate Generation.* New York: Springer, 1986.

[8] R. O. Duda, D. G. Stork, and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: John Wiley & Sons, second ed., 2000.

[9] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[10] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.

[11] A. E. Gelfand and A. F. M. Smith, "Sampling-based approaches to calculating marginal densities," *Journal of the American Statistical Association*, vol. 85, pp. 398–409, 1990.

[12] A. Gelman and D. B. Rubin, "Inference from iterative simulation using multiple sequences," *Statistical Science*, vol. 7, pp. 457–472, 1992.

[13] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, 1984.

[14] P. J. Green, "Reversible jump Markov chain Monte Carlo computation and bayesian model determination," *Biometrika*, vol. 82, pp. 711–732, 1995.

[15] U. Grenander and M. I. Miller, "Representations of knowledge in complex systems," *Journal of the Royal Statistical Society, Series B*, vol. 56,

pp. 549–603, 1994.

[16] J. A. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.

[17] W. K. Hastings, "Monte Carlo sampling methods using Markov chains, and their applications," *Biometrika*, vol. 57, pp. 97–109, 1970.

[18] A. K. Jain and R. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[19] G. J. McLachlan and K. E. Basford, *Mixture Models: Inference and Applications to Clustering*. New York: Marcel Dekker, 1988.

[20] G. J. McLachlan and T. Krishnan, *The EM Algorithm and its Extensions*. New York: Wiley, 1997.

[21] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[22] K. Pearson, "Contribution to the mathematical theory of evolution," *Philosophical Transactions of the Royal Society, Series A*, vol. 185, pp. 71–110, 1894.

[23] D. B. Phillips and A. Smith, "Bayesian model comparison via jump diffusions," in *Markov Chain Monte Carlo in Practice* (W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, eds.), Boca Raton, FL: Chapman and Hall/CRC, 1996.

[24] A. E. Raftery, "Hypothesis testing and model selection," in *Markov Chain Monte Carlo in Practice* (W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, eds.), Boca Raton, FL: Chapman and Hall/CRC, 1996.

[25] S. Richardson and P. J. Green, "On Bayesian analysis of mixtures with an unknown number of components (with discussion)," *Journal of the Royal Statistical Society, Series B*, vol. 59, pp. 731–792, 1997.

[26] B. D. Ripley, *Stochastic Simulation*. New York: Wiley, 1987.

[27] G. O. Roberts and A. F. M. Smith, "Simple conditions for the convergence of the Gibbs sampler and the Metropolis/Hastings algorithm," *Stochastic Processes and their Applications*, vol. 49, pp. 207–216, 1994.

[28] C. P. Robert, "Mixtures of distributions: inference and estimation," in *Markov Chain Monte Carlo in Practice* (W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, eds.), Boca Raton, FL: Chapman and Hall/CRC, 1996.

[29] G. Rosenkranz, "The Bayes factor for model evaluation in a hierarchical Poisson model for area counts", Ph.D. thesis, Department of Biostatistics, University of Washington, 1992.

[30] P. H. A. Sneath and R. Sokal, *Numerical Taxonomy*. San Fransisco: Freeman, 1973.

[31] M. Stephens, "Bayesian analysis of mixture models with an unknown number of components–an alternative to reversible jump methods," *Annals of Statistics*, vol. 28, no. 1, pp. 40–74, 2000.

[32] M. Tanner and W. Wong, "The calculation of posterior distributions by data augmentation (with discussion)," *Journal of the American Statistical Association*, vol. 82, pp. 528–550, 1987.

[33] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. San Diego: Academic Press, 1999.

[34] L. Tierney, "Markov chains for exploring posterior distributions," Tech. Rep., School of Statistics, University of Minnesota, Minneapolis, MN, 1991.

[35] A. F. M. Smith, D. M. Titterington and U. E. Makov, *Statistical Analysis of Finite Mixture Distributions*. New York: Wiley, 1985.

[36] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. New York: Addison–Wesley, 1974.

[37] T. Y. Young and T. W. Calvert, *Classification Estimation and Pattern Recognition*. New York: Elsevier, 1974.

Appendix

Markov Chain Monte Carlo (MCMC) methods

The general idea behind Markov Chain Monte Carlo (MCMC) methods is as follows:
Let $\pi(x)$ be a probability density. If $x_1, x_2, x_3, \ldots, x_n$ are randomly drawn realizations from π, then

$$\int f(x)\pi(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

However, if direct sampling from π is not possible, then one can still produce samples which are random in the limit, using the following idea. Form a Markov chain X_t with equivalent state space to x, which has π as its unique equilibrium distribution. If this chain is run for a sufficiently long time, realizations from the sample path of the chain can be used as approximately random samples from π, enabling Monte Carlo methods to be applied.

The Gibbs sampler [13], the Metropolis-Hastings algorithm [17] and the jump-diffusion algorithm [15, 23] are instances of MCMC methods and are described in the following sections.

Two fundamental theorems that are most relevant to MCMC theory, and respectively state conditions for the existence of a unique equilibrium distribution, and the convergence of empirical averages to their expectations, are stated below.

Theorem 4.1 *If X_t has invariant distribution π, and is irreducible and aperiodic, then π is the unique equilibrium distribution, that is,*

$$\lim_{t\to\infty} ||P^t(x,\cdot) - \pi(\cdot)|| = 0, \ \forall x,$$

where $||\cdot||$ denotes the total variational distance.

Theorem 4.2 *If X_t has invariant distribution π, and is irreducible and aperiodic, and $P(x,\cdot)$ is absolutely continuous with respect to π, then for any real-valued absolutely-integrable function f*

$$\frac{1}{n}\sum_{i=1}^{n} f(X_i) \to \int f(\theta)\pi(\theta)d\theta \quad \text{almost surely},$$

for all values of X_0.

The relevance of the above two asymptotic results is clear: the first theorem may be exploited to produce an (approximate) independent sample from π (the successive X_t being correlated, however). The second theorem ensures that the ergodic average of a function of interest over realizations from a particular chain provides a strongly consistent estimator of its expectation.

A Markov chain is essentially defined then by its transition kernel P, and in the following sections we consider some particular forms of P for which the above theory can be shown to hold, and which lead to conceptually appealing algorithms for sampling.

The Gibbs sampler

The Gibbs sampler was originally developed for discrete problems in image analysis by Geman and Geman [13]. Gelfand and Smith [11] extended the algorithm to a continuous setting, and demonstrated how it could be used to solve general Bayesian inference problems. In fact, the Gibbs sampler is a generalized Metropolis algorithm where the probability of accepting the candidate state is always unity [12].

Let $\boldsymbol{X}_t = (X_{t,1}, X_{t,2}, \ldots, X_{t,n})$. Then $\boldsymbol{X}_{t+1}$ is obtained from $\boldsymbol{X}_t$ by successively drawing random variates from the full conditional distributions of π in the following way:

- generate $X_{t+1,1}$ from $\pi(X_1|X_{t,j}, j > 1)$.
- generate $X_{t+1,i}$ from $\pi(X_i|X_{t+1,j}, j < i, X_{t,j}, j > i)$.
- generate $X_{t+1,n}$ from $\pi(X_n|X_{t+1,j}, j < n)$.

This defines a Markov chain with transition kernel P given by

$$P(x,y) = \prod_{i=1}^{n} \pi(y_i|y_j, j < i, x_j, j > i).$$

Provided each of the conditional distributions is well-defined so that P is well-defined, it is not difficult to see that π is a stationary distribution for P. For any measurable set A,

$$\begin{aligned}\int P(x,A)\pi(x)dx &= \int \left(\int_A p(x,y)dy\right)\pi(x)dx \\ &= \int_A \left(\int p(x,y)\pi(x)dx\right)dy \\ &= \int_A \left(\int \prod_{i=1}^{n} \pi(y_i|y_j, j<i, x_j, j>i)\pi(x)dx\right)dy \\ &= \int_A \prod_{i=1}^{n} \left(\int \pi(y_i|y_j, j<i, x_j, j>i)\pi(x)dx\right)dy \\ &= \int_A \prod_{i=1}^{n} \pi(y_i|y_j, j<i)dy \\ &= \int_A \pi(y)dy \\ &= \pi(A),\end{aligned}$$

making use of the following equality:

$$\begin{aligned}&\int \pi(y_i|y_j, j<i, x_j, j>i)\pi(x)dx \\ &= \int \pi(y_i|y_j, j<i, x_j, j>i)\pi(x_j, j>i)dx_{i+1}\dots dx_n \\ &= \int \pi(y_i, x_j, j>i|y_j, j<i)dx_{i+1}\dots dx_n \\ &= \pi(y_i|y_j, j<i).\end{aligned}$$

Roberts and Smith [27] provide a sufficient condition for P to be well-defined, namely, that π be lower semi-continuous at 0. This condition is satisfied if, for each x such that $\pi(x) > 0$ there exists an open neighborhood $N_x \subset \Theta$, with $x \in N_x$, and an $\epsilon > 0$, such that

$$\pi(y) \geq \epsilon > 0 \quad \forall y \in N_x.$$

Further sufficient conditions ensuring irreducibility and aperiodicity of P are that $\int \pi(x)dx_i$ is locally bounded for each i, and that Θ is connected. For most applications, it is straightforward to check whether these conditions are satisfied, and provided they hold, it follows that the Markov chain X_t has unique equilibrium distribution π.

The implementation of the Gibbs sampler relies crucially on the facility to generate realizations from each of the full conditional distributions of π. In more complex problems, sophisticated techniques for random-number generation will be required [7, 26].

The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm [21, 17] forms the Markov chain $\{X_t, t = 1, 2, \ldots\}$ in the following way. Let $q(x,y)$ be an arbitrary proposal density, or probability transition function, so that

$$\int q(x,y)dy \ = \ 1 \ \ \forall x.$$

The sampled state for X_{t+1} is obtained from the realization $X_t = x$ by drawing a candidate state y at random with probability density given by $q(x,y)$. The proposed new state is accepted with probability $\alpha(x,y)$, and otherwise rejected in favor of the previous state, that is, x, where the acceptance probability α is defined by

$$\alpha(x,y) \ = \ \begin{cases} \min\left(\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right) \text{if} & \pi(x)q(x,y) > 0, \\ 1 \qquad\qquad\qquad\quad \text{if} & \pi(x)q(x,y) = 0. \end{cases}$$

This produces a Markov chain with transition probability kernel P given by

$$P(x,dy) \ = \ p(x,y)dy \ + \ r(x)\delta_x(dy),$$

where

$$p(x,y) \ = \ \begin{cases} \alpha(x,y)q(x,y) \text{ if} & x \neq y, \\ 0 & \text{otherwise.} \end{cases}$$

$$r(x) \ = \ 1 - \int p(x,y)dy,$$

and δ_x denotes point mass at x.

Note that if the proposal density is symmetric in its arguments, that is, $q(x,y) = q(y,x) \ \ \forall x$ then the acceptance probability only involves the ratio of the target densities at the new and old states. This corresponds to the original version of the Metropolis algorithm [21]. Note also that the density

π is needed only upto proportionality. It is easy to see that π satisfies the reversibility condition

$$\pi(x)p(x,y) \;=\; \min(\pi(y)q(y,x), \pi(x)q(x,y)) \;=\; \pi(y)p(y,x),$$

and it therefore follows that π is a stationary distribution for P [34]. For any measurable set A,

$$\begin{aligned}
\int P(x,A)\pi(x)dx &= \int \left(\int_A p(x,y)dy\right)\pi(x)dx \;+\; \int r(x)\delta_x(A)\pi(x)dx \\
&= \int_A \left(\int p(x,y)\pi(x)dx\right)dy \;+\; \int r(x)\pi(x)dx \\
&= \int_A \left(\int p(y,x)\pi(y)dx\right)dy \;+\; \int r(x)\pi(x)dx \\
&= \int_A (1-r(y))\pi(y)dy \;+\; \int_A r(x)\pi(x)dx \\
&= \int_A \pi(y)dy \\
&= \pi(A).
\end{aligned}$$

Using the results of Roberts and Smith [27], it is straightforward to check the irreducibility and aperiodicity of P. This is because if q is irreducible and $q(x,y) = 0 \equiv q(y,x) = 0$, then P is irreducible, and if q is aperiodic then so is P. In practice, it is not difficult to find suitable choices of q satisfying these two conditions, and subject to this it follows that the Markov chain X_t has unique equilibrium distribution, as required. An excellent discussion of the merits of prospective proposal densities can be found in the [34]. Some possible choices of q are described below.

Random-walk chains: Let f be a density on Θ, and suppose that q is defined by

$$q(x,y) \;=\; f(x-y),$$

so that the candidate state y is obtained by incrementing the current state x with a random variate generated from f. If f is spherically symmetric about the origin, then q is symmetric since

$$q(x,y) = f(x-y) = f(|x-y|) = f(|y-x|) = f(y-x) = q(y,x),$$

so that this is a version of the Metropolis algorithm. If f is positive on Θ, or if f is positive on a neighborhood of the origin and Θ is open and connected, then q is irreducible and aperiodic. Possible choices of f: a normal distribution with mean zero, a t-distribution with mean zero, *etc.*

Independence chains: When q is chosen to produce candidate states not depending on the current state, that is, $q(x,y) = q(y)$, then the chain generated is called an independence chain. If q is such that it assigns positive probability to each measurable set A for which $\pi(A) > 0$, then q will be irreducible and aperiodic as well, so that both theorems apply.

Jump-diffusion sampling

The jump component

Suppose that at time t the process is in state $(\boldsymbol{\theta}, k)$. The probability of jumping out of model $\mathcal{M}_k$ is specified through a jump intensity $q((\boldsymbol{\theta}, k), (\boldsymbol{\psi}, h))$, where $\boldsymbol{\psi} \in \Theta_h$ and $h \neq k$.Informally, $q((\boldsymbol{\theta}, k), (\boldsymbol{\psi}, h))dt$ can be thought of as the probability density of jumping from $(\boldsymbol{\theta}, k)$ to $(\boldsymbol{\psi}, h)$ in an infinitesimal interval of time dt. Let $\mathcal{T}(\boldsymbol{\theta}, k)$ denote the set of all $(\boldsymbol{\psi}, h)$ which can be reached from $(\boldsymbol{\theta}, k)$ in a single jump. This is called the jump set of $(\boldsymbol{\theta}, k)$. Often it is convenient to choose $\mathcal{T}(\boldsymbol{\theta}, k)$ such that

$$\mathcal{T}(\boldsymbol{\theta}, k) \subset \Theta_{k-1} \cup \Theta_{k+1},$$

that is, from model $\mathcal{M}_k$ one can jump to model $\mathcal{M}_{k-1}$ or $\mathcal{M}_{k+1}$ only. Regularity conditions given by Grenander and Miller [15] require that $\mathcal{T}(\boldsymbol{\theta}, k)$ must be chosen so that jumps are always local, that is, for some suitably defined metric D and constant B,

$$D((\boldsymbol{\theta}, k), (\boldsymbol{\psi}, h)) \; = \; D((\boldsymbol{\psi}, h), (\boldsymbol{\theta}, k)) < B,$$

for all states $(\boldsymbol{\psi}, h)$ in $\mathcal{T}(\boldsymbol{\theta}, k)$. Also the jump sets are required to be reversible, that is,

$$\text{if } (\boldsymbol{\psi}, h) \in \mathcal{T}(\boldsymbol{\theta}, k) \text{ then } (\boldsymbol{\theta}, k) \in \mathcal{T}(\boldsymbol{\psi}, h)).$$

To decide when to jump, the marginal jump intensity, namely,

$$r(\boldsymbol{\theta}, k) \; = \; \sum_h \int (\boldsymbol{\psi}, h) \in \mathcal{T}(\boldsymbol{\theta}, k) q((\boldsymbol{\theta}, k), (\boldsymbol{\psi}, h)) d\boldsymbol{\phi}$$

must be calculated. The marginal probability of jumping out of model $\mathcal{M}_k$ in time interval $(t, t + dt)$ is then $r(\boldsymbol{\theta}, k)dt$, which generally does not depend on

$(\boldsymbol{\theta}, k)$. Jump times $t_1, t_2, \ldots$ can be sampled by generating unit exponential random variables $e_1, e_2, \ldots$, and setting t_i to be such that

$$\int_{t_i-1}^{t_i} r((\boldsymbol{\theta}, k)^{(t)})dt \geq e_i, \tag{4.20}$$

where $t_0 = 0$. In practice, the algorithm is implemented in discrete time-steps of size Δ.

If a jump must be made at time t (that is, if $t = t_i$ for some i), then the new state (ψ, h) must be selected according to the transition kernel

$$Q((\boldsymbol{\theta}, k), (\psi, h)) = \frac{q((\boldsymbol{\theta}, k), (\psi, h))}{r(\boldsymbol{\theta}, k)}, \ (\psi, h) \in \mathcal{T}(\boldsymbol{\theta}, k).$$

Informally, $Q((\boldsymbol{\theta}, k), (\psi, h))$ is the conditional probability density of jumping from $(\boldsymbol{\theta}, k)$ to $\psi \in \Theta_h$, given that a jump out of model $\mathcal{T}_k$ must be made immediately. Jump intensity $q((\boldsymbol{\theta}, k), (\psi, h))$ must be chosen so that a balance condition is satisfied, to ensure that the stationary distribution of the process in $\pi(\boldsymbol{\theta}, k)$ [15]. Grenander and Miller [15] and Amit *et al.* [1] have proposed two strategies that ensure this; these are briefly described below:
Gibbs jump dynamics: Under this choice of dynamics, the process jumps from one subspace to another with transition intensity given by

$$q((\boldsymbol{\theta}, k), (\psi, h)) = C\pi_h \exp\{L_h(\psi) + P_h(\psi)\},$$

for $(\psi, h) \in \mathcal{T}(\boldsymbol{\theta}, k)$, where constant C can be chosen arbitrarily, though its choice affects the frequency of jumps.
Metropolis jump dynamics: Under this choice of dynamics, the process jumps from one subspace to another with transition intensity $q((\boldsymbol{\theta}, k), (\psi, h))$ given by

$$q((\boldsymbol{\theta}, k), (\psi, h)) = \min(1, \exp[L_h(\psi) - L_k(\boldsymbol{\theta})])\pi_h \exp[P_h(\psi)],$$

for $(\psi, h) \in \mathcal{T}(\boldsymbol{\theta}, k)$. This process may be simulated as follows. Jump times are calculated from (4.20) using a modified jump intensity

$$q((\boldsymbol{\theta}, k), (\psi, h)) = C\pi_h \exp[P_h(\psi)].$$

At each jump time, a candidate state (ψ, h) is generated from the prior restricted to $\mathcal{T}(\boldsymbol{\theta}, k)$, that is, with density proportional to $\pi_h \exp[P_h(\psi)]$, and accepted with probability

$$\min(1, \exp[L_h(\psi) - L_k(\boldsymbol{\theta})]),$$

else reject this point and set (ψ, h) equal to the current point $(\boldsymbol{\theta}, k)$. This is just the Metropolis-Hastings algorithm.

Moving between jumps: the diffusion component

At times between jumps, that is, for $t_{i-1} < t < t_i$, $i = 1, 2, \ldots$, the process follows a Langevin diffusion in the subspace to which the process jumped at the previous jump time. Within the fixed subspace Θ_k, this is given by

$$d\boldsymbol{\theta}^{(t)} = \frac{dt}{2}\left[\frac{d}{d\boldsymbol{\theta}}(L_k(\boldsymbol{\theta}) + P_k(\boldsymbol{\theta}))\right]_{\boldsymbol{\theta}^{(t)}} + dW_{n(k)}^{(t)}, \boldsymbol{\theta} \in \Theta_k,$$

where $W_{n(k)}^{(t)}$ is standard Brownian motion in $n(k)$ dimensions. This can be approximated, using a discrete time-step $\triangle$, by

$$\boldsymbol{\theta}^{(t+\triangle)} = \boldsymbol{\theta}^{(t)} + \frac{\triangle}{2}\frac{d}{d\boldsymbol{\theta}}\left[(L_k(\boldsymbol{\theta}^{(t)}) + P_k(\boldsymbol{\theta}^{(t)}))\right] + \sqrt{\triangle}\boldsymbol{z}_{n(k)}^{(t)},$$

where $\boldsymbol{z}_{n(k)}^{(t)}$ is vector of $n(k)$ independent standard normal variates.

Reversible jump Markov chain Monte Carlo methods

The Markov chain Monte Carlo methods discussed so far have been restricted to problems of Bayesian computation where the joint distribution of all variables has a density with respect to some fixed standard underlying measure. Therefore they cannot be applied to problems like Bayesian model determination, in which the dimensionality of the parameter vector is typically not fixed. Green [14] proposed a new framework for the construction of reversible Markov chain samplers that jump between parameter subspaces of differing dimensionality, which is flexible and entirely constructive.

Let there be a countable collection of candidate models $\{\mathcal{M}_k, k \in \mathcal{K}\}$, where model $\mathcal{M}_k$ has a vector $\boldsymbol{\theta}^{(k)}$ of unknown parameters, assumed to lie in $\mathbb{R}^{n_k}$, and the dimension n_k may vary from model to model. We observe data $\boldsymbol{x}$. A natural hierarchical structure is expressed by modeling the joint distribution of $(k, \boldsymbol{\theta}^{(k)}, \boldsymbol{x})$ as

$$p(k, \boldsymbol{\theta}^{(k)}, \boldsymbol{x}) = p(k)p(\boldsymbol{\theta}^{(k)}|k)p(\boldsymbol{x}|k, \boldsymbol{\theta}^{(k)}),$$

that is, the product of model probability, prior and likelihood. For convenience, the pair $(k, \boldsymbol{\theta}^{(k)})$ is abbreviated as $\boldsymbol{y}$. For given k, $\boldsymbol{y}$ lies in $\mathcal{C}_k = \mathcal{K} \times \mathbb{R}^{n_k}$. Generally, $\boldsymbol{y}$ varies over $\mathcal{C} = \cup_{k\in\mathcal{K}}\mathcal{C}_k$.

In a typical application with multiple parameter subspaces $\{C_k\}$ of different dimensionality, it becomes necessary to devise different types of move between the subspaces. These are combined to form a hybrid sampler, by random choice between available moves at each transition, in order to traverse freely across the combined parameter space C. Attention is restricted to Markov chains in which detailed balance* is attained within each move type.

When the current state is y a move of type m is proposed that takes the state to dy' with probability $q_m(y, dy')$. This is a sub-probability measure on m and y', that is, $\sum_m q_m(x, C) \leq \infty$ and, with probability $1 - \sum_m q_m(x, C)$, no change to the present state is proposed. Not all moves m are available from each starting states y, so that for each y, $q_m(x, C) = 0$ for some m. The proposed move is not automatically accepted. The probability of acceptance is $\alpha_m(y, y')$. It has been shown that, for detailed balance, $\alpha_m(y, y')$ must satisfy

$$\alpha_m(y, y') \;=\; \min\left\{1, \frac{\pi(dy')q_m(y', dy)}{\pi(dy)q_m(y, dy')}\right\},$$

where π is the density of y, provided the dimension-matching assumption holds, namely, that $\pi(dy)q_m(y, dy')$ has a finite density $f_m(y, y')$ with respect to a symmetric measure ζ_m on $C \times C$.

*In MCMC simulation from a target density π, a Markov transition kernel $P(x, dx')$ is said to satisfy detailed balance if

$$\int_A \int_B \pi dx P(x, dx') = \int_B \int_A \pi dx' P(x', dx)$$

for all appropriate A, B.

Chapter 5
SHAPE IN IMAGES

K. V. Mardia

Department of Statistics
University of Leeds
Leeds LS2 9JT, UK
e-mail: *k.v.mardia@leeds.ac.uk*

Abstract

The Bayes theorem is a vehicle for incorporating prior knowledge in updating the degree of belief in light of data. For example, the state of tomorrow's weather can be predicted using belief or likelihood of tomorrow's weather given today's weather data. We give a brief review of the recent advances in the area with emphasis on high level Bayesian image analysis where shape is an important element of prior modeling. It has been gradually recognized that knowledge based algorithms based on Bayesian Analysis are more widely applicable and reliable than *ad hoc* algorithms. Advantages include the use of explicit and realistic statistical models making it easier to understand the working behind such algorithms and allowing confidence statements to be made about conclusions. These systems are not necessarily as time consuming as might be expected. However, more care is required in using the knowledge effectively for a given specific problem; this is very much an art rather than a science.

5.1 High-level Bayesian image analysis

Since the early 1980s, statistical approaches to image analysis using the Bayesian paradigm Bayesian image analysis have proved to be very successful. Initially, the methodology was primarily developed for low-level image analysis (see, for example, [32]) but is increasingly used for high-level tasks.

To use the Bayesian framework one requires a prior model which represents our initial knowledge about the objects in a particular scene, and a likelihood or noise model for an image which is the joint probability distribution of the grey levels in the image, dependent on the objects in the scene. By using the Bayes Theorem one derives the posterior distribution of the objects in the scene, which can be used for inference. Examples of the tasks of interest include segmentation of the scene into *object* and *background*, and object recognition. The computational work involved is generally intense because of the large amount of data in each image.

An appropriate method for high-level Bayesian image analysis is the use of deformable templates, pioneered by Grenander and his colleagues [20, 21]. Our description follows the common theme of [35, 37, 42]. We assume that we are dealing with applications where we have prior knowledge on the composition of the scene and we can formulate parsimonious geometric descriptions for objects in the images. For example, in medical imaging we can expect to know *a priori* the main subject of the image, *e.g.*, a heart or a brain. Consider our prior knowledge about the objects under study to be represented by a parameterized ideal prototype or template S_0. Note that S_0 could be a template of a single object or many objects in a scene. A probability distribution is assigned to the parameters with density (or probability function) $\pi(S)$, which models the allowed variations S of S_0. Hence, S is a random vector representing all possible templates with associated density $\pi(S)$. Here S is a function of a finite number of parameters, say $\theta_1, \ldots, \theta_p$.

In addition to the prior model we require an image model. Let the observed image I be the matrix of grey levels x_i, where $i = (i_1, i_2) \in \{1, \ldots, r\} \times \{1, \ldots, c\}$ are the $r \times c$ pixel locations. The image model or likelihood is the joint probability density function (p.d.f.) (or probability function for discrete data) of the grey levels given the parameterized objects S, written as $L(I|S)$. The likelihood expresses the dependence of the observed image on the deformed template. It is often convenient to generate an intermediate synthetic image but we will not need it here (see [37]).

By the Bayes theorem, the posterior density $\pi(S|I)$ of the deformed template S given the observed image I is

$$\pi(S|I) \propto L(I|S)\pi(S). \tag{5.1}$$

An estimate of the true scene can be obtained from the posterior mode (the maximum *a posteriori* or MAP estimate) or the posterior mean. The posterior mode is found either by a global search, gradient descent (which is often impracticable due to the large number of parameters) or by techniques such as simulated annealing [15]) or iterated conditional modes (ICM) [6]. Alternatively, Markov chain Monte Carlo (MCMC) algorithms (see, for example, [7, 16]) provide techniques for simulating from a density. In Section 5.3 we will give a specific example of the MCMC technique. The latter technique has the advantage that it allows a study of the whole posterior density itself, and so credibility or confidence regions can be easily obtained. For more information on MCMC methods, please refer to the appendix of Chapter 3.6.

5.2 Prior models for objects

The key to the successful inclusion of prior knowledge in high-level Bayesian image analysis is through specification of the prior distribution. Many approaches have been proposed, including methods based on outlines, landmarks and geometric parameters. The prior can be specified either through a model with known parameters or with parameters estimated from training data.

5.2.1 Geometric parameter approach

One approach is to consider a geometric template for S consisting of parametric components, *e.g.*, line segments, circles, ellipses, arcs. Examples include a circle of random radius to represent the central disc of galaxies [50]; simple geometric shapes for facial features such as eyes and mouths [48, 49]; circular templates to represent pellets in an image, where the number of pellets is unknown [5].

In these models, distributions are specified for the geometrical parameters, and the hierarchical approach of Phillips and Smith [48, 49] provides a simple method. Often templates are defined by both global and local parameters. The global parameters represent the object on a coarse scale and the local parameters give a more detailed description on a fine scale. The idea of a hierarchical model for templates is to specify the marginal distribution of the global parameters and a conditional distribution for the local parameters given the global values. This hierarchical division of parameters can be extended to give a complete description of the local dependence structure between variables. Hence, conditionally, each parameter depends only on variables in the levels immediately above and below it.

In general, we assume that templates can be summarized by a small number of parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$ say, where variation in $\boldsymbol{\theta}$ will produce a wide range of deformations of the template. By explicitly assigning a prior distribution to $\boldsymbol{\theta}$, we can quantify the relative probability of different deformations. The prior can be based on training data which need not be a large data-set. By simulation, we can check the possible shapes that could arise.

For example, consider the mouth template of Phillips and Smith [48, 49]. They use marginal normal distributions for (x, y) (location), θ (rotation), b (half the width) and conditional normal distributions for $a \mid b$ (height given halfwidth), $c \mid a$ (depth given height) and $d \mid a$ (curvature of parabola given height). Here x, y, θ and b are global, a is intermediate, and c and d are local.

In more complicated image scenes where several templates are required, the organization of the templates can be considered at a higher level of hierarchy. For example, there may be nesting relationships between the templates, which are subject to constraints. For example, with human face templates there are global constraints such as the requirement that the eyes, mouth and nose must be strictly contained within the head boundary but this is deterministic not stochastic. We now discuss a specific example relating to a mushroom but it could be the iris of an eye in medical context.

5.2.2 Mushroom template model and its prior densities

For simplicity, we regard a mushroom as a circle [39]. A simple two dimensional template for a circle requires center (θ_1, θ_2) and log-radius θ_3. This is also a small component of the eye template. Here we have three parameters.

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3). \tag{5.2}$$

Next we discuss the prior distribution for $\boldsymbol{\theta}$. For an image $\mathbf{F}$ of size $N \times N$, say, it is simplest to take (θ_1, θ_2) to be uniformly distributed over the square $0 < \theta_1 < N$, $0 < \theta_2 < N$ so that the density of (θ_1, θ_2) is simply $1/N^2$. Suppose the radius r has prior mean μ with variance σ^2. Since $r > 0$, it is preferable to model $\theta_3 = \log r$ by a normal distribution $N(\log \mu, \sigma^2/\mu^2)$ since approximately

$$\mathsf{var}(\theta_3) = (d \log r/dr)^2_{r=\mu} \mathsf{var}(r) = \sigma^2/\mu^2. \tag{5.3}$$

That is, θ_3 has a lognormal distribution. Then the joint probability density function of $\boldsymbol{\theta}$ is

$$\pi^*(\boldsymbol{\theta}) = C \exp\{-\frac{\mu^2}{2\sigma^2}(\theta_3 - \log\mu)^2\}, 0 < \theta_1, \theta_2 < N, \theta_3 > 0. \tag{5.4}$$

Note that the model can be viewed as hierarchical in the sense that we can write it as

$$P(\theta_1, \theta_2) P(\theta_3 | \theta_1, \theta_2) \tag{5.5}$$

so that the "global" parameters (the location θ_1, θ_2) are followed by the "local" parameter (the log-radius θ_3). This hierarchy is not very relevant here since θ_3 is independent of θ_1 and θ_2, but in general it is helpful to describe location, scale and orientation as global parameters and other parameters as local deformations.

5.2.3 Landmarks: shape distributions and point distribution models

Consider a single object in $\mathbb{R}^2$ with k landmarks. One approach to specifying the template is to work with the landmarks directly. Denote the coordinates of the landmarks in $\mathbb{R}^2$ as $\boldsymbol{X}$, a $2k \times 1$ vector. The configuration can be parameterized as an overall location (x_c, y_c) (*e.g.*, centroid), a scale β (*e.g.*, length between the baseline landmarks 1 and 2), a rotation Γ (*e.g.*, the rotation of the baseline) and some suitable shape coordinates, such as Procrustes tangent coordinates, shape PC scores or Bookstein coordinates U^B. A shape distribution could be chosen as a prior model together with more vague priors for location, scale and rotation.

A principal component (PC)-based prior model for object recognition was suggested by Cootes *et al.* [10]. The model has proved very successful in a variety of medical and engineering applications. As well as a prior for shape one also needs to introduce vague priors for location, rotation and scale, and an independence model is often reasonable.

A measure for shape difference could also be used in a prior density such as the full Procrustes distance $d_F(S, S_0)$, where S is the observed template and

S_0 is the ideal template. Suitable prior models could then be suggested based on such distances or costs (energies).

The Gibbs distribution has a probability density function of the form

$$\pi(S) = \frac{1}{Z}\exp(-\frac{1}{2}E_{int}(S, S_0)) \tag{5.6}$$

and has been used widely in image analysis. Simple expressions for $E_{int}(S, S_0)$ will be of particular interest. The term $E_{int}(S, S_0)$ is called the internal energy function and Z is the integrating constant or partition function. For example, we could take $E_{int}(S, S_0) = 2\kappa \sin^2 \rho(S, S_0)$ and so

$$\pi(S) \propto \exp(-\kappa d_F^2(S, S_0)), \tag{5.7}$$

which is the shape distribution corresponding to the complex Watson distribution [11].

5.2.4 Graphical templates

Amit and Kong [3, 4] and Amit [1] use a graphical template method which is based on comparisons between triangles in an image and a template. Cost functions are given for matching triangles in the deformed template to triangles in the observed template and the total cost gives a measure of discrepancy. The cost functions involve hard constraints limiting the range in which the observed angles can deviate from the template angles and soft constraints penalizing the deviations from template angles.

The procedure had been implemented into a fast algorithm which finds an optimal match in an image in polynomial time, provided the template graph is decomposable (see [1, 2]).

5.2.5 Thin-plate splines

The thin-plate spline can be used in a prior model. The pair of thin-plate splines transformation from the ideal template S_0 to a deformed version S has an energy function associated with it, namely, the total minimum bending energy $J(\Phi)$. Thus if $E_{int}(S, S_0) = J(\Phi)$, then a prior distribution of the deformation could be obtained using the Gibbs distribution. This would inhibit landmark changes that require a large bending energy. If S is an affine transformation of S_0, then the total bending energy is zero. This prior was suggested by Mardia *et al.* [42], and further applications were given by Mardia and Hainsworth [40].

5.2.6 Outlines

5.2.6.1 *Edges*

In modeling the outline of an object, Grenander and co-workers specify a series of points around the outline connected by straight-line segments. In their method, variability in the template is introduced by pre-multiplying the line segment by random scale-rotation matrices. Let us assume that the outline is described by the parameters $\theta^{\mathrm{T}} = \{\theta_1^{\mathrm{T}}, \theta_2^{\mathrm{T}}\}$, where θ_1 denotes a vector of similarity parameters (*i.e.*, location, size and rotation) whereas θ_2 denotes say k landmarks of the outlines. For θ_1, we can construct a prior with center-of-gravity parameters, a scale parameter and a rotation by an angle α. Conditional on θ_1, we can construct a prior distribution of θ_2 (see Chapter 11 of [11].

Another general approach is given by Miller *et al.* [47]. The model has been applied successfully to images of mitochondria. This model with circulant symmetry has also been studied by Kent *et al.* [30], who consider additional constraints to make $\{\theta_2\}$ invariant under the similarity transformations. They also explore the eigenstructure of the circulant covariance matrix [30].

5.2.6.2 *The snake*

The snake [29] in its simplest form is a spline with a penalty on the mean curvature. Snakes are used for fitting a curve to an image when there is no underlying template, with the aim that the resulting estimated curve is smooth. Let the outline be $f(t) \in \mathbb{C}, t \in [0,1], f(0) = f(1)$. The penalty in the snake can be written as

$$-2\log P(t) \propto \int_0^1 \alpha(t)|f''(t)|^2 \mathrm{d}t + \int_0^1 \beta(t)|f'(t)|^2 \mathrm{d}t, \tag{5.8}$$

where $\alpha(t)$ and $\beta(t)$ denote the degree of stiffness and elasticity at t, respectively. For $t_j = j/k, j = 0, 1, \ldots, k$, denote $f(t_j) = u_j + iv_j$. We find that the right-hand side of (5.8) can be written approximately for large k as [39]

$$\left\{\sum \alpha_j(u_{j+1} + u_{j-1} - 2u_j)^2 + \sum \beta_j(u_{j+1} - u_j)^2\right\} \tag{5.9}$$

$$+\left\{\sum \alpha_j(v_{j+1} + v_{j-1} - 2v_j)^2 + \sum \beta_j(v_{j+1} - v_j)^2\right\}. \tag{5.10}$$

Thus $\{u_j\}$ and $\{v_j\}$ also form a separable Markov Random Field of order 2.

5.2.6.3 *Semi-landmarks*

Bookstein [8, 9] has introduced the shape analysis of outlines based on semi-landmarks, which are allowed to slip around an outline. This approach appears promising for the shape analysis of curved outlines.

Alternatively, one may be interested in shape analysis of the outlines and, for example, in differences in average shape between groups. Bookstein [9] has investigated the shape of the corpus callosum in the midsagittal sections of MR scans of the brain in control subjects and schizophrenic patients. An important practical aspect of the work is how to sample the points – one needs to achieve a balance between coarse sampling of landmarks where information might be lost and over-fine sampling of landmarks where power can be lost.

5.2.6.4 *Polar prior for outlines*

A prior for star-shaped objects is given by Mardia and Qian [44]. An example of a star-shaped object suitable for such analysis is the outline of a leaf. Let an object be represented by a set of vertices

$$\{g^0(1), \ldots, g^0(k)\} = c^0.$$

Its template S_0 can represent a number of different types of objects, say q. A translated, scaled, rotated version of c_0 with shift μ, scale ρ and rotation θ, is given by $\{sg^0(1), \ldots, sg^0(k)\}$ and can be described in polar coordinates by

$$sg^0(j) = \mu + \mu_0 + \rho R^0(j)(\cos\{\theta(j)+\theta\}, \sin\{\theta(j)+\theta\})^{\mathrm{T}}, \; j = 1, \ldots, k, \quad (5.11)$$

where $\mu_0 + R^0(j)(\cos\theta(j), \sin\theta(j))^{\mathrm{T}}$ is the expression for $g^0(j)$. Therefore we can write the prior probability for the orientation or pose of the object as

$$\pi(s|c^0) = \pi(\mu, \rho, \theta|\mu_0, R^0), \quad (5.12)$$

where $R^0 = \{R^0(1), \ldots, R^0(k)\}^{\mathrm{T}}$. Here the parameters consist of μ, ρ and θ. A deformation can be incorporated (see [37]).

5.3 Inference

Inference about a scene in an image is made through the posterior distribution S. The full range of Bayesian statistical inference tools can be used and, as stated in Section 5.1, the maximum of the posterior density (the MAP estimate) is frequently used, as well as the posterior mean. There may be alternative occasions when particular template parameters are of great interest, in which case one would maximize the appropriate marginal posterior densities [49].

Example 5.1 Consider the circular mushroom template. As in [35], denote the circle template by $S(\theta)$ with parameters $\theta = (\theta_1, \theta_2, \theta_3)$ which segments the image into two regions: inside S and outside S. Suppose the image is subject to observational noise. A simple model is

$$x_i = \nu_0 + \epsilon_i \text{ if } i \in S; \; x_i = \nu_1 + \epsilon_i \text{ if } i \notin S, \tag{5.13}$$

where $x_i \in \mathbb{R}$ denotes the grey level at the ith pixel in an $N \times N$ grid, and we suppose that the ϵ_i are independent normally distributed. The parameters (ν_0, τ^2) and (ν_1, τ^2) summarize the mean and variance of the intensities inside the circle and in the background, respectively. Hence the likelihood of the image is

$$L(x|\theta) \propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i\in S}(x_i-\nu_0)^2 - \frac{1}{2\tau^2}\sum_{i\notin S}(x_i-\nu_1)^2\right\}. \tag{5.14}$$

More realistic models might include autocorrelation between the errors or an allowance for blurring, or both.

By the Bayes Theorem the posterior density of S given the data x is

$$\pi(\theta|x) \;\; \propto \;\; L(x|\theta)\pi(\theta). \tag{5.15}$$

Hence we get

$$\pi(\theta|x) \propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i\in S}(x_i-\nu_0)^2 - \frac{1}{2\tau^2}\sum_{i\notin S}(x_i-\nu_1)^2 - \frac{\mu^2}{2\sigma^2}(\theta_3 - \log\mu)^2\right\} \tag{5.16}$$

with support $0 < \theta_1, \theta_2 < N, \quad -\infty < \theta_3 < \infty$.

□

One possible estimate of $\theta = (\theta_1, \theta_2, \theta_3)$ is given by the posterior mean. One way to calculate the posterior mean is by a simulation method which does not depend on the complicated normalizing constant in $\pi(\theta|x)$. For discrete values of θ_1, θ_2 and θ_3, a grid search is an alternative approach. A popular technique is a Markov chain Monte Carlo (MCMC) procedure using the Metropolis-Hastings algorithm [25, 46]. This procedure generates a Markov chain whose equilibrium distribution is the posterior distribution of $\theta|x$. For simplicity, we write $\pi(\theta|x)$ as $\pi(\theta)$ for this discussion, and choose an arbitrary initial estimate of θ. Then at each iteration, generate θ_{new}, a new set of values from

$$N(\theta_{\text{old}}, \text{diag}(\sigma_1^2, \sigma_2^2, \sigma_3^2)), \tag{5.17}$$

say, with density

$$g(\theta_{\text{new}}|\theta_{\text{old}}) \propto \exp\left\{ -\frac{1}{2}\sum_{j=1}^{3}(\theta_{\text{new},j} - \theta_{\text{old},j})^2/\sigma_j^2 \right\}, \tag{5.18}$$

where θ_{old} denotes the value of θ at the previous iteration. This distribution is called the *proposal* distribution and its parameters σ_1^2, σ_2^2 and σ_3^2 are changed iteratively to approximately match the variance of the posterior distribution. Calculate the *Hastings ratio*

$$p = \pi(\theta_{\text{new}}|x)g(\theta_{\text{old}}|\theta_{\text{new}})/\{\pi(\theta_{\text{old}})g(\theta_{\text{new}}|\theta_{\text{old}})\}, \tag{5.19}$$

where now $\pi(\theta_{\text{new}}|x)$ is the posterior density. Here the proposal density is symmetric, $g(\theta_{\text{old}}|\theta_{\text{new}}) = g(\theta_{\text{new}}|\theta_{\text{old}})$, so that $p = \pi(\theta_{\text{new}}|x)/\pi(\theta_{\text{old}}|x)$.

We accept $\theta = \theta_{\text{new}}$ with probability $\min(1, p)$ otherwise we keep $\theta = \theta_{\text{old}}$, *i.e.*, if $p \geq 1$, we take $\theta = \theta_{\text{new}}$, whereas if $p < 1$, we perform a further randomization by drawing a random sample from uniform (0,1), and accept $\theta = \theta_{\text{new}}$ with probability p. Typically, a burn-in period is allowed in initial simulations and the average is taken of a subset of the remaining simulations.

For the algorithm to work in practice, we need suitable choices for σ_j so that the proposal density will roughly approximate the posterior distribution. Also we need to judge when convergence has taken place (see [7, 16, 19].

Above we have updated all components of θ at once. Alternatively, it is possible to update the components of θ one at a time using individual proposal densities $g_j(\theta_{\text{new},j}|\theta_{\text{old},j})$, *i.e.*, $\theta_{\text{new},j} \sim N(\theta_{\text{old},j}, \sigma_j^2)$. Hence we change only one component of θ at a time in turn to complete a sweep. For this example,

$$p_j = \pi(\theta_{\text{new},j}|A_j, x)g(\theta_{\text{old},j}|A_j)/\{\pi(\theta_{\text{old},j}|A_j, x)g(\theta_{\text{new},j}|A_j)\}, \tag{5.20}$$

where $A_j = \{\theta_{\text{new},k}, k < j; \theta_{\text{old},k}, k > j\}, j = 1, 2, 3$. Then we select $\theta_j = \theta_{\text{new},j}$ with probability $\min(1, p_j)$.

5.4 Multiple objects and occlusions

Most of the work described above has been in terms of one template. In this section, S_0 will denote a collection of different types of objects with S as their deformed observed version of S_0. Working with a fixed known number of objects is a straightforward adaptation of the one-object case. However, more difficult is the situation where an unknown number of objects are in the scene. The parameter space is then a mixture of discrete and continuous components and suitable techniques based on the Bayesian framework have been proposed by Grenander and Miller [22], Baddeley and van Lieshout [5] (using spatial birth–death processes) and Green [19] (using reversible jump MCMC methodology).

By suitably specifying the prior model and including penalty terms in the likelihood, issues such as overlap of objects or non-allowable neighboring objects could be built into the procedure. For example, Grenander and Miller [22] and Miller *et al.* [47] have built-in penalties to prevent overlapping of mitochondria on micrograph images. Mardia *et al.* [39] have provided an integrated approach for an unknown number of occluded multiple objects of different types following Baddeley and van Lieshout [5] and van Lieshout [54], which we now describe.

Suppose in the image I with grey levels $x = \{x_i, i \in I\}$, there are n objects $c_1, \ldots, c_n$ (n is unknown) which are any combination of q specific types $(o_1, \ldots, o_q)$, *e.g.*, $q = 3$ for circle, ellipse, square. As before, each object c_i depends on a finite number of real parameters such as size, shape and location. Let U denote the space of all possible objects so that a point $u \in U$ represents an object $R(u) \subseteq I$. Thus $c = \{c_1, \ldots, c_n\}, c_k \in U, k = 1, \ldots, n$, is an object configuration. For simplicity, let us assume that the likelihood of x given c is given by

$$L(x|c) = \prod_i f(x_i|\nu^{(c)}(i)), \tag{5.21}$$

where $f(x_i|\nu^{(c)}(i))$ is a known probability density function for the grey level x_i given $\nu^{(c)}(i)$, the population mean grey level at pixel i, and the object configuration c. Here we are assuming that x_i are conditionally independent given c. Let the *silhouette* formed by the n objects in the configuration be $S(c) = \bigcup_{k=1}^{n} R(c_k)$. Furthermore, for simplicity, assume that $\nu^{(c)}(i) = \nu_0$ if $i \in S(c)$ and ν_1 if $i \notin S(c)$. Thus ν_0 and ν_1 are foreground and background signal values as before, and we assume equal variances in the foreground and background. Van Lieshout [54] has shown that various well-known techniques from image analysis can be derived from this formulation, and we give some examples.

5.4.1 Classical Hough transform

Let the noise be *independent and identically* (*i.i.d.*) normal for each pixel. Consider a single parameterized object u in image I. Consider the hypotheses $H_0 : \nu(i) = \nu_0$ if there is an object u in I versus $H_1 : \nu(i) = \nu_1$ if there is no object in I. The log-likelihood ratio statistic for this problem is found to be proportional to

$$\sum_{i \in R(u)} x_i, \quad u \in U, \tag{5.22}$$

which is the 'sum of votes' (sum of grey levels) from all pixels belonging to the object u. This is the classical Hough transform (see, for example, [5, 54]). Usually, the objects are parameterized such as the line $x \cos\theta + y \sin\theta = \rho$. A parameterized circle used before for the mushroom is another well-known example. This formulation allows various extensions, such as multiple objects and a different form of noise density.

5.4.2 Morphological operations

Let us assume that each pixel in a binary image with background value 0 and foreground value 1, is independently distributed with probability function

$$g(x_i|\nu) = \begin{cases} \nu^{x_i}(1-\nu)^{1-x_i}, & \text{if } i \notin S(c), \\ 1, & \text{if } i \in S(c), \end{cases}$$

so only background pixels are changed and $0 < \nu < 1$ is the probability of flipping a background pixel to 1. Let X be the set of pixels with value 1. It is found that a maximum likelihood estimator (MLE) $\hat{c}$ of the objects c is

$$\hat{c} = \{u \in U : R(u) \subset X\}, \tag{5.23}$$

which is a generalized erosion operator of mathematical morphology (see [51]). Other solutions are subsets of this solution. In particular, for $U = I \subset \mathbb{R}^2$ and $R(u) = u + R$, where R is a fixed subset of I, then

$$\hat{c} = \{u : (u + R) \subseteq X\} = X \ominus R, \tag{5.24}$$

which is the classical erosion operator of X by R. Thus, the classical erosion operator is the maximum likelihood estimation under a simple noise model. The corresponding silhouette is the opening of X by R.

Similarly, by exchanging foreground and background, we obtain a similar result for the dilation and in this case the corresponding silhouette is the closing of X by R.

The Hough transform and morphology examples are two instances where one can unify well-known results of image analysis in a single statistical framework.

5.4.3 A Markovian object process

To constrain the number of objects N as well as overlap between objects, we can use an object process with the following prior density:

$$\begin{aligned} &\pi(N=n)\pi(c_1, c_2, \ldots, c_n | N=n) \\ = \;& Z^{-1}(n!)^{-1} \exp\left\{-\gamma_1 n - \gamma_2 \textstyle\sum_{c_k \sim c_l} d(c_k, c_l)\right\}, \end{aligned} \tag{5.25}$$

where Z is the partition function, γ_1 and γ_2 are two parameters where $\gamma_1 n$ describes the potential for the presence of each object, and $\gamma_2 d(c_k, c_l)$ is the interaction potential between neighboring pairs c_k and c_l. Note that if $\gamma_2 = 0$, then we see that there are no restrictions on occlusions and N is Poisson with mean $\exp(-\gamma_1)$. So larger γ_1 leads to fewer objects on an average and larger γ_2 implies limited occlusions. The sum is over all pairs of neighboring objects $c_k \sim c_l$, with $k < l$, *i.e.*, for fixed k, only neighbors of c_k contribute to the sum. For more details see [5, 39].

5.5 Warping and image averaging

5.5.1 Warping

Images can be deformed using mapping between image domains and this process is called *image warping*. Warping or morphing of images is commonplace in the television and cinema industry, although more serious applications such as medical image registration and image averaging also benefit from the technique.

Definition 5.1 Consider an image $f(t)$ defined on a region $D_1 \in \mathbb{R}^2$ and deformed to $f(\Phi(t))$, where $\Phi(t) \in D_2 \in \mathbb{R}^2$. For example, a set of landmarks T_1 could be located in the original image and then deformed to new positions T_2. We call $f(\Phi(t))$ the *warped* image. ■

A practical approach to warping is to use the inverse deformation from D_2 to D_1 to *look up* the corresponding grey levels from the region D_1. Consider a pixel location $t \in D_2$. The deformation $\Phi(t)$ is computed from the deformed region D_2 to the original plane D_1. Then the new grey level at t is assigned as the grey level $f(\Phi(t))$ (in practice, the pixel closest to location $\Phi(t)$).

The advantage of using the reverse deformation is that if the original image is defined on a regular lattice, then the warped image is still defined on a regular lattice. An alternative approach is to map from D_1 to D_2 resulting in a irregular non-square lattice for D_2, and then linear interpolation is used to obtain the final image on a regular lattice [40].

Examples of warping include data fusion for medical imaging. For example, we may wish to combine information from an X-ray image (which has anatomical information) and a nuclear medicine image (which shows functional information). Mardia and Little [43] deform an X-ray image to a nuclear medicine image. For other statistical work see [26, 28].

Glasbey and Mardia [17] have given a review of the subject. Glasbey and Mardia [18] formulate the choice of warping functions statistically as maximum penalized likelihood, where the likelihood measures the similarity between images after warping and the penalty is a measure of distortion of a warping. The paper addresses two issues simultaneously, of how to choose the warping function and how to assess the alignment. A new, Fourier-von Mises image model is identified, with phase differences between Fourier-transformed images having von Mises distributions. Also, new null-set distortion criteria are proposed, with each criterion uniquely minimized by a particular set of polynomial functions. A conjugate gradient algorithm is used to estimate the warping function, which is numerically approximated by a piecewise bilinear function. The method is motivated by, and used to solve, three applied problems: to register a remotely-sensed image with a map, to align microscope images obtained using different optics, and to discriminate between species of fish from photographic images. It also updates references since the review of Glasbey and Mardia [17].

5.5.2 Image averaging

We can use the warping approach to construct an average from several images of objects [33] .

Definition 5.2 Consider a random sample of images $f_1, \ldots, f_n$ containing landmark configuration $X_1, \ldots, X_n$, from a population mean image f with a population mean configuration μ. We wish to estimate μ and f up to arbitrary Euclidean similarity transformations. The shape of μ can be estimated by the full Procrustes mean of the landmark configurations $X_1, \ldots, X_n$. Let Φ_i^* be the deformation obtained from the estimated mean shape $[\hat{\mu}]$ to the ith configuration. The *average image* has the grey level at pixel location t given by

$$\bar{f}(t) = \frac{1}{n} \sum_{i=1}^{n} f_i\{\Phi_i^*(t)\}. \tag{5.26}$$

■

Galton [13, 14] obtained averaged pictures of faces using photographic techniques over a century ago and the technique was called composite photography. He was interested in the typical average face of groups of people, such as criminals and tuberculosis patients. He believed that faces contained information that could be used in many applications, *e.g.*, the susceptibility of people to particular diseases. Another early example is the *average* photograph of 12 American mathematicians obtained by Raphael Pumpelly taken in 1884 [53]. Shapes of landmark configurations from photographs of the faces of individuals are currently being used for forensic identification (see, for example, [36, 39].

5.6 Discussion

Over the last decade, statistical algorithms have emerged for image analysis which are widely applicable and reliable. There is also an advantage in using explicit stochastic models so that we have a better understanding of the working behind the algorithms and we can make confidence statements about conclusions.

We have treated here only one aspect where statistics has an impact on image analysis. Another area is where the aim is not only object recognition but knowledge representation of objects, such as the creation of anatomical atlases of the brain. In such cases deformable templates and associated probabilities distributions can be used to describe normal subjects and patients. Grenander and his colleagues are playing a key rôle in this area. Their recent work in brain mapping has been extended to higher dimensional manifolds, *i.e.*, landmarks (dimension 0) to sulci (lines of dimension 1), cortex and related surfaces (dimension 2), volumes and sub-volumes (dimension 3) (see, for example, [22]). Another important area of image analysis is the use of scale space techniques (*e.g*, [56]). For example, cores have been used for image registration (*e.g.*, [12]). The core is formed by stacking medial transforms of an image at various scales. Other methods incorporating multiscale methods include deformable templates in scale space (*e.g.*, [55]). Further aspects of shape including scale space ideas can be found in the volume edited by Ying-Lie *et al.* [57] and in the book by Dryden and Mardia [11].

There are various other developments in statistical image analysis where shape is important, including image sequences and robust vision (see, for example, [42, 48]. Further examples can be seen in the edited volumes [34, 37, 38, 41, 45].

For a recent collection of papers in deformable models in medical image analysis, see [52]; for an excellent review of general deformable templates, see [27]. Grenander and Miller [23] describe the state of the art in computational anatomy. Hallinan *et al.* [24] provide many innovative ideas on pattern analysis as applied to the face in particular. Zhu [58] has developed statistical models for shape primitives in particular.

To sum up, image analysis continues to provide new and challenging statistical methodology, and shape analysis often plays a vital rôle.

References

[1] Amit, Y. (1997). Graphical shape templates for automatic anatomy detection with applications to MRI scans. *IEEE Transactions on Medical Imaging*, 16:28-40.

[2] Amit, Y. and Geman, D. (1996). Shape quantization and recognition with randomized trees. Technical report, Department of Statistics, University of Chicago.

[3] Amit, Y. and Kong, A. (1993). Graphical templates for image matching. Technical Report 373, Department of Statistics, University of Chicago.

[4] Amit, Y. and Kong, A. (1996). Graphical templates for model registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:225-236.

[5] Baddeley, A. J. and van Lieshout, M. N. M. (1993). Stochastic geometry models in high-level vision. In Mardia, K. V. and Kanji, G. K., editors, *Statistics and Images: Vol. 1*, pages 231-256. Carfax, Oxford.

[6] Besag, J. E. (1986). On the statistical analysis of dirty pictures (with discussion). *Journal of the Royal Statistical Society, Series B*, 48:259-302.

[7] Besag, J., Green, P.J., Higdon, D. and Mengersen, K. (1995). Bayesian computation and stochastic systems (with discussion). *Statistical Science*, 10:3-66.

[8] Bookstein, F. L. (1996). Applying landmark methods to biological outline data. In Mardia, K. V., Gill, C. A. and Dryden, I. L., editors, *Proceedings in Image Fusion and Shape Variability Techniques*, pages 59-70, Leeds. University of Leeds Press.

[9] Bookstein, F. L. (1996). Landmark methods for forms without landmarks: morphometrics of group differences in outline shape. *Medical Image Analysis*, 1:225-243.

[10] Cootes, T. F., Taylor, C. J., Cooper, D. H. and Graham, J. (1992). Training models of shape from sets of examples. In Hogg, D. C. and Boyle, R. D., editors, *British Machine Vision Conference*, pages 9-18, Berlin. Springer-Verlag.

[11] Dryden, I. L. and Mardia, K. V. (1998). *Statistical Shape Analysis*. Wiley, Chichester.

[12] Fritsch, D. S., Pizer, S. M., Chaney, E. L., Lui, A., Raghavan, S. and Shah, T. (1994). Cores for image registration. In *Proceedings of SPIE Medical Imaging '94*, volume 2167, pages 128-142.

[13] Galton, F. (1878). Composite portraits. *Journal of the Anthropological Institute of Great Britain and Ireland*, 8:132-142.

[14] Galton, F. (1883). *Enquiries into Human Faculty and Development*. Dent, London.

[15] Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 6:721-741.

[16] Gilks, W. R., Richardson, S. and Spiegelhalter, D. J., editors (1996). *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London.

[17] Glasbey, C. A. and Mardia, K. V. (1998). A review of image warping methods. *Journal of Applied Statistics*. 25:155-171.

[18] Glasbey, C. A. and Mardia, K. V. (2001). A penalized likelihood approach to image warping. *Journal of the Royal Statistical Society, Series B* (with discussion).

[19] Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711-732.

[20] Grenander, U. (1994). *General Pattern Theory*. Clarendon Press, Oxford.

[21] Grenander, U. and Keenan, D. M. (1993). Towards automated image understanding. In Mardia, K. V. and Kanji, G. K., editors, *Statistics and Images: Vol. 1*, pages 89-103. Carfax, Oxford.

[22] Grenander, U. and Miller, M. I. (1994). Representations of knowledge in complex systems (with discussion). *Journal of the Royal Statistical Society, Series B*, 56:549-603.

[23] Grenander, U. and Miller, M. I. (1998). Computational anatomy: an emerging discipline. *Quarterly of Applied Mathematics*, 56:617-694.

[24] Hallinan, P. W., Gordon, G. G., Yuille, A. L., Giblin, P. and Mumford, D. (1999). *Two- and Three-Dimensional Patterns of the Face*. A.K. Peters, Mass.

[25] Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97-109.

[26] Hurn, M. A., Mardia, K. V., Hainsworth, T. J., Kirkbride, J. and Berry, E. (1996). Bayesian fused classification of medical images. *IEEE Transactions on Medical Imaging*. 15:850-858.

[27] Jain, A. K., Zhong, Y. Dubuisson-Jolly, M. P. (1998). Deformable template models: A review. *Signal Processing*, 71:109-129.

[28] Johnson, V. E., Bowsher, J. E., Jaszczak, R. J. and Turking, T. G. (1995). Analysis and reconstruction of medical images using prior information. In Gastonis, C., Hodges, J. S., Kass, R. E. and Singpurwalla, N. D., editors, *Case Studies in Bayesian Statistics Vol. 11*, pages 149-218. Springer-Verlag, Berlin .

[29] Kass, M., Witkin, A. and Terzopoulos, D. (1988). Snakes: active contour models. *International Journal of Computer Vision*, 1:321-331.

[30] Kent, J. T., Anderson, C. R. and Dryden, I. L. (1995). Deformations with circulant symmetry. Technical Report STAT95/17, University of Leeds.

[31] Mardia, K. V. (1989a). Discussion to 'A survey of the statistical theory

of shape' by D.G. Kendall. *Statistical Science*, 4:108-111.

[32] Mardia, K. V. (1989). Markov models and Bayesian methods in image analysis. *Journal of Applied Statistics*, 16:125-130.

[33] Mardia, K. V. (1993). Discussion to papers on 'Gibbs sampler and other MCMC methods'. *Journal of the Royal Statistical Society, Series B*, 55:83-84.

[34] Mardia, K. V., editor (1994). *Statistics and Images: Vol. 2.* Carfax, Oxford.

[35] Mardia, K. V. (1997). Bayesian image analysis. *Journal of Theoretical Medicine.* 1:63-77.

[36] Mardia, K. V., Coombes, A., Kirkbride, J., Linney, A. and Bowie, J. L. (1996a) On statistical problems with face identification from photographs. *Journal of Applied Statistics*, 23:655-675.

[37] Mardia, K. V. and Gill, C. A., editors (1995). *Current Issues in Statistical Shape Analysis.* Proceedings of the Leeds Annual Statistics Research Workshop, Leeds. University of Leeds Press.

[38] Mardia, K. V., Gill, C. A. and Aykroyd, R. G., editors (1997). *The Art and Science of Bayesian Image Analysis.* Proceedings of the Leeds Annual Statistics Research Workshop, Leeds. University of Leeds Press.

[39] Mardia, K. V., Gill, C. A. and Dryden, I. L., editors (1996). *Image Fusion and Shape Variability.* Proceedings of the Leeds Annual Statistics Research Workshop, Leeds. University of Leeds Press.

[40] Mardia, K. V. and Hainsworth, T. J. (1993). Image warping and Bayesian reconstruction with grey-level templates. In Mardia, K. V. and Kanji, G. K., editors, *Statistics and Images: Vol. 1*, pages 257-280. Carfax, Oxford.

[41] Mardia, K. V. and Kanji, G. K., editors (1993). *Statistics and Images: Vol. 1*, Oxford. Carfax, Oxford.

[42] Mardia, K. V., Kent, J. T. and Walder, A. N. (1991). Statistical shape models in image analysis. In Keramidas, E. M., editor, *Computer Science and Statistics: Proceedings of the 23rd INTERFACE symposium*, pages 550-557. Interface Foundation, Fairfax Station.

[43] Mardia, K. V. and Little, J. L. (1994). Image warping using derivative information. In Bookstein, F. L., Duncan, J. S., Lange, N. and Wilson, D. C., editors, *Mathematical Methods in Medical Imaging III, Proceedings*, Vol. 2299, pages 16-31. SPIE, Washington.

[44] Mardia, K. V. and Qian, W. (1995). Bayesian method for compact

object recognition from noisy images. In Titterington, D. M., editor, *Complex Stochastic Systems in Science and Engineering*, pages 155-165. Clarendon Press, Oxford.

[45] Mardia, K. V., Qian, W., Shah, D. and De Souza, K. (1996c). Deformable template recognition of multiple occluded objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:1036-1042.

[46] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087-1092.

[47] Miller, M. I., Joshi, S., Maffit, D. R., McNally, J. G. and Grenander, U. (1994). Membranes, mitochondria and amoebae: shape models. In Mardia, K. V., editor, *Statistics and Images: Vol. 2*, pages 141-163. Carfax, Oxford.

[48] Phillips, D. B. and Smith, A. F. M. (1993). Dynamic image analysis using Bayesian shape and texture models. In Mardia, K. V. and Kanji, G. K., editors, *Statistics and Images: Vol. 1*, pages 299-322. Carfax, Oxford.

[49] Phillips, D. B. and Smith, A. F. M. (1994). Bayesian faces via hierarchical template modeling. *Journal of the American Statistical Association*, 89:1151-1163.

[50] Ripley, B. D. and Sutherland, A. I. (1990). Finding spiral structures in galaxies. *Philosophical Transactions of the Royal Society of London, Series A*, 332:477-485.

[51] Serra, J. (1982). *Image Analysis and Mathematical Morphology.* Academic Press, London.

[52] Singh, A., Goldgof, D. and Terzopoulos, D., editors (1998). *Deformable Models in Medical Image Analysis.* IEEE Computing Society, Los Alamitos, CA.

[53] Stigler, S. M. (1984). Can you identify these mathematicians? *Mathematical Intelligence*, 6:72.

[54] van Lieshout, M. N. M. (1995). *Stochastic geometry models in image analysis and spatial statistics.* CWI Tract 108, Amsterdam.

[55] Wilson, A. (1995). *Statistical Models for Shape Deformations.* PhD thesis, Duke University, Durham.

[56] Witkin, A. (1983). Scale-space filtering. In Bundy, A., editor, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, Germany*, pages 1019-1022. Kaufman, Los

Altos .

[57] Ying-Lie, O., Toet, A., Foster, D., Heijmans, H. J. A. M. and Meer, P., editors (1994). *Proceedings of the NATO Conference on Shape in Pictures.* Springer-Verlag, Berlin.

[58] Zhu, S. C. (1999). Embedding Gestalt laws in Markov random fields - a theory for shape modeling and perceptual organization. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 21:1170-1187.

Chapter 6

DECISION TREES FOR CLASSIFICATION : A REVIEW AND SOME NEW RESULTS

R. Kothari and M. Dong

Artificial Neural Systems Laboratory
Department of Electrical & Computer Engineering
& Computer Science
University of Cincinnati
Cincinnati, OH 45221-0030, U.S.A.
e-mail: *{rkothari,mdong}@ececs.uc.edu*

Abstract

Decision trees represent a simple and powerful method of induction from labeled examples. Due to the fact that larger decision trees produce poorer test performance, considerable amount of research has been directed towards producing decision trees of small size and depth. From this perspective, we present an overview of decision tree induction algorithms and approaches and propose a novel method of look-ahead based decision tree construction.

6.1 Introduction

Top-down induction of decision trees is a simple and powerful method of inferring classification rules from a set of labeled examples [27]. Each node of the tree implements a decision rule that splits the examples into two or more partitions. New nodes are created to handle each of the partitions and a node is considered terminal or a leaf node based on a stopping criterion. This standard approach to decision tree construction thus corresponds to a top-down greedy algorithm that makes locally optimal decisions at each node.

There are two advantages that decision trees have over many other methods of classification methods. The first is that the sequence of decisions made from the root node to the eventual labeling of a test input is easy to follow. This gives them an intuitive appeal that other methods of classification such as feed-forward neural networks lack [33]*. The second is the ease with which they can be extended to non-numeric domains where the attributes are categorical rather than numerical.

One of the challenges in decision tree induction is to develop algorithms that produce decision trees of small size and depth. In part, smaller decision trees lead to lesser computational expense in determining the class of a test example. More significantly however, larger decision trees lead to poorer generalization (test) performance [1]. Motivated by these considerations, a large number of approaches have been proposed towards producing smaller decision trees. Broadly these may be classified into three categories,

(C1) The first category includes those efforts which suggest different criteria to partition the examples at each node. Some examples of the different node splitting criteria include entropy or its variants [27], the chi-square statistic [15, 22], the G statistic [22], and the GINI index of diversity [1]. Despite these efforts, there appears to be no single node splitting that performs the best in all cases [2, 23]; nonetheless there is little doubt that random splitting performs the worst.

(C2) The second category is based on pruning a decision tree either during the construction of the tree or after the tree has been constructed. In either case, the idea is to remove branches will little statistical validity based on some criteria [9, 24, 28, 37].

(C3) The third category of efforts towards producing smaller decision trees is motivated by the fact that a locally optimum decision at a node may produce incorrectly classified instances that require a large number of additional nodes for classification. The so-called look-ahead methods attempt to establish a decision at a node by analyzing the classifiability of examples resulting from the split [7, 25, 30, 35].

Our goal here is to review a *sampling* of the different algorithms in each

*Of course, there are several advantages that feed-forward neural networks enjoy over decision trees, primary among them being that of scalability.

of the three categories mentioned above. A graph-theory–oriented review of decision tree induction algorithms has also been done [34]. Finally, we suggest a new approach to constructing decision trees which utilizes look-ahead to produce smaller decision trees.

6.2 The different node splitting criteria

Prior to reviewing the different node splitting criteria we introduce some notation used throughout this Chapter. We assume there are a total of C classes denoted by $\Omega = \{\omega_1, \omega_2, \ldots, \omega_C\}$. On the basis of some (possibly noisy) examples of input-output observations the goal is to construct a decision tree which upon construction can "reliably" assign a test input to a specific class.

At a particular node in the tree, let there be N training examples represented by,

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})$$

where, $x^{(i)}$ is a vector of n attributes and $y^{(i)} \in \Omega$ is the class label corresponding to the input $x^{(i)}$. Of these N examples, N_{ω_k} belong to class ω_k, $\sum_k N_k = N$. The decision rule at the node splits these examples into V partitions, or V child nodes, each of which has $N^{(v)}$ examples. In a particular partition, the number of examples of class ω_k is denoted by ${N_k}^{(v)}$. $\sum_k {N_k}^{(v)} = N^{(v)}$.

There are two primary variants of the different node splitting criteria for decision tree induction; those that use a single attribute to split the examples and those that use all the attributes to split the examples. When based on a single attribute, the partition boundary is orthogonal to the chosen attribute. Typically, when all the attributes are used, the partition is based on a linear discriminant, *i.e.*, the partitioning is achieved using a hyperplane though higher-order parameterizations are possible. In the following, we present some of the more commonly used node splitting criteria.

6.2.1 Information gain based node splitting

Partitioning on the basis of information gain (or its variants) is perhaps the most commonly used node splitting criteria. It also forms the basis of the popular ID3 algorithm [27] and is based on choosing the attribute that results in the largest decrease in entropy. More specifically the information gain resulting from splitting the examples bases on attribute x_j can be written as,

$$G(x_j) = \left[\sum_{k=1}^{C} -\left(\frac{N_k}{N}\right)\log\left(\frac{N_k}{N}\right)\right] - \left[\sum_{v=1}^{V}\left(\frac{N^{(v)}}{N}\right)\sum_{k=1}^{C} -\left(\frac{N_k^{(v)}}{N^{(v)}}\right)\log\left(\frac{N_k^{(v)}}{N^{(v)}}\right)\right] \quad (6.1)$$

The first term in (6.1) is the entropy at the parent node and the second term is the weighted entropy of the child nodes. The difference thus reflects the decrease in entropy or the information gained from the use of attribute x_j. The attribute chosen is the one that results in the largest information gain.

Though very popular, one difficulty that arises is that the information gain criterion as defined above favors a large number of partitions (V). To discourage a large number of partitions a factor based on the entropy of the size of the splits was proposed in C4.5 [29]. More specifically,

$$g = \sum_{v=1}^{V}\left(\frac{N^{(v)}}{N}\right)\log\left(\frac{N^{(v)}}{N}\right) \quad (6.2)$$

$G(x_j)/g$ can then be used.

A closely related node splitting criteria is the G statistic which is proportional to $NG(\cdot)$ which gives the approximate number of bits by which a split would compress the data.

6.2.2 Chi-squared statistic based node splitting

The chi-squared statistic (χ^2) based partitioning is based on comparing the obtained values of the frequency of a class because of the split to the *a priori* frequency of the class [15, 22]. More specifically,

$$\chi^2 = \sum_{k=1}^{C}\sum_{v=1}^{V}\frac{(N_k^{(v)} - \tilde{N}_k^{(v)})^2}{\tilde{N}_k^{(v)}} \quad (6.3)$$

where, $\tilde{N}_k^{(v)} = (N^{(v)}/N)N_k$ denotes the *a priori* frequency. Clearly, a larger value of χ^2 indicates that the split is more homogeneous, *i.e.*, has a greater frequency of instances from a particular class. The attribute chosen is the one with the largest value of χ^2. The G statistic mentioned above approximates the χ^2 distribution.

6.2.3 GINI index of diversity based node splitting

The GINI index of diversity is based on

$$D(x_j) = \frac{1}{N}\left[\sum_{k=1}^{C}\sum_{v=1}^{V}\frac{{N_k^{(v)}}^2}{N^{(v)}} - \sum_{k=1}^{C}\frac{{N_k}^2}{N}\right] \tag{6.4}$$

Typically we would like a node to be "pure", *i.e.*, have instances of a single class. Similar to the decrease in entropy (or gain in information) used in the information gain based node splitting methods, here the decrease in "impurity" as given by (6.4) is used. The attribute chosen is the one which results in the largest decrease in impurity.

6.2.4 Discriminant based node splitting

The decision rule used in the above approaches uses a single attribute. It is also possible to use a decision rule based on all the attributes. Typically, when all the attributes are used, the partition is based on a linear discriminant, *i.e.*, the partitioning is achieved using a hyperplane. Higher-order parameterizations of the discriminant are also possible, though one should be aware of the decrease in the number of examples available as one moves further down the decision tree. The task then simply becomes that of finding the parameters of the discriminant that result in the most number of correct classifications at each node. The resulting tree in this case of course is binary, *i.e.*, each node has a left child and a right child corresponding to the two sides of a hyperplane.

Several additional criteria for node splitting have also been proposed (for example, the exact probability metric based on Fisher's Exact Test [19], the orthogonality metric based splitting [8],the Kolmogorov-Smirnov distance and test based splitting [32] and several others for single attribute based partitioning). Additionally, more complex decision rules have also been suggested for use at each node (for example, the use of a neural network at each internal node [10, 13]). As mentioned before, there appears to be no single node splitting that performs the best in all cases [2, 23]. On an average, any of the criteria discussed above are good candidates for node splitting. Indeed, it is unlikely that decision trees generated on the basis of a superior node splitting criteria are likely to be significantly smaller or more accurate. In part, this arises due to greedy, local and irreversible nature of the decisions made at each node. A sequence of locally optimal decision does not guarantee an optimal or close to optimal result.

6.3 Pruning

The size of the overall tree strongly influences the generalization performance obtained from the tree based classifier [1]. Pruning† is the removal of sub-trees of the tree that have little statistical validity [9, 24, 28, 37] thereby obtaining a tree with smaller size. Of course, there is no guarantee that pruning will improve the (some conditions appear in [16]).

In general, pruning is done after a tree has been constructed. However, an implicit form of pruning may be based on a using a stopping criteria which prevents the creation of a child sub-tree at a given node. Typically, a node is prevented from splitting further depending on the return indicated by these criteria; for example, the node is stopped from splitting further when the information gain falls below a threshold. Of course, preventing further expansion of the node is based on a criteria that also utilizes local information in the sense that it is hard to estimate what would happen a few levels below the node being investigated for pruning. A variation of the above two themes is when construction of the tree is interlaced with pruning [11].

The more popular pruning methods are based on removing a node (sub-tree) after the tree has been constructed. Of course, sequentially removing nodes is not without its drawbacks. In particular, there is considerable computational difficulty in evaluating higher order removals (*i.e.*, at each step the "least useful" node is removed; this corresponds to a greedy strategy which may not be the most optimal). We present some of the popular post-construction pruning approaches below. In general the approaches are based on estimating the sensitivity of a sub-tree to some measure (the error rate for example) and removing those sub-trees which have minimal impact on the measure.

6.3.1 Error complexity based pruning

One popular method of pruning is the so-called error complexity based pruning [1]. It is based on,

$$E_{\text{er}} = \frac{\tilde{R} - R}{L} \tag{6.5}$$

†The generalization or prediction error can be shown to be composed of the squared bias plus the variance. In that context, pruning introduces an additional bias with the expectation that the variance would reduce by a larger amount.

where, R denotes the error rate (probability of error) of the unpruned tree, $\tilde{R}$ is the error rate after a node is removed, and L is the number of leaf nodes in the sub-tree of the node being evaluated for pruning. The error rate is computed based on a set of instances independent of the set used for constructing the tree[‡]. E_{er} is computed for each non-terminal node and the node with the smallest error complexity measure is removed.

6.3.2 Minimum error based pruning

The minimum error based pruning [26] is based on the following equation for computing the expected error rate,

$$E_{\text{me}} = \frac{N - N_k + C - 1}{N + C} \tag{6.6}$$

where, N is the number of instances at a given node, and N_k are the number of instances of the *dominant* class. The expected error rate of the unpruned tree is computed by computing the expected error rate using (6.6) for each branch weighted by the number of instances in that branch. The expected error rate if a node is pruned (*i.e.*, all its children are removed) is then computed based on the instances at that node. If the expected error rate with pruning is lower than without, the children below the node are pruned and the node is made into a terminal node.

There are several other heuristic methods of pruning that have been suggested. The critical value method [24] for example, uses a threshold on the values obtained during node splitting to remove sub-trees. The reduced error method of pruning [28] is based on using an independent dataset and observing the number of errors at each non-terminal node when a node is retained and when the child sub-tree is removed. The child sub-tree corresponding to the node which results in the largest decrease in the number of misclassifications is removed. Other methods based on minimal description length [31] can also be used.

While pruning can, in many cases, improve the generalization accuracy, there remain two significant drawbacks associated with it. First, most pruning algorithms have poor time complexities (often $O(N^3)$ of $O(N^4)$ or higher).

[‡]When the amount of data is limited, one has to resort to creating an independent set based on resubstitution based sampling of the data to create the data used for evaluation. However, the error rate is underestimated in such a case is underestimated and corrections need to be applied.

Second, pruning is itself based on a greedy strategy and as such cannot guarantee the optimality of the solution or the vicinity of a solution to the optimal one.

6.4 Look-ahead

The use of look-ahead in decision tree induction is to examine the examples in each partition resulting from a node split. Due to the local nature of the decision made at each node, it is entirely possible that a split produces partitions that require a large number of additional nodes for classification. The central question in look-ahead is: *What criteria is used to implement the look-ahead policy?* In the approaches proposed thus far, look-ahead uses the same criteria as that used in the greedy algorithm to split the node. For example, when a linear discriminant is used to split a node, then the look-ahead is based on using a linear discriminant in each of the resulting partitions to evaluate the classifiability of the partitions. In general, the look-ahead can be a z-step look-ahead where z is an integer ≥ 0. $z = 0$ corresponds to a purely greedy algorithm with no look-ahead and a "very large" value of z corresponds to an exhaustive search.

Strangely, mixed results are reported (ranging from look-ahead makes no difference to look-ahead produces larger trees [25]). There are fewer reports of look-ahead being generally beneficial to the goal of generating smaller decision trees [35].

6.5 Other issues in decision tree construction

As with all *empirical* approaches, appropriate validation of decision trees is important prior to use in any application. For proper validation any of the methods based on sampling without replacement (such as cross-validation [36], hold-out methods) or sampling with replacement (such as bootstrap [5, 6]) can be used [18]. Some analysis of the sample size sufficient for decision tree induction based on pruning has also been done [17].

Many of the open problems in pattern recognition such as that of missing data also affect decision tree construction. Within the context of decision tree construction however the modal method is claimed to provide reasonable results [27]. This method replaces the missing value of an attribute in an instance by the most commonly occurring value of that attribute amongst the examples at that node of the same class. Related to the issue of missing data is that of *don't care* data. The distinction between missing and don't care data is that the former is relevant to the class of training examples while latter is irrelevant [4].

6.6 A new look-ahead criterion: some new results

In this section, we propose a novel look-ahead criterion and present some results obtained with it. In part, we believe that poor results in prior research with look-ahead is due to the criterion and the mechanism in which look-ahead is implemented. Consider for example, Fig. 6.1 which shows examples of two classes in a specific partition resulting from the decision rule made at a node. Look-ahead with a linear discriminant would predict poor classifiability if a 1-step look-ahead is used even though in this case a 2-step look-ahead would have resulted in a vastly different conclusion. In general of course, when a z-step look-ahead is used, it is entirely possible that a $(z+1)$-step look-ahead would result in substantially different results.

Our approach to look-ahead is based on the notion of texture that is commonly used in image processing [14]. To clarify, consider a 2-class classification problem where each instance has n attributes (variables) and a single variable represents the class label. It is possible to visualize a surface in $(n+1)$ dimensions (n input variables and 1 variable for the class label). For example, the data shown in Fig. 6.1 can be visualized in 3-dimensions where two of the dimensions represent x_1 and x_2 and the third dimension represents the class label. This third dimension is 1 for one class and 0 for the other class. When instances of different classes are interlaced the surface is rough (moving rapidly between 0 and 1). Homogeneous regions of instances of the same class correspond to constant (smooth) patches of the surface. Clearly a rough surface corresponds to a situation in which classification would be considerably difficult. The smoothness of the *class-label surface* thus directly corresponds to the classifiability of the instances and can be used in evaluating the classifiability of the examples resulting in each partition resulting from a node split. When compared to present methods of look-ahead the proposed method is superior in that it directly tries the obtain the classifiability from the structure and distribution of the data and does not require an arbitrary choice of the number of look-ahead steps.

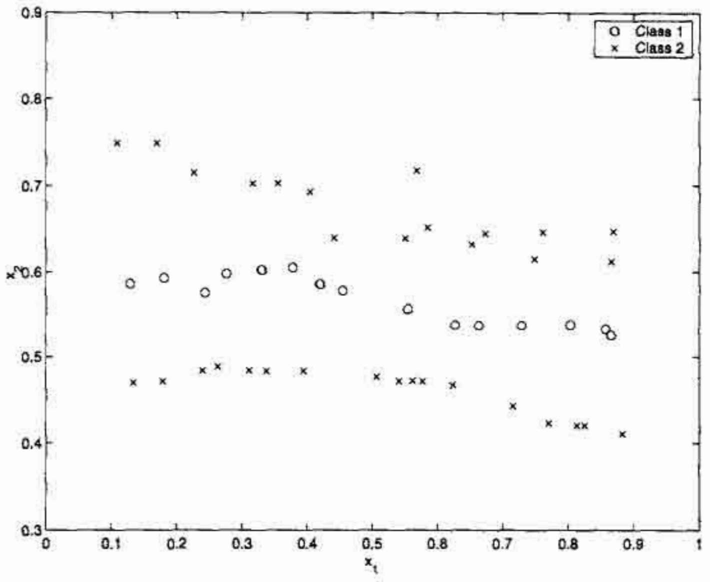


Fig. 6.1 A 1-step look-ahead with a linear discriminant would predict poor classifiability. Substantially different results are obtained if a 2-step look-ahead is used in this case

The basis of obtaining the classifiability for look-ahead is to find a class co-occurrence matrix. More specifically, for each of the partitions, we find a matrix $A^{(v)}$ of dimension $C \times C$ (recall that C is the number of classes). An element of $A^{(v)}$, say, $A_{jk}^{(v)}$, represents the number of instances of class ω_k that occur within a circular neighborhood of radius r of an instance of class ω_j, *i.e.*,

$$A_{jk}^{(v)} = \sum_{l=1}^{N_{\omega_j}^{(v)}} \sum_{m=1}^{N_k^{(v)}} f(x^{(l)}, x^{(m)}) \tag{6.7}$$

where, $x^{(l)}$ and $x^{(m)}$ are instances of class ω_j class ω_k respectively in the v^{th} partition, and $f(\cdot)$ is an indicator function which is 1 if $\| x^{(l)} - x^{(m)} \| \leq r$. The overall class co-occurrence matrix is simply,

$$A = \sum_{v=1}^{V} A^{(v)} \tag{6.8}$$

In the preferred case A would become strongly diagonal. With increasing confusion A becomes less and less diagonally dominant. The classifiability can then be expressed as,

$$L = \sum_{i=1}^{C} A_{ii} - \sum_{i=1}^{C} \sum_{\substack{j=1 \\ j \neq i}}^{C} A_{ij} \tag{6.9}$$

Assuming a linear discriminant is used as the decision rule at each node, the algorithm for for decision tree induction can then be based on maximizing,

$$J = G + \lambda L \tag{6.10}$$

where, G is similar to that defined in (6.1) with the exception that since a linear discriminant is proposed, G is not a function of a particular attribute. λ in (6.10) is a Lagrange parameter and controls the relative weighting between the information gained (number of instances correctly classified) and the classifiability of the incorrectly classified instances. $\lambda = 0$ implies no look-ahead in which case the proposed method simply uses information gain.

Since the objective function J in (6.10) is not continuous with respect to the parameters of the linear discriminant, the maximization and the identification of the discriminant has to be done using techniques other than gradient descent. We have found Genetic Algorithms [12, 21] to provide good results. However, some experimentation may be required to get an appropriate value of λ and r.

We present some results obtained with the proposed texture based look-ahead decision tree construction algorithm. In each case, the genetic algorithm was run for 600 generations and the population size was 300. The first result is based on a simple 2 class classification problem and allows for easy visualization. The top panel Fig. 6.2 shows the sequence of decisions made with $\lambda = 0$, *i.e.*, no look-ahead. The sequence of decision are identified within parentheses in the panel; for example "(1)" implies that it was the first hyperplane established, *i.e.*, at the root node. The bottom panel of Fig. 6.2 shows the sequence of decisions with $\lambda = 3$ and $r = 0.5$. It is clear that the use of the proposed texture based look-ahead results in a smaller decision tree. Indeed, for the case with no look-ahead the information gain at the root node was 0.3113 while with look-ahead the information gain was only 0.2543 for the choice that was made. This smaller information gain however was a better choice due to the structure and classifiability of the remaining instances and is demonstrated in Fig. 6.2.

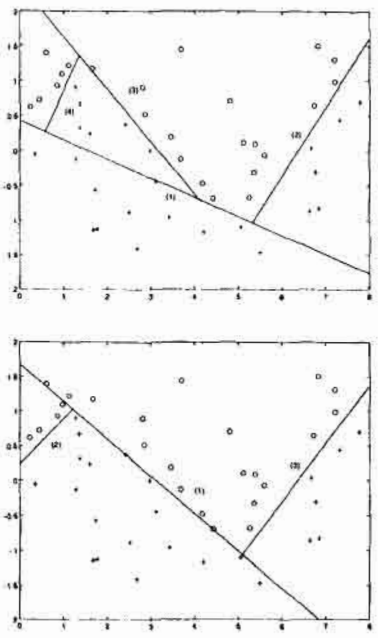


Fig. 6.2 Results obtained with a 2 class classification problem. The top panel is without the use of look-ahead (*i.e.*, a greedy strategy, $\lambda = 0$ in (6.10)). The bottom panel is with look-ahead and $\lambda = 3$

For the second simulation we used the glass classification problem [20]. There are a total of 10 attributes (such as refractive index, weight percentages of Sodium, Magnesium, Aluminum, Silicon, Potassium, Calcium, Barium, Iron and an "ID" attribute). The ID attribute was ignored and a 2 class classification problem was created. The class label denotes whether the glass can be used for "windows" (buildings, vehicles etc.) or not. There are a total of 214 instances (163 window glass and 51 non-window glass) and there are no missing attributes. As before, we ran the simulations with no look-ahead ($\lambda = 0$) and with look-ahead ($\lambda = 1$ and $r = 3$). In each case we constructed the decision tree to provide 100% accuracy on the training dataset. When no look-ahead is used the total number of nodes obtained were 15 with 8 leaf nodes. With look-ahead the total number of nodes were 9 with 5 leaf nodes. For comparison, ID3 produces 21 nodes with 11 leaves. Though ID3 also uses information gain, it uses a single attribute for the decisions at each node. In contrast, the result without look-ahead in our case is based on a linear discriminant at each decision node. Clearly, the proposed look-ahead based methods produces results considerably superior to that obtained without look-ahead when either a linear discriminant is used at a node or when single attribute based decision is implemented at each node.

6.7 Conclusions

Decision trees represent a simple and powerful method of induction from labeled examples. We reviewed some of the more popular node splitting criteria, examined the issue of pruning as well as look-ahead for constructing decision trees. While, the discussion here was restricted to crisp decision trees, many of the concepts presented here have also migrated to fuzzy decision tree induction [3, 38].

Generating a decision tree with the minimum number of leaves is NP-hard [37]. However, we believe that look-ahead is the most promising and the least studied of approaches aimed at producing decision trees of small size and depth. Towards that we also presented a novel look-ahead algorithm which is based on examining the joint distribution of the instances of difference classes.

Acknowledgment

Part of this work was done when Dr. Kothari was on a sabbatical leave from the University of Cincinnati.

References

[1] L. Breiman, J. H. Friedman, J. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA, 1984.

[2] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1989.

[3] K. J. Cios and L. M. Sztandera. Continuous ID3 algorithm with fuzzy entropy measures. In *Proceedings of IEEE International Conference on Fuzzy Systems*, pages 469–476, 1992.

[4] N. Diamantidis and E. A. Giakoumakis. Don't care values in induction. *Artificial Intelligence*, 8:505–514, 1996.

[5] B. Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7:1–26, 1979.

[6] B. Efron and T. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, New York, NY, 1993.

[7] J. F. Elder. Heuristic search for model structure. In D. Fischer and H-J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V, Lecture Notes in Statistics*, volume 112, pages 131–142. Springer-Verlag, Berlin, 1995.

[8] U. M. Fayyad and K. B. Irani. The attribute selection problem in decision tree generation. In *Proc.* 10th *National Conference on Artificial Intelligence*, pages 104–110. MIT Press, Cambridge, Massachusetts, 1992.

[9] J. Fürnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27:139–172, 1997.

[10] S. Gelfand and H. Guo. *Tree Classifiers With Multilayer Perceptron Feature Extraction*. PhD thesis, School of Electrical Engineering, Purdue University, West Lafayette, 1991.

[11] S. B. Gelfand, C. S. Ravishankar, and E. J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:163–174, 1991.

[12] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[13] M. Golea and M. Marchand. A growth algorithm for neural networks. *Europhysics Letters*, 12(3):205–210, 1990.

[14] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*. Addison-Wesley, Reading, Massachusetts, 1992.

[15] A. Hart. Experience in the use of inductive system in knowledge engineering. In M. Bramer, editor, *Research and Developments in Expert Systems*, Cambridge, 1984. Cambridge University Press.

[16] H. Kim and G. J. Koehler. An investigation on the conditions of pruning an induced decision tree. *European Journal of Operational Research*, 77:82–95, 1994.

[17] H. Kim and G. J. Koehler. Theory and practice of decision tree induction. *Omega, International Journal of Management Science*, 23(6):637–652, 1995.

[18] R. Kothari. Prediction error: The bias/variance decomposition, methods of minimization, and estimation. In M. L. Padgett, N. Karayiannis,

and L. Zadeh, editors, *Handbook of Applied Computational Intelligence*. CRC Press, Boca Raton, FL, 2000.

[19] J. K. Martin. An exact probability metric for decision tree splitting and stopping. *Machine Learning*, 28:257–291, 1997.

[20] C. J. Merz and P. M. Murphy. UCI repository of machine learning databases. Technical report, Department of Information and Computer Science, University of California at Irvine, [http:// www.ics.uci.edu/ ~mlearn/MLRepository.html], 1996.

[21] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1994.

[22] J. Mingers. Expert systems - experiments with rule induction. *Journal of the Operational Research Society*, 38:39–47, 1987.

[23] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.

[24] J. Mingers. An empirical comparison of pruning methods for decision-tree induction. *Machine Learning*, 4:227–243, 1989.

[25] S. K. Murthy and S. Salzberg. Lookahead and pathology in decision tree induction. In *Proc.* 14th *International Conference on Artificial Intelligence*, pages 1025–1031, San Mateo, California, 1995. Morgan Kaufman.

[26] T. Niblett and I Bratko. Learning decision rules in noisy domains. In M. A. Bramer, editor, *Research and development in Expert Systems III*, pages 25–34. Cambridge University Press, Cambridge, 1986.

[27] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[28] J. R. Quinlan. Simplifying decision trees. *Internation Journal of Man-Machine Studies*, 27:221–234, 1987.

[29] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[30] J. R. Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proc.* 14th *International Conference on Artificial Intelligence*, pages 1019–1024, San Mateo, California, 1995. Morgan Kaufman.

[31] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[32] E. Rounds. A combined non-parametric approach to feature selection and binary decision tree design. *Pattern Recognition*, 12:313–317, 1980.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by back-propagating errors. *Nature*, 332:533–536, 1986.

[34] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21:660–674, 1991.

[35] U. K. Sarkar, P. P. Chakrabarti, S. Ghose, and S. C. DeSarkar. Improving greedy algorithms by look-ahead search. *Journal of Algorithms*, 16:1–23, 1994.

[36] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

[37] X. Wang, B. Chen, G. Qian, and F. Ye. On the optimization of fuzzy decision trees. *Fuzzy Sets and Systems*, 112:117–125, 2000.

[38] L. X. Wang and J. M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22:1414–1427, 1992.

Chapter 7

SYNTACTIC PATTERN RECOGNITION

A. K. Majumdar and A. K. Ray

Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur, INDIA
e-mail: {*akmj,akray*}*@ece.iitkgp.ernet.in*

Abstract

In syntactic pattern analysis, the design of a pattern classifier involves inference of a set of generative grammars for each class of pattern. The inferred string grammar defines a class of patterns represented by the strings in that language. In this article, we have presented a number of techniques for grammatical inference. A procedure involving an automated inference of nonrecursive pattern grammar using formal power series theoretic technique has been described.

In any practical system, the strings belonging to a particular pattern class are always liable to be corrupted with noise. Error-correcting inference schemes have been suggested where the imperfectly formed strings in a particular pattern class are accepted by the inferred automata. The importance of high-dimensional pattern grammars in describing complex patterns has been highlighted and some important grammars have been described. Stochastic grammars and fuzzy grammars as generators of imprecisely defined strings have also been adequately described.

7.1 Introduction

The problem of pattern recognition was first related to formal linguistic techniques by Narasimhan [40] when he suggested "··· recognition presupposes the capability to analyze and articulate the structural aspects of input pictures. It is only on the basis of such articulated description that it is possible to acquire general problem solving behavior concerned with pattern discrimination". There exists an inherent structure inside a pattern and along with this structure there also exist interrelationships among the primitive elements which form a pattern. The limitations of decision theoretic pattern classification techniques lie in their incapability to articulate these interrelationships among the pattern substructures. This has led to the era of structural or syntactic pattern recognition. The interrelationships among pattern elements, called primitives, and the articulated description of a pattern in terms of such relations provide the basis for a structural or linguistic approach to pattern recognition. In syntactic pattern recognition, each pattern is characterized by a string of primitives and the classification of a pattern in this approach is based on the analysis of the string with respect to the grammar defining that pattern class. In structural approach the problem of abstraction and generalization is essentially the problem of selection of appropriate primitives, characterizing a pattern using these primitives and inferring a generative grammar from a finite set of sample pattern strings. Fu [18], and Gonzalez and Thomason [25] provide excellent accounts and introduce fundamental concepts on structural aspects of pattern classification. The syntactic approach to pattern recognition involves a set of processes, namely,

- selection and extraction of a set of appropriate primitives
- analysis of pattern description by identification of the inter-relationship among the primitives
- recognition of the allowable structures defining the inter-relationship between the pattern primitives

A typical block diagram of a syntactic pattern recognition system is shown in Fig. 7.1.

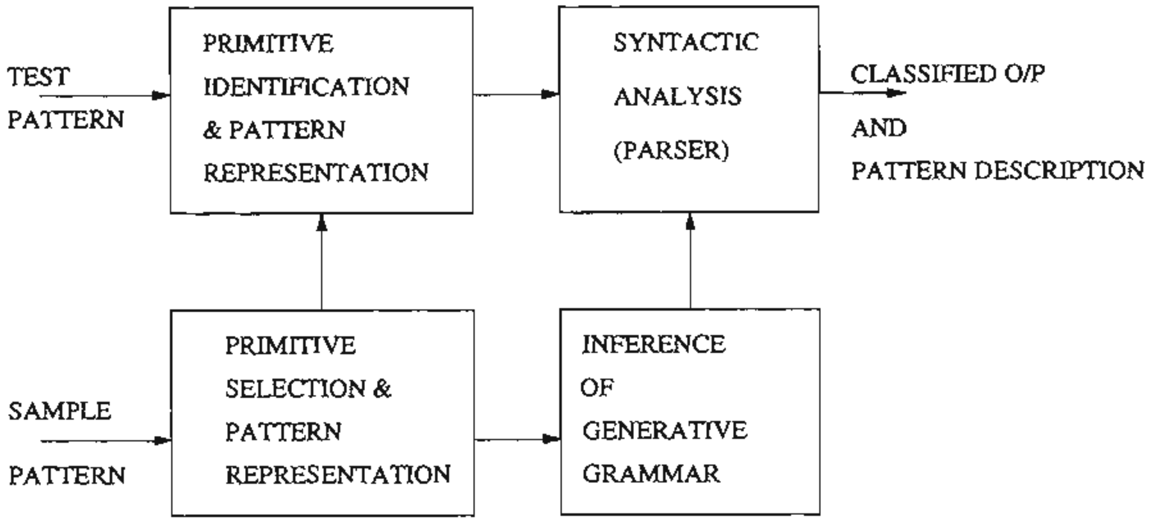

TEST PATTERN
PRIMITIVE IDENTIFICATION & PATTERN REPRESENTATION
SYNTACTIC ANALYSIS (PARSER)
CLASSIFIED O/P AND PATTERN DESCRIPTION
SAMPLE PATTERN
PRIMITIVE SELECTION & PATTERN REPRESENTATION
INFERENCE OF GENERATIVE GRAMMAR

Fig. 7.1 Syntactic pattern recognition

7.2 Primitive selection strategies

As mentioned earlier, segmentation of patterns poses the first major problem in syntactic pattern recognition. A pattern may be described by a string of sub-patterns or primitives, which may easily be identified. If each sub-pattern is complex in structure, each of these may again be described in terms of even simpler sub-patterns, which are again easily identifiable. The problem of primitive selection is a fundamental one and various approaches are available, which may be grouped as

(1) general methods based on boundary tracing
(2) general methods emphasizing segmentation of the pattern in several regions
(3) knowledge-based segmentation techniques

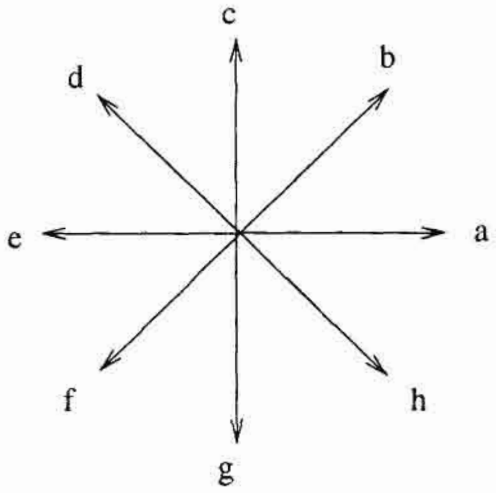


Fig. 7.2 Directional code

One of the most frequently used schemes of boundary descriptions is the chain code method of Freeman [17]. Under this approach, a rectangular grid is overlaid on a two-dimensional pattern and straight line segments are used to connect the adjacent grid points covering the pattern. Let us consider a

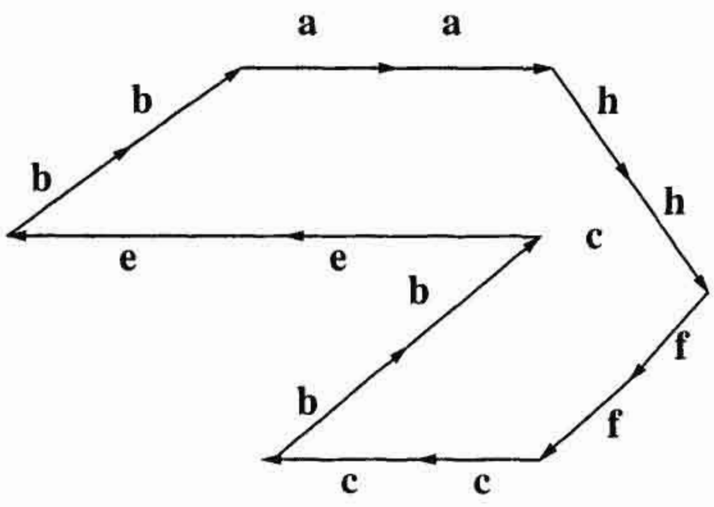


Fig. 7.3 Freeman code for a closed pattern

sequence of n points $\{p_i | i = 1, 2, \dots, n\}$ which describe a closed curve. Here the point p_i is a neighbor of $p_{i-1}(\text{mod } n)$. The Freeman chain code contains n vectors $p_i\vec{p_{i-1}}$ and each of these vectors is represented by an integer $m = 0, 1, \dots, 7$ as shown in Fig. 7.2. The angle subtended by a vector with the horizontal line is $\frac{\pi}{4}m$. Each line segment is assigned an octal digit according to its slope and the pattern is represented by a chain of octal digits. This type of representation yields patterns composed of a string of symbolic valued primitives. Fig. 7.3 shows the Freeman chain code of a closed pattern. Apart from its simplicity, Freeman's method has other advantages too. The coded patterns may be rotated through 45° simply by adding an octal digit to every digit in the chain. This method may be used for coding any arbitrary two-dimensional figure composed of straight lines or curved segments and has been widely used in character or line drawing recognition applications. The major limitation of this procedure is that the patterns need adequate preprocessing in order to ensure proper representation. Some alternative chain code representations have also been suggested which preserve the shape of the pattern, yet perform considerable data reduction. Some of these are

- property-encoding applications in boundary tracing of binary pictures [51]
- object recognition using a two-dimensional polar transformation [7]

- vertex chain codes [8]

The vertex chain code is invariant under translation and rotation and is usually invariant to mirror transform as well. Here the chain code is extracted for a pattern of any shape which is composed of a finite number of cells. These cells may be triangular, rectangular or hexagonal in shape.

Identification of structural interrelationships

Once a satisfactory solution to the primitive selection and extraction problem is available, the next step is the identification of structural relationships among the extracted pattern primitives. A pattern may be described as a string or a sentence of primitives. First-order logic may be used for describing the primitive interrelationships where a pattern is described by certain predicates, and objects occurring in the pattern may be defined using the same predicates. When the patterns are represented as strings of primitives they may be considered as sentences of regular, context-free or context-sensitive languages. Thus suitable grammars may be defined for generating pattern languages by specifying a set of production rules which generate the sentences in the said pattern language. The corresponding computing machines, known as automata, have the capability of recognizing whether a string of primitives belongs to a specific pattern class.

7.3 Formal linguistic model: basic definitions and concepts

In this section, we present the basic definitions and results from formal language and automata theory which are used in the study of grammatical inference. Most of the material in this section may be found in Gonzalez and Thomason [25], Fu and Booth [20], and Hopcroft and Ullman [26].

An alphabet V is a finite set of symbols called letters. A sentence X over an alphabet V is a string of finite length, formed with zero or more letters from V, where the same letters may occur a number of times. The length of X, denoted by $|X|$, is the number of symbols used to form X. The empty sentence consisting of no symbol is denoted by λ, such that $\lambda Z = Z\lambda = Z$. The set of all sentences over an alphabet V including λ, is denoted by V^*. The positive closure of V is denoted by V^+, such that $V^+ = V^* - \lambda$. A language is a finite or countably infinite set of sentences over an alphabet.

Definition 7.1 A phrase-structure grammar embodying a set of rules of syntax is defined formally as a 4-tuple

$$G = (V_N, V_T, P, S),$$

where V_N is a finite set of nonterminals, V_T is a finite set of terminals, $V_N \cup V_T = V$ and $V_N \cap V_T = \emptyset$. Here P is a set of production rules of the form $\alpha \rightarrow \beta$ where α is a sentence in V^+ containing at least one nonterminal, β is a sentence in V^* and $S \in V_N$ is the starting symbol. ■

Henceforth nonterminals will be denoted by capital letters (such as A, B, *etc.*); terminals by lower-case letters (such as a, b, c, *etc.*); strings of terminal symbols by x, y, z, *etc.*; and mixed strings of terminal and nonterminal by lower-case Greek letters like α, β. It may be observed here that a grammar can generate the sentences of a language which represents a pattern class. The set of patterns, represented as strings of primitive symbols and belonging to a pattern class, form a language.

Definition 7.2 The language generated by G, denoted by $L(G)$, may be defined as the set

$$L(G) = \{w \in V_T{}^* | S \underset{G}{\overset{+}{\Rightarrow}} w\},$$

■

where the notation "$S \underset{G}{\overset{+}{\Rightarrow}} w$" means that the string is derived from the starting symbol S by application of one or more production rules in P. $L(G)$ so formed is a subset of $V_T{}^*$ and the string w is formed by concatenating the terminal symbols in V_T in the grammar. The string w is called a sentence over the alphabet V. Starting with the non-terminal S, w is obtained by using a finite number of production rules in P. We will next define different types of phrase-structure grammars as follows.

Definition 7.3 Let G be a phrase-structure grammar with productions of the form $\alpha \rightarrow \beta$; then we may define the following types of grammars:

(1) If there is no additional restriction on the productions, the resulting grammar is an unrestricted (type 0) grammar.

(2) G is termed a context-sensitive grammar (type 1) if its productions are of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ where $\alpha_1 \alpha_2 \in V^*$, $\beta \in V^+$ and $A \in V_N$. Here the nonterminal A may be rewritten as β only when

A appears in the context of substrings α_1 and α_2. It may be noted here that $|\alpha| \leq |\beta|$, where $|\alpha|$ and $|\beta|$ are the lengths of α and β.

(3) If $a \in V_N$ and $\beta \in V^+$ then G is a context-free (type 2) grammar, provided each production rule is of the form $A \rightarrow \beta$. Here the nonterminal A may be replaced by a string β regardless of the context in which A appears as in the production $A \rightarrow \beta$.

(4) If the productions are of the form $A \rightarrow aB$ or $A \rightarrow a$, for $A, B \in V_N$ and $a \in V_T$, then G is a regular (type 3) grammar.

■

It is easy to note that the context-free grammar is a special case of context-sensitive grammar while the regular grammar is a special case of context-free grammar. Thus a language generated by type n grammar may as well be generated by a type $n-1$ grammar. The reverse, however, is not true. Thus a language is classified by the class of the most restricted grammar which generates it. The increased power of a context-free grammar over a regular grammar, as a generator of languages, is inherent in a property known as self-embedding. A context-free grammar supports a set of self-embedded nonterminal symbols. A nonterminal symbol A is self-embedded if the grammar supports a production rule of the form $A \rightarrow \alpha A \beta$, where $\alpha, \beta \in V^+$. Regular and context-free grammars find wide applications in syntactic pattern analysis and henceforth we will be interested only in these two classes of grammars and their corresponding automata. A grammar G partitions the set $V_T{}^*$ into two disjoint subsets $L(G)$ and $\overline{L(G)}$, where

$$\overline{L(G)} = V_T{}^* - L(G).$$

Here $\overline{L(G)}$ is known as the complementary language to $L(G)$. A very important lemma characterizing context-free and regular languages is the pumping lemma which states that for a sufficiently long string w in a context-free language L of the form

$$w = u_1 w_1 u_2 w_2 u_3 C_1 C_2 \neq \lambda,$$

the word $u_1 w_1 u_2 w_2^i u_3$ belongs to L for every $i \geq 0$. The corresponding results in the regular language may be stated as follows. For a sufficiently long string w with $w = u_1 w_1 u_2$, $w_1 \neq \lambda$, the string $u_1 w_1^i u_2$ also belongs to the regular language for every $i \geq 0$. Just as grammars were defined as generators of pattern languages, the corresponding computing machines known as automata can recognize if a string belongs to a specific language of pattern classes. We will restrict our discussion only to finite-state automaton and pushdown automaton as recognizers of regular and context-free languages.

Definition 7.4 A nondeterministic finite-state automaton is a computing system specified as a quintuple

$$(Q, V_T, \delta, q_0, F),$$

where Q is a finite set of states, V_T is a finite input alphabet, δ is a mapping $Q \times V_T \rightarrow 2^Q$, the collection of all subsets of Q. $q_0 \in Q$ is the starting state and $F \in Q$ is the set of final states. ■

To recognize a string $x \in V_T{}^*$, each symbol of the string is scanned from the left-most symbol to the right across the tape. The string is said to be recognized by the automaton, if starting from an initial state q_0, the machine follows a sequence of states to halt in one of the final states in F when all of x is scanned. The sequencing is completely guided by the δ-mapping. If there exists a state-input combination (q, a) for which $\delta(q, a) = \phi$, then the finite state automaton does not accept the string.

If, for each state-input combination, there exists a unique next state specified by the δ-mapping, the automaton operates deterministically since it is not forced to choose the next state from among several possibilities. Such automata are known as completely specified deterministic finite state automata. The interrelationship between a finite automata and a regular grammar may be stated as follows.

Lemma 7.1 *A language is said to be regular if and only if it is accepted by a finite state automaton.*

This means a finite state automaton accepts only those strings which are generated by a regular grammar. Just as finite state automata are recognizers of regular languages, computing devices which can recognize context-free languages are called pushdown automata.

Definition 7.5 A pushdown automaton (PDA) may formally be defined as a seven-tuple

$$a_p = (Q, V_T, \delta, q_0, z_0, F, Z),$$

where Q, V_T, q_0 and F are as defined earlier, Z is a finite pushdown list alphabet, δ is a mapping, such that for every current-state current-input symbol and top-of-the-stack symbol, the automaton assumes a specific next state and writes a string on the top of the stack, *i.e.*, $\delta : Q \times (V_T \cup \lambda) \times Z \rightarrow Q \times {V_T}^*$; Z_0 is the initial pushdown list symbol. ■

Here the automaton, starting in the state q_0, scans the string $x \in V^*{}_T$ with Z_0 the initial pushdown list symbol on top of the stack. A pushdown automaton thus utilizes a pushdown stack on which the automaton can write a string, including a null symbol at every state transition. A string $x \in {V_T}^*$ is said to be accepted by the pushdown automaton *if and only if* the automaton, after scanning the entire string, either reaches one of the final states in F or, alternatively, the automaton leaves the stack empty after the entire string is scanned.

7.4 High-dimensional pattern grammars

The string representation of patterns is quite adequate for structurally simpler forms of patterns. The classical string grammars are, however, weak in handling structurally complex pattern classes. This is because the only relationship supported by string grammars is the concatenation relationship between the pattern primitives. Here each primitive element is attached only with two other primitive elements – one to its right and the other to its left. Such a simple structure thus may not be sufficient for characterizing more complex patterns, which may require better connectivity relationship for their description. An appropriate extension of string grammars has been suggested in the form of high-dimensional grammars. These grammars are more powerful generators of languages and are capable of generating complex patterns like chromosome patterns, nuclear bubble chamber photographs.

A picture description grammar, proposed by Shaw [50], is a context-free generator of a picture description language (PDL), in which a primitive is an n-dimensional pattern with two distinct connecting points. Here the primitive elements may be of arbitrary shapes and they may be connected only at a specified point on the shape primitive. The PDL have been used to describe nuclear bubble chamber photography.

Ledley *et al.* [32] had described a syntactic approach for chromosome classification using a set of primitives which sufficiently describe telocentric and submedian chromosomes. They had derived a context-free grammar for generating the strings belonging to these two classes of chromosomes.

More general forms of grammars capable of generating more complicated pattern structures have also been proposed. These include tree, web, plex and shape grammars.

In tree grammars, a tree with a finite set of nodes corresponding to the pattern primitives is specified by a set of relations between each node and its neighbors, which describe the relationship of pattern primitives with other pattern substructures. Tree grammars have been applied to syntactic pattern recognition by researchers for pattern description and classification. Some applications of tree grammar include

(1) fingerprint classification by Moayer and Fu [38]
(2) analysis of bubble chamber photograph by Fu and Bhargava [19]

Relational graphs as direct generators of trees have also been used for describing patterns. An interesting class of tree grammars, known as tree adjoining grammars, was proposed by Joshi *et al.* [29]. Such grammars generate context-sensitive languages utilizing two composition operations, namely,

(1) adjoining operation
(2) substitution operation

The concept of tree adjoining grammar has been extended by Aizawa and Nakamura [1] to generate a set of quadtrees representing a class of digital images. Quadtrees have been used for representing complex binary patterns and such representations have been found to be effective for feature extraction and pattern characterization. The quadtree adjoining grammar yields a compact description of a set of patterns, provided the patterns may be represented by initial quadtrees and by using a few adjoining operations.

The syntactic technique has also been used for picture generation. This may be performed by growing the picture by adding modular primitive shape units in bits and pieces until the desired picture is generated completely. Random-context picture grammar has been utilized by Ewert and Van der Walt [12], who performed successive refinements on the pattern by using a set of rewriting rules in specific picture grammar. The introduction of geometric context in this grammar has enriched the generation of picture utilizing random-context picture grammar. An attributed grammar, consisting of a syntactic part with a relational attribute coupled with a set of semantic rules, has also been used for pictorial pattern recognition. Array grammars have been proposed as a generalization of string grammars, where the rewriting rules involve replacement of a 2-D subarray by another subarray. Array grammars and array automata have been described comprehensively by Milgram and Rosenfeld [37].

Webs, which are undirected labelled graphs, have been used for syntactic pattern description [44]. In a string grammar, each primitive symbol is concatenated with only two other primitive elements, one to the right and the other to the left of the element. A class of grammars was suggested by Feder [13], in which a set of primitive elements having multiple connectivity structure may be used. These grammars are known as plex grammars. Plex grammars use primitive structures called n-attaching point entity (NAPE). Each attaching point entity is nothing but a shape primitive having a set of attaching points at which another set of shape primitives may be joined. A set of identifiers associated with each NAPE has been used for pattern generation. The n-attaching point entities are primitive elements in which there are n specified points on the primitive elements at which other point elements may be connected. Thus this class of grammars possesses more generating capabilities compared to the string grammars. These grammars have been used for generating the complex structures of organic compounds like rubber molecules, where the chain of $C - H$ structure repeats to form complex organic compounds.

7.5 Structural recognition of imprecise patterns

In many practical applications, uncertainty exists in the process of classification, where some of the patterns may occur more frequently than others and certain variations or distortions in the pattern may be more likely than others. In such cases, probabilistic measures are employed in the process of classification. In such grammars, known as *stochastic grammars*, probabilities are assigned to the production in the generative grammar in such a way that the *a priori* likelihoods of the classes and the individual strings in a class are reflected in the productions.

When the imprecision in the pattern strings is more due to vagueness and inexact description rather than randomness, fuzzy grammars have been proposed for syntactic analysis. In fuzzy grammars, each production rule is associated with a fuzzy membership grade.

As in the case of the decision theoretic approach, the formal linguistic method can also be extended to fuzzy or stochastic syntactic models, by which fuzzy or probabilistic aspects may introduced into the linguistic model.

Thus in all such cases, it is important to construct a recognizer which can recognize imprecisely-defined strings of patterns. It may be noted that ordinary automata cannot recognize such strings and hence the need of recognizers of fuzzy or stochastic languages. Here we will primarily discuss the methodology for inferring a fuzzy grammar and a similar procedure may be used for stochastic grammar inference as well.

7.5.1 Stochastic grammars for pattern representation

In many practical situations, certain amount of uncertainty in the form of noise and variations in the pattern measurements exist. There exist non-stochastic approaches to error correcting grammars, which have been discussed in earlier sections, with the implicit assumption that all the patterns under consideration are equally likely to occur. Alternatively, there are situations, where certain patterns may occur more frequently than others and there exist probabilistic measures for the occurrence of each pattern. Stochastic grammars and automata have been used to tackle such problems where probabilistic measures are employed in the classification process. In stochastic grammars, probabilities are assigned to each of the productions in the generative grammars in such a way that the *a priori* likelihoods of the classes and also the individual sentences in each class are properly reflected in the probabilistic grammar. Fu [18], Fu and Huang [21], and Gonzalez and Thomason [25] provide background for stochastic grammars and automata. Once the stochastic syntactic description of the string corresponding to the input pattern is complete, it is assigned a probability with respect to each of the generative grammar classes. Various methods of estimation of production probabilities and the use of stochastic syntactic analysis for regular and context-free classes of generative grammars for pattern classification have been reported, for instance, by Gonzalez and Thomason [25], and Maryanski and Booth [36].

7.5.2 Fuzzy syntactic model

Since the introduction of fuzzy set theory by Zadeh [58], various models of fuzzy automata have been studied in the literature. Fuzzy models are applied in cases where the imprecision in the pattern description arises from an intrinsic ambiguity (imprecise definition), rather than an associated randomness in the environment. Once a set of generative grammars is inferred using a finitely large set of training patterns, the assessment of the overall efficiency of the inferred grammar or automata in correctly classifying the patterns not included in the training set, is done in the generalization phase.

7.5.3 Fuzzy automata and languages

The theory of fuzzy languages and their role in describing pattern classes have received importance during the last three decades. As an extension to the crisp language recognition theory, where a machine computes the characteristic functions of the language, the recognition of fuzzy languages may be viewed as a problem where the machine computes the fuzzy membership functions of the strings of the language. Each string in a fuzzy language has a membership grade associated with it, which denotes the class membership of the string in that language. The membership function of a string in a language may be computed by a set of state configurations, such that the state transitions and the final state configuration for a given input string is uniquely associated with the membership function of the string. Lee and Zadeh [33] have applied the fuzzy set theory to formal languages. Various models of fuzzy programs and fuzzy automata have been proposed by Santos [47]. These models include the max-min, max-product and probabilistic automata.

The properties of fuzzy automata have been investigated by Mizumoto *et al.* [39]. These concepts are based on the notion of fuzzy sets of type 2. Although ordinary fuzzy automata and finite automata are special cases of fuzzy-fuzzy automata, the power of fuzzy-fuzzy automata as an acceptor is the same as that of fuzzy automata.

The notion of a regular fuzzy expression and its use in recursive generation of the family of fuzzy languages have been shown by Santos [48], Kandel and Lee [30], Thomason and Marinos [55], among others. Wee and Fu [57] have proposed fuzzy automata as models of a learning system. The learning behavior of the unsupervised system is reflected by the presence of a non-stationary fuzzy transition matrix.

Some of the major applications of fuzzy syntactic analysis are handwritten English script recognition by DePalma and Yau [10], recognition of handwritten characters by Kickert and Koppelaar [31], and Stallings [53], and identification of skeletal maturity from X-rays by Pathak and Pal [42].

It may be mentioned that although various types of fuzzy grammars and fuzzy automata have been developed by Santos [47], Kandel and Lee [30], Thomason and Marinos [55] and others, the problem of fuzzy grammatical inference needs more investigation.

The definitions of formal language theory may be generalized for fuzzy grammars by associating a membership grade to each of its productions. As in the case of crisp grammars, regular fuzzy grammars generate regular fuzzy languages. We now present the preliminary notions related to formal fuzzy language and fuzzy automata briefly.

Definition 7.6 A fuzzy language L over an alphabet V_T is defined to be a fuzzy subset of $V_T{}^*$, such that a sentence $x \in V_T{}^*$ has a membership grade $\mu_L(x)$, which denotes the degree of membership of L. Here $0 < \mu_L(x) \leq 1$. ■

Definition 7.7 A fuzzy regular grammar may be formally defined as a four-tuple

$$\langle V_T, V_N, S_f, P \rangle$$

where the starting symbol S_f is a fuzzy subset of V_N, the set of nonterminals; V_T is a set of terminals and P is a finite set of production rules of the form

$$A \xrightarrow{\theta} aB$$

or

$$A \xrightarrow{\theta} a,$$

where $A, B \in V_N$, $a \in V_T$ and $0 \leq \theta \leq 1$ determines the grade of membership corresponding to the production rule. ■

Definition 7.8 A fuzzy regular language is a fuzzy subset of $V_T{}^*$ such that the membership grade of a string $x \in V_T{}^*$ in the fuzzy language can be determined by applying the production rules of a fuzzy regular grammar, as defined above. ■

The membership grade, $\mu_L(x)$, in a fuzzy regular language L, of a string $x \in V_T{}^*$ may be computed as follows:

Let $x = x_1 x_2 \ldots x_m$ and let the starting symbol $S_j \in V_N$ have a membership grade $\mu_s(S_j)$. The string x may be derived as follows:

$$S_j \overset{\theta_{i1}}{\Rightarrow}_{FRG} \alpha_1 \rightarrow \ldots \overset{\theta_{ik}}{\Rightarrow} x_1 \ldots x_m,$$

where $\alpha_i 1(V_T \cup V_N)^*$. The membership of the string x in the fuzzy regular language L then becomes

$$\mu_L(x) = \max_{i,j} [\min(\theta_{i_1} \dots \theta_{i_k})],$$

where the max operator is applied over all possible derivations of x, utilizing the fuzzy regular grammar production rules. If no such derivation exits then $\mu_L(x) = 0$.

Example 7.1 Let

$$\begin{aligned}
V_T &= (a, b, c),\\
V_N &= (S_1, S_2, S_3, S_4, S_5, S_6),\\
S &= \{(0.9, S_1), (0.2, S_2)\}\\
P \ &: \ S_1 \xrightarrow{0.8} aS_6\\
&\quad\ S_2 \xrightarrow{1} bS_3\\
&\quad\ S_4 \xrightarrow{0.7} aS_5\\
&\quad\ S_1 \xrightarrow{0.3} aS_2\\
&\quad\ S_3 \xrightarrow{1} c\\
&\quad\ S_5 \xrightarrow{1} bS_6\\
&\quad\ S_1 \xrightarrow{1.0} aS_4\\
&\quad\ S_4 \xrightarrow{1} bS_2\\
&\quad\ S_6 \xrightarrow{1} b
\end{aligned}$$

Then, for the string $abbc \in V_T^*$, we have $\mu_L(abbc) = 0.9$. This may be observed from the derivation of $abbc$ from the starting symbol S_1 with membership grade $\mu(S_1) = 0.9$. A possible derivation is

$$S_1 \xrightarrow{1} aS_4 \xrightarrow{1} abS_2 \xrightarrow{1} abbS_3 \xrightarrow{1} abbc.$$

□

As in the case of a crisp language and automaton, the acceptor of fuzzy regular language is a fuzzy finite automaton (FFA), which is formally defined below.

Definition 7.9 A fuzzy finite automaton (FFA) M is defined as

$$M \equiv (Q, V_T, \mathbf{\Pi}, \boldsymbol{F}, \boldsymbol{\eta}),$$

where
$Q = \{q_1, \ldots, q_n\}$ is a nonempty finite set of internal states,
V_T is the input alphabet,
$\mathbf{\Pi}$ is an n-dimensional fuzzy row vector known as initial state designator, *i.e.*,

$$\mathbf{\Pi} = (\Pi_{q_1}, \ldots, \Pi_{q_n})',$$

and $0 \leq \Pi_{qi} \leq 1$,
$\boldsymbol{\eta} = (\eta_{q_1}, \ldots, \eta q_n)$ is a n-dimensional column vector called the final-state designator, $0 \leq \eta_{q1} \leq 1$,
$\boldsymbol{F}(x)$, for $x \in V_T$, is a fuzzy transition matrix such that for $q_i, q_j \in Q$, the (i,j)th element of $\boldsymbol{F}(x)$ is equal to $f_M(q_i, x, q_j)$, where the function

$$f_M : Q \times V_T \times Q \rightarrow [0, 1]$$

is the fuzzy state transition function. ∎

For $q_i, q_j \in Q$ and $x \in V_T$, $f_M(q_i, x, q_j)$ is the grade of transition from state q_i to state q_j when the input is x. For an input string $x = x_1x_2 \ldots x_m$, the grade of transition from $\{q_{i_1}, \ldots, q_{i_m}\}$, where $q_{i_j} \in Q$, can be defined by an m-ary fuzzy relation. For the computation of f_M over $V_T{}^*$ we may use

$$f_M(q_{i_1}, x, q_{i_m}) = \max\min\{f_M(q_{i_1}, x_1, q_{i_n}), \ldots, f_M(q_{i_{r_{m-2}}}, x_m, q_{i_{r_m}})\},$$

where, $q_{i1}, \ldots, q_{i_{r_{m-2}}} \in Q$.

The membership grade of the string x, accepted by the fuzzy automaton M, is

$$\mu_M(x) = \max \min_{q_{i_1}, q_{i_m} \in Q} \left[\Pi_{q_{i_1}} f_M(q_{i_1}, x, q_{i_m}) \eta_{q_{i_m}}\right].$$

It is obvious that a fuzzy language contains a large number of sentences with membership function values ranging from 0 to 1. In the syntactic description of patterns, we would be only interested in those strings whose membership grades are more than a certain threshold, say λ. The set of strings accepted by M with threshold λ is defined as

$$L(M, \lambda) = \{x | \mu_M(x) > \lambda, \, x \in {V_T}^*\}$$

As in the case of crisp regular grammars and automata, for every fuzzy regular language L there exists a fuzzy finite automata M such that for each $x \in {V_T}^*$, $\mu_L(x) = \mu_M(x)$, and *vice versa*.

Some interesting properties of fuzzy regular languages are briefly stated below.

(1) The union of two fuzzy regular languages L_1 and L_2 is a fuzzy regular language with

$$\mu_L(x) = \max\left[\mu_{L_1}(x), \mu_{L_2}(x)\right]$$

for all $x \in {V_T}^*$.

(2) The intersection of two fuzzy regular languages L_1 and L_2 is a fuzzy regular language with

$$\mu_L(x) = \min\left[\mu_{L_1}(x), \mu_{L_2}(x)\right]$$

for all $x \in {V_T}^*$.

(3) $L = L_1 \circ L_2$(concatenation) is a fuzzy regular language with

$$\mu_L(xy) = \min\left[\mu_{L_1}(x), \mu_{L_2}(y)\right]$$

for all $x, y \in {V_T}^*$.

7.5.4 Fuzzy context-free languages

A fuzzy context-free grammar G has productions

$$A \xrightarrow{\theta} \alpha$$

where $\alpha \in (V_T \bigcup V_N)^*$, $0 \leq \theta \leq 1$. As in the case of a crisp context-free grammar, the strings generated by a fuzzy context-free grammar are accepted by a fuzzy pushdown automaton (FPDA), which is defined as follows.

Definition 7.10 A fuzzy pushdown automaton (FPDA) is defined as

$$M = (Q, V_T, \mathbf{\Gamma}, \mathbf{\Pi}, f, \boldsymbol{\eta})$$

where $Q = \{q_1, \ldots, q_n\}$ is a nonempty finite set of internal states, V_T is the input alphabet, $\mathbf{\Gamma}$ is a list of stack symbols, $\mathbf{\Pi}$ is the initial state designator, η is the final state designator, and f is the fuzzy transition map

$$f(q, a, z, \hat{q}, z_1, \ldots, z_r) = \theta,$$

where $0 \leq \theta \leq 1$. The transition mapping identifies the possibility of transition from state q, with input a and top-of-stack symbol z to a new state $\hat{q}$ and in the process the automaton writes $z_1, \ldots, z_r$ on to the stack, with z_1 at the top. ∎

For every fuzzy context-free language L, there exists a fuzzy pushdown automaton M such that $\mu_L(x) = \mu_M(x)$, $\forall x \in V_T{}^*$. Thus a fuzzy pushdown automaton acts as a parser for the fuzzy context-free grammar.

7.6 Grammatical inference

In syntactic pattern recognition, the problem of grammatical inference is one of central importance. This approach is based on the underlying assumption of the existence of at least one grammar characterizing each pattern class. The identification and extraction of the grammar characterizing each pattern class form the core problem in the design of a syntactic pattern classifier.

The problem of grammatical inference involves the development of algorithms to derive grammars using a set of sample patterns which are representatives of a pattern class under study. This may thus be viewed as a learning procedure using a finitely large and growing set of training patterns. In syntactic pattern classification, the strings belonging to a particular pattern class may be considered to form sentences belonging to the language corresponding to the pattern class. A machine is said to recognize a pattern class if for every string belonging to that pattern class, the machine decides that it is a member of the language and for any string not in the pattern class, it either rejects or loops for ever. For details on the origin of grammatical inference and subsequent developments, one may refer to the works of Chomsky [9], Chomsky and Miller [9], Solomonoff [52], Feldman [14], Feldman *et al.* [15], Gold [22], Horning [27], Pao [43], Fu and Booth [20], Biermann and Feldman [3], Gonzalez and Thomason [23, 25], Gonzalez *et al.* [24], Fu [21], Huang and Fu [28], Richetin and Vernadat [45], Tsai and Fu [56], Lu and Fu [35], and so on.

Inferring an appropriate grammar or the corresponding automaton using a set of samples belonging to different pattern classes, is a vital and indispensable aspect of syntactic pattern recognition. Out of these pattern strings there are strings which surely belong to a certain language characterizing a pattern class. Such a set of strings R^+ is termed a positive sample of a language $L(G)$. A complementary set of strings R^-, which surely do not belong to $L(G)$, is a negative sample of a language $L(G)$, *i.e.*, $R^- \subseteq \overline{L(G)}$.

Definition 7.11 A sample of a language $L(G)$ is a subset of $R = R^+ \cup R^-$. ■

Normally in inference problems, we will have situations where $R^+ \cup R^-$ has finitely many elements. However, if the positive sample set is allowed to grow such that $R^+ = L(G)$ then R^+ is said to be complete. Similarly if we have a large set of negative samples, *i.e.*, if R^- is allowed to grow such that $R^- = \overline{L(G)}$, then R^- is called complete. The sample set R is complete if $R = (L(G), \overline{L(G)})$. While inferring a grammar it is essential that each production rule of the grammar must be utilized in generating at least one string in the language. A positive sample R^+ of $L(G)$ is structurally complete if each production rule in G is used to generate at least one string in R^+.

For generating a positive sample set $R^+ = \{x_1, \ldots, x_n\}$ we may have a canonical regular grammar, $G_c = (V_{NC}, V_T, P, S)$ with V_T consisting of all the terminal symbols of R^+; P is a set of rewriting rules of the form $A_{i1} \to aA_{i2}, A_{i2}$ and $A_{i-1} \to a_{in}$, where each A_{in} represents a nonterminal $A_{in} \in V{*}_{CN}$, each a_{in} is a terminal *i.e.*, $a_{in} \in V_T$, and $S \in V_{NC}$ is the starting symbol. The set of non-terminals V_{NC} of the canonical grammar will be quite large and may be partitioned into a set of blocks to form a non-terminal set V_{ND} of a derived grammar
$G_D = (V_{ND}, V_T, P_D, B)$ where B is the starting symbol corresponding to the block containing S in G_c and P_D is a set of rewriting rules of G_D. The set of rewriting rules P_D may be given as follows:

(1) P_D contains a rewriting rule of the form $A_i \to aA_j$ if and only if there exist $Z_r, Z_s \in V_{NC}$ such that $Z_r \to az_s,\ Z_r \in A_j$;
(2) P_D contains a rewriting rule of the form $A_i \to a$ if and only if there exists $Z_r \in V_{NC}$ such that $Z_r \to a,\ Z_r \in A_i$.

Definition 7.12 Let $R^+ \in V_T{}^*$ be a positive sample set and let $Z \in V_T{}^*$ be a string such that $Zw \in R^+$ for $w \in V_T{}^*$. For any positive integer k, the k-tail of the string Z with respect to R^+ is defined as the set

$$H(Z, R^+, k),$$

where

$$H(Z, R^+, k) = \{w | Zw \in R^+, |w| \leq 1\}.$$

■

Definition 7.13 The formal derivative of a set of strings A with respect to the symbol $a \in V_T$ is defined as

$$D_a A = \{x | ax \in A\}.$$

■

The formal derivative of any string with respect to λ results in the same string.

Definition 7.14 The canonical derivative finite-state grammar G_{CD} associated with the positive sample set $R^+ = \{X_1, \ldots, X_q\}$ is defined as

$$G_{CD} = \{V_N, V_T, P, S\},$$

such that the following conditions hold:

(1) $v = \{U_1, \ldots, U_r\}$ be the distinct derivatives of R^+ not equal to λ and $U_1 = D_\lambda R^+$;

(2) $U_1 = S$;

(3) V_T is formed by the set of all distinct symbols in R^+, and V_N is identical to μ;

(4) P is as follows:
$u_i \to aU_j$ if and only if $D_a U_i = U_j$ for $U_i, U_j \in V_N$ and $a \in V_T$, and
$V_i \to a$ if and only if $\lambda \in D_a U_i$ for $U_i \in V_N$ and $a \in V_T$.

■

Having defined the basic elements of formal language theory, we next extend these concepts to the problem of inferring a grammar which generates (or, alternately, constructing an automaton which will accept) the patterns belonging to a certain class.

In the light of the discussion, and the definitions and results presented in the previous paragraphs, we will now formulate the process of grammatical inference. Given any string x from a sample of the language $L(G)$ generated by a grammar G, there exists a finite number of ways by which x can be generated by G. Moreover, a finite sample set may be associated with an infinite number of languages. Thus it is not possible to uniquely derive a grammar that generates a given sample. A grammar may be inferred which describes the strings in S^+ along with a number of strings similar to those contained in S^+. Since a finitely large number of grammars may generate the same positive sample, a number of admissible grammars are first enumerated in the process of inference.

Definition 7.15 A class of grammars $\mathcal{G}$ is said to be admissible if $\mathcal{G}$ is denumerable and if for any $w \in V_T{}^*$, it is decidable whether $w \in L(G)$ for any $G \in \mathcal{G}$. ∎

It may be observed that any finite set of strings can be described by at least one finite-state grammar (Fu and Booth [20]). The problem of inference is thus to identify at least one grammar out of the admissible set of grammars that satisfies the criterion that $L(G) \geq S^+$ and $S \leq L^-(G)$.

While inferring a finite-state grammar using a finite sample, it is assumed that the positive sample set is structurally complete. Different techniques have been proposed for selecting one or more grammars from the admissible class of grammars. Fu and Booth [20], Gonzalez and Thomason [25], Biermann and Feldman [4], and Horning [27] provide excellent accounts of the various facets of grammatical inference. We now briefly describe different techniques and approaches adopted for inference of regular and context free grammar. Several heuristic approaches to the grammatical inference problem have been proposed which are not presented as solutions to well-posed problems.

7.6.1 Supervised inference strategies

In this section, we present some standard techniques for grammatical inference that are based on supervised construction of grammars or automata using a set of positive sample strings.

The technique proposed by Chomsky and Miller [9] involves a four-step procedure where a regular grammar is inferred from a positive sample string with the help of a teacher. In this technique, all the cycles and sub-cycles of

R^+ are detected with the aid of a teacher and two sets of strings ${R_0}^+$ and ${R_c}^+$ are formed, where the first set ${R_0}^+$ contains the strings of R^+ in which no cycles are formed, while the second set ${R_c}^+$ contains the set of strings having one or more cycles, *i.e.*, ${R_0}^+ \cup {R_c}^+ = R^+$.

Let us now discuss the procedure for identifying a cycle in a valid sample set. Given a string x in a language L, we first identify all the substrings in x. For example, if $x = abca$ is a valid string in L, its substrings are $\{a, b, c, a, ab, bc, ca, abc, bca\}$. Each of the above substrings is deleted, one at a time, from the string $abca$, and the supervisor confirms whether the resultant string after deletion, is a valid string in the language or not. Thus the resultant strings after deletion of the substrings from x are

$$\{bca, aca, aba, abc, ca, aa, ab, a, a\}.$$

The next step is the confirmation by the supervisor, which of the resultant strings are valid strings in the language. Suppose bca is a valid string in L. Then the supervisor next checks if the strings $\{aabca, aaabca, aaaabca, \ldots, a^m bca\}$ are all valid strings in L. If all of them are in $L(G)$, there exists a cycle in a and we may denote this cycle by $(a)bca$. There may exist cycles of any length in a sample set. In case we have cycles of length greater than one, there is a possibility that there may exist subcycles in the same strings. Thus, suppose there is a cycle in bc. Then one should check if there is any subcycle in b or c. Once the cycles and subcycles are detected in the strings of the positive sample set, and the two sets of strings ${R_0}^0$ and ${R_c}^+$ are formed, a set of productions is next formed using the string in R_c^+. The resulting set of productions is augmented so that the augmented set of production rules can generate the strings in ${R_0}^+$. The regular grammar thus inferred is minimized using standard techniques.

This method was generalized by Solomonoff [52] for context-free grammars. The method depends on the determination of all the cycles in a language. Solomonoff's assertion was that there always exists at least one finite set of strings with their cycle markers such that this finite set can generate the entire language. The basic cycle forms can be converted to the conventional form of a context-free grammar.

One of the problems with these methods is that a large and highly redundant grammar would result from the inference procedures. The other limitation of these methods is that the supervisor needs to check a large sample set, for the detection of cycles in the sample set. The computational complexity of these methods thus becomes prohibitively large.

7.6.2 Unsupervised inference strategies

An unsupervised strategy for inferencing has been proposed by Feldman [14] which is based on the detection of iterative regularity. In this method, the resulting grammar is inferred without the aid of any supervisor and the method employs only a positive sample set. The strategy employed here involves, in the first step, generation of a canonical grammar that generates only the given sample set. The sample strings are taken one after another in order of decreasing length, with the longest string first in the queue. Given a structurally complete set S, an admissible class of ordered finite-state grammars $\mathcal{G}_F$ can be defined such that

$$\mathcal{G}_F = \{G_1, \ldots, G_c\},$$

where G_j is the jth derived grammar obtained from the canonical grammar G_c. The number of non-terminals of G_i is always less than or equal to that of G_j for $i < j$. It is important here to note that any such procedure would be practically viable only if the number of such admissible grammars can be reduced. Inference methods based on formal derivatives and k-tail methods have been proposed which reduce the number of admissible grammars. The computational complexity in the case of inference procedures based on formal derivatives becomes large for strings of large length and thus they are not considered to be very useful. However, the k-tail method has been observed to perform better. This method, suggested by Bierman and Feldman [5], infers a finite state automaton directly from the sample set R^+. This procedure also identifies the iterative regularity in the strings of R^+. In the proposed algorithm, k-tail of string $z \in V_T{}^*$, is defined as a set $h(z, R^+, k)$ where $h(z, R^+, k) = \{w | zw \in R^+, |w| \leq k\}$, k being a positive integer. The length of the string $w \in V_T{}^*$ is less than or equal to k, which is an adjustable parameter. The set of states of the inferred automaton is obtained as sets of symbols which are the k-tails of sub-strings $w \in V_T{}^*$. For each $a \in V_T{}^*$, the state transition function δ is given as,

$$\delta(q, a) = \{q\prime \in Q | q\prime = h(z, a, R^+, k)\},$$

where $q = h(z, R^+, k)$ and the initial state is $q_o = h(\lambda, R^+, k)$. The set of final states contains those sets having the empty string λ in their k-tail representation and is given as

$$F = \{q | h(z, R^+, k) = \lambda\}.$$

The inferred non-deterministic automaton is next converted to a deterministic minimal automaton by standard techniques. The major problem in k-tail methods is that the nature of the string accepted by the automaton depends on the value of k, and the size of the language accepted by the automaton decreases with an increase in k. If m is the length of the largest string in R^+, then the automaton with $k \geq m$ will accept only those strings in R^+, while if $k = 0$, the automaton accepts $V_T{}^*$ including the string λ. For values of $k < m$, the automaton may accept string from R^- as well.

7.6.3 Enumerative inference strategies

The enumerative approach to grammatical inference involves enumeration of all the possible candidate grammars and subsequently the appropriate grammar is searched. This makes use of a 'goodness measure', by optimizing which the best grammars may be inferred. Gold [22] proposed the idea of *identification by enumeration*. Horning [27] had reported the computer implementation of the enumeration method based on a Bayesian approach. He used the concept of grammatical covering to eliminate a large class of grammars. The class of grammars is visualized as being organized in a tree structure where a grammar G with n nonterminals represents a node on the tree and each grammar G' with $n+1$ nonterminals which is covered by G is associated with a node branching downwards. A node, with all its children, is eliminated if the grammar corresponding to it fails to generate the given sample set. Horning's method yields an effective grammar that converges and tolerates noise of known distribution. The procedure is computationally expensive since the procedure involves enumeration and rejection of a large number of grammars which are not deductively acceptable. Biermann and Feldman [4] had pointed out that for m terminals and n non-terminals there are approximately $2^{nm(n+1)}$ possible grammars. However, pruning techniques may be used for selecting the set of admissible grammars. Pao [43] utilized pruning techniques based on the notion of grammatical coverage for inference using enumeration. First, a finite-state machine M_0, that can accept only the strings that belong to R^+, is constructed. An ordered lattice of automata is next formed by merging states of M_0 such that M_0 is the lowest point in the lattice whereas the highest point is the universal acceptor. The nodes at higher levels are obtained by merging the nodes in the adjacent lower level. It has been shown that M_0 essentially represents the canonical grammar whereas other nodes in the lattice are associated with derived grammars. In this method, a systematic approach of organizing the admissible grammars has been suggested.

7.6.4 An inference strategy based on formal power series

We describe a formal power series theoretic technique for inferring a grammar. The algorithm is suitable for inferring fuzzy as well as crisp grammars. With appropriate extension stochastic grammars may as well be inferred using this strategy. In syntactic pattern classification, a set of predicates describes the structural relationship among primitives defined in their arguments. These propositions, which are binary-valued in predicate calculus, find an extension in the fuzzy domain with a multiplicity of values in the interval $[0, 1]$ for fuzzy patterns. The membership grades of the strings of pattern primitives are evaluated based upon the relative frequency of occurrence of the string sequence in the sample.

As has been already mentioned, the inference of a regular grammar is based on the assumption that each string in the sample set has a grade of fuzzy membership in the corresponding language, that is to be estimated *a priori*. This membership may be $\{0, 1\}$ for crisp patterns and $[0, 1]$ for fuzzy patterns.

The inferred finite-state automaton assigns a membership grade to each of its productions and estimates the membership grades of the initial state and the set of final states of the automaton. The problem of construction of minimal automata, involving the development of algorithms capable of deriving automata from a set of sample sentences, has received paramount attention in recent times. Many investigations [2, 4, 6, 14, 20, 30, 38, 57] have been reported in literature on the automated inference of regular and context-sensitive grammars in deterministic and non-deterministic domains. Stochastic language and automata have also been proposed for systems operated in non-ideal environments [16, 19].

The theory of formal power series was first related to formal language theory by Schutzenberger [49], and the concept was later extended by Biswas [6], Eilenberg [11], Fliess [16], and Salomaa and Soittola [46].

The formal power-series representation yields a minimal automaton. In the next section some of the preliminary concepts of formal power series are introduced. These results are used in later sections to derive a minimal automaton from the given sample strings. Finally, an application of the proposed interface technique to character recognition is shown.

7.6.4.1 *Preliminaries of formal power series*

In this section, some of the definitions and results of the theory of formal power series are introduced. Most of the definitions and results in this section can be found in [6, 11, 16, 46, 49].

Definition 7.16 A monoid $M = \langle ; * \rangle$ is an algebraic structure consisting of a set m and the binary operation $*$ such that the following conditions are satisfied:

(1) For all $a, b, c \in m$, $a * (b * c) = (a * b) * c$, *i.e.*, $*$ is an associative binary operation
(2) There exists an $e \in m$ such that, for all $a \in m$, $e * a = a * e = a$.

■

A monoid is commutative if it also satisfies the law

(3) For all $a, b \in m$, $a * b = b * a$.

Here we will consider the free monoid $V_T{}^*$ generated by the words over an alphabet V_T, including the empty word.

A set of $m \times m$ square matrices, where the (i, j)th element a_{ij} of each matrix belongs to the semiring A, constitutes a semiring denoted by $A^{m \times m}$, where the usual rules of addition and multiplication of square matrices can be extended in an obvious manner.

It may be noted that modules are special cases of semimodules.

Definition 7.17 Let $D = \langle C, +, \rangle$ be an A-semimodule and be a non-empty subset of C. Then $U = \langle \acute{O}, + \rangle$ is a subsemimodule of D if for all $d_1, d_2 \in \acute{O}$ and $a, b \in \acute{O}$, $ad_1 + bd_2 \in \acute{O}$. ■

The subsemimodule generated by $\acute{O}$ is the smallest of all subsemimodules of D containing $\acute{O}$.

Hereafter, we will use the same notation for the algebraic structure and the corresponding underlying set. For example, the same letter A will represent the semiring A and the underlying set.

Definition 7.18 Let M be a monoid and A be a semiring. The mapping r of M into A is called a formal power series and r is written as a formal sum

$$r = \sum_{w \in M} (r, w)w \tag{7.1}$$

■

The values of $(r, w) \in A$ are also referred to as the coefficients of the series. We will consider here the free monoid $V_T{}^*$ generated by words over an alphabet V_T and r will be a series with non-commuting variables in V_T.

The collection of all formal power series r, as defined above, is denoted by $A\langle\langle M\rangle\rangle$. Given any language $L \subseteq V_T{}^*$, the language may be uniquely associated with a formal power series r belonging to $A\langle\langle V_T{}^*\rangle\rangle$.

The elements of the support of r, $r \in A\langle\langle V_T{}^*\rangle\rangle$, are the words $w \in V_T{}^*$ such that $(r, w) \neq 0$, and hence $supp(r)$ may be considered as a language over the alphabet $V_T{}^*$.

For the purpose of inference of regular grammars, we would require a recognizable series. A series r of $A\langle\langle M\rangle\rangle$ is termed A-recognizable, *i.e.*, $r \in A^{rcc}\langle\langle M\rangle\rangle$, if and only if

$$r = (r, \lambda)\lambda + \sum_{w \notin \lambda} p(\mu, w)w,$$

where $\mu : M \to A^{m \times m}$, $m \neq 1$, is a representation. Next we state the Schutzenberger theorem, which provides a convenient characterization of recognizable power series.

Theorem 7.1 *If $r \in A^{rcc}\langle\langle M\rangle\rangle$, then there exist a row vector α, a representation μ, and a column vector β such that*

$$r = \sum_{w \in M} (\alpha(\mu w)\beta)w. \tag{7.2}$$

Conversely, any series $\sum_{w \in M}(\alpha(\mu w)\beta)w$ belongs to $A^{rcc}\langle\langle M\rangle\rangle$.

Theorem 7.2 *(Kleene-Schutzenberger theorem) For a free monoid $V_T{}^*$, the families of $A^{rcc}\langle\langle V^*{}_T\rangle\rangle$ and $A^{rat}\langle\langle V^*{}_T\rangle\rangle$ coincide.*

The rational power series as described, however, can be characterized by its Hankel matrix, defined below.

Definition 7.19 The Hankel matrix of r (a series of $A\langle\langle V_T{}^*\rangle\rangle$) is a doubly infinite matrix $\boldsymbol{H}(r)$, whose rows and columns are indexed by the words of

$V_T{}^*$ and whose elements with indices u (row index) and v (column index) are equal to (r, uv). ∎

We next observe that if we appropriately define addition of functions in $A^{V_T{}^*}$ and multiplication of functions in $A^{V_T{}^*}$ by an element $a \in A$, then $A^{V_T{}^*}$ becomes an A-semimodule. We define addition of f_1 and f_2 for any $f_1, f_2 \in A^{V_T{}^*}$ as

$$(f_1 + f_2)u = f_1(u) + f_2(u),\ \forall u \in V_T{}^* \tag{7.3}$$

It may be noted that $f_i(u) \in A$ for $i = 1, 2, \ldots, n$. Hence $f_1(u) + f_2(u)$ corresponds to addition of elements in A. Then $(f_1 + f_2)A^{V_T{}^*}$ is a commutative monoid with respect to the addition of functions as defined here. Similarly, af, for $a \in A$ and $f \in A\langle\langle V_T{}^*\rangle\rangle$ can be defined as

$$(af)(u) = a \cdot f(u)\ \forall u \in V_T{}^* \tag{7.4}$$

Hence $A^{V_T{}^*}$ is an A-semimodule.

We next introduce a new operation where, for any $w \in V_T{}^*$ and $F \in A^{V_T{}^*}$, we define the function

$$wF(v) = F(vw),\ \forall v \in V_T{}^* \text{ and } wF \in A^{V_T{}^*} \tag{7.5}$$

It can be shown that the operator transforming F into wF is linear. If we consider the function F_v corresponding to the vth column of $\boldsymbol{H}(r)$, then from (7.3) and (7.5) we have

$$(wF_v)(u) = F_v(uw) = (r, uwv)\ \forall u \in V_T{}^* \tag{7.6}$$

This results in

$$(wF_v)(u) = F_{wv}(u)\ \forall u \in V_T{}^* \tag{7.7}$$

Thus the operation of premultiplication of F_v by w results in a new function F_{wv} that corresponds to the wvth column of $\boldsymbol{H}(r)$.

In the case of inference of crisp grammars, we will be concerned only with semiring A. Our aim is to construct the minimal automaton, crisp or fuzzy, that will accept sentences in R^+ of a crisp or fuzzy language as discussed in the next section.

7.6.4.2 *Construction of a minimal automaton*

To construct a fuzzy automaton from a set of sentences belonging to a positive sample set of a fuzzy language, the Hankel matrix is formed using all possible factorizations of each of the strings w_i. As observed in the previous section, the Hankel matrix is formed by the words of $V_T{}^*$, with each element equal to (r, uv), where u and $v \in V_T{}^*$ correspond to the row and column indices of $\boldsymbol{H}(r)$. The closed interval $[0,1]$ forms a commutative semiring with respect to max and min operations.

(1) The interval $[0,1]$ is a commutative monoid with the identity element 0 with respect to max operation $(*)$. This can be proved from the following equations:

   (a) $a * (b * c) = (a * b) * c \; \forall a, b, c \in [0,1]$,
   (b) $(a * 0) = (0 * a) = a$,
   (c) $(a * b) = (b * a)$.

(2) The interval $[0,1]$ is a commutative monoid with identity 1 with respect to min $(\cdot)$ as the binary operation, as shown below.

   (a) $a \cdot (b \cdot c) = (a \cdot b) \cdot c \; \forall a, b, c \in [0,1]$,
   (b) $(a \cdot 1) = (1 \cdot a) = a$,
   (c) $(a \cdot b) = (b \cdot a)$.

(3) The max-min operation is distributive, which can be verified from the conditions

$$a \cdot (b * c) = (a \cdot b) * (a \cdot c),$$
$$a * (b \cdot c) = (a * b) \cdot (a * c).$$

These equations can be verified by considering the following set of ordered inequalities:

$$a \geq b \geq c; \;\; a \geq c \geq b,$$
$$b \geq a \geq c; \;\; b \geq c \geq a,$$
$$c \geq a \geq b; \;\; c \geq b \geq a.$$

(4) $(a \cdot 0) = (0 \cdot a) = 0$.

The interval $[0,1]$ thus forms a semiring with respect to max and min operations and will henceforth be called a fuzzy semiring. In fact, excluding the complementation properties, the interval $[0,1]$ satisfies all of De Morgan's laws with respect to max and min operations. It may be pointed out here that the boolean $\{0, 1\}$ also forms a semiring with respect to binary AND and OR operations. Now given the fuzzy column vectors $(\boldsymbol{h}_1, \dots, \boldsymbol{h}_n)$ (*i.e.*, $\boldsymbol{h}_i \in A^n$, where A is a fuzzy semiring), a fuzzy column $\hat{\boldsymbol{h}}$ is said to belong to the fuzzy A-subsemimodule generated by the set of generators $\{\boldsymbol{h}_1, \dots, \boldsymbol{h}_n\}$ if there exists a set of parameters $\delta_1, \dots, \delta_n$, where not all $\delta_i = 0$ and $\delta \in [0, 1]$, such that

$$\hat{\boldsymbol{h}} = \max\left[\min(\delta_1, h_2), \min(\delta_2, h_2), \dots, \min(\delta_n, h_2)\right]. \qquad (7.8)$$

In this case, we say that $\hat{\boldsymbol{h}}$ is dependent on $\{\boldsymbol{h}_1, \dots, \boldsymbol{h}_n\}$. If no such δ_i, $(\delta_i \neq 0)$ exists, $\hat{\boldsymbol{h}}$ is said to be independent of $\{\boldsymbol{h}_1, \dots, \boldsymbol{h}_n\}$. It may be pointed out here that since the interval $[0, 1]$ does not form a field with respect to max and min operations, the concept of rank of a vector space, which is very useful in determining the independent basis of a vector space, is not applicable in the present context. In the next section we present an algorithm for identifying a set of independent columns of $\boldsymbol{H}(r)$.

In view of the above observation regarding independent columns, the following corollary holds.

Corollary. If A is a fuzzy semiring, then $r \in A^{rat}\langle\langle V_T{}^*\rangle\rangle$ if there are finitely many independent columns of $\boldsymbol{H}(r)$.

Assuming now that $\boldsymbol{H}(r)$ has finitely many independent columns and $r \in A^{rat}\langle\langle V_T{}^*\rangle\rangle$, by Theorems 7.1 and 7.2, r may be expressed as

$$r = \sum_{w \in V_T{}^*} (\alpha(\mu w)\beta)w \qquad (7.9)$$

where α is a row vector, β is a column vector, μ is a representation, and $w \in V_T{}^*$.

Since $\boldsymbol{H}(r)$ has finitely many independent columns, let $\{F_{v_1}, \dots, F_{v_m}\}$ constitute a minimal set of independent columns of $\boldsymbol{H}(r)$ associated with $\{v_1, \dots, v_m\}$ where $v_i \in V_T{}^*$, for $i = 1, \dots, m$ are the strings associated with these columns.

Since $F_{v_1}, \dots, F_{v_m}$ constitute a finite set of independent columns of $\boldsymbol{H}(r)$, xF_{v_i}, $x \in V_T$, must be linearly dependent on $\{F_{v_1}, \dots, F_{v_m}\}$, where xF_{v_i} should be interpreted as in (7.7). Hence xF_{v_i} may be represented as

$$xF_{v_i} = \sum_{j=1}^{m} (\mu x)_{ji} F_{v_j} \quad \text{for } x \in V_T,$$

where $\mu : V_T{}^* \to A^{m\times m}$. We must now establish that μ is a representation. Assuming that the above equation holds for $x = w_1$ and $x = w_2$,

$$\begin{aligned}(w_1w_2)F_{v_1}(v) &= F_{v_i}(uw_1w_2)\\ &= \sum(\mu w_2)_{ji}(F_{v_i})(vw_1)\\ &= \sum(\mu w)_{ji}\sum(\mu w_1)_{jk}F_{v_k}(v)\\ &= \sum(\mu w_1 \mu w_2)_{ki}F_{v_k}(v).\end{aligned}$$

This equation is valid for $x = w_1w_2$. Since it is valid for $x \in V_T$, it also holds for any $x \in V_T{}^*$. Thus to construct μ, we need to consider the dependencies of xF_{v_i} for $i = 1, \ldots, m$ and $x \in V_T$, on $\{F_{v_1}, \ldots, F_{v_m}\}$. Once μ is constructed, α and β can be constructed as follows. Since r belongs to a finitely-generated subsemimodule of $A\langle\langle V_T{}^*\rangle\rangle$, there exist elements $\{\beta_1, \ldots, \beta_m\} \in A$ such that $r = \sum \beta_i F_{v_i}$, where F_{v_1} is now treated as a function in $A^{V_T{}^*}$. Then

$$\begin{aligned}(r, w) &= \sum \beta_i F_{v_i}(w)\\ &= \sum \beta_i(wF_{v_i})\\ &= \sum \beta \sum (\mu w)_{ji} F_{v_i}(\lambda)\\ &= [F_{v_1}(\lambda)F_{v_2}(\lambda)\ldots F_{v_m}(\lambda)] \sum \mu w[\beta_1, \ldots, \beta_m]^T,\end{aligned}$$

where T stands for matrix transposition.

Considering $\alpha = [F_1(\lambda)\ldots F_m(\lambda)]$ and $\beta = [\beta_1, \cdots, \beta_m]^T$, $(r, w) = \sum \lambda(\mu w)\beta$. Here α corresponds to the entries in $F_{v_1}, \ldots, F_{v_m}$ for the row in $\boldsymbol{H}$ labeled by $\lambda \in V_T{}^*$. Also, β_i corresponds to the coefficients of F_{v_i} in the expansion of F_λ in terms of $(F_{v_1}, \ldots, F_{v_m})$.

It may be noted at this point that α and β correspond to the initial and final states, respectively, of fuzzy automaton M. Once α, β and μ are determined, the desired fuzzy automaton that recognizes the strings in R^+ can be constructed.

$$M = \{\Pi, (q_1, \ldots, q_m), F, \eta\}$$

can now be defined with $\Pi = \alpha$, $\eta = \beta$, and $f(q_I, x, q_k) = [\mu(x)]_{ki}$, $x \in V_T$. Thus the steps required to construct the fuzzy automaton that accepts only the strings in R^+ (a positive sample set of strings) of a fuzzy language are as follows:

(1) Construct the fuzzy Hankel matrix $\boldsymbol{H}(r)$.

(2) Identify a complete set of independent columns of $\boldsymbol{H}(r)$.
(3) Obtain the fuzzy vectors α, β and the fuzzy matrices $\mu(x_i)$, $\forall x_i \in V_T$.
(4) Construct the fuzzy automaton.

It should be noted here that while inferring a grammar from a positive set R^+ of samples of finite length, any column corresponding to a word v, $v \in V_T{}^*$, that is not a factorization of any strings $w_i \in R^*$ will be identically zero. The same situation arises in the case of the rows of $\boldsymbol{H}(r)$ corresponding to a word u that is not a factorization of w_i. Thus the Hankel matrix essentially reduces to the form

$$\boldsymbol{H}(r) = \begin{bmatrix} \hat{\boldsymbol{H}}(r) & 0 \\ 0 & 0 \end{bmatrix}$$

where the zeros are infinite matrices and $\hat{\boldsymbol{H}}(r)$ is a finite submatrix of $\boldsymbol{H}(r)$. In the case of recursive production of the strings with cycles, the inference procedure deals with a Hankel matrix of the form

$$\boldsymbol{H}(r) = [\boldsymbol{H}_1(r) \quad \boldsymbol{H}_2(r) \quad 0],$$

where $\boldsymbol{H}_1(r)$ is a finite submatrix and contains all the relevant information. The problem of identification of a set of independent columns of $\boldsymbol{H}(r)$ thus reduces to identifying the set of independent columns of $\boldsymbol{H}(r)$, which will henceforth be designated as $\boldsymbol{H}(r)$ only.

7.6.4.3 *Identification of the set of independent fuzzy column vectors*

We have just defined the dependence of a column vector $\hat{\boldsymbol{h}}$ on $(\boldsymbol{h}_1, \cdots, \boldsymbol{h}_N)$, the set of generators of the finite fuzzy Hankel matrix $\boldsymbol{H}(r)$. Here we present an algorithm that checks whether $\hat{\boldsymbol{h}}$ belongs to the subsemimodule F generated by this set of generators and also identifies its coefficients δ_i. The jth element of the vector $\boldsymbol{h}_k$ will be denoted by h_{jk}. Now given the set of fuzzy column vectors $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N$ of dimension M, a set of row vectors $S(i)$ is formed, for $i = 1, \ldots, M$, as

$$S(i) = \{j | j \in (1, \ldots, N), \hat{h}_i \leq h_{ji}\}.$$

In the following procedure, in order to identify δ_j, $j = 1, \ldots, N$, we examine the dependencies of $\hat{h}_i$ on $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_N\}$ for $i = 1, \ldots, M$. When $\hat{h}_i$ can be expressed in terms of h_{k_i}, $k = 1, \ldots, N$, the coefficients of h_{ji} will be denoted by δ_{ji}, *i.e.*,

$$h_i = \sum_{j=1}^{N} \delta_{ji} h_{ji}.$$

Each such equation identifies a range of admissible value of δ_{ji}. To identify such constraints on δ_{ji}, note that for $k \in S(i)$,

$$\hat{h}_i > h_{ji} \Rightarrow \delta_{ki} \in [0,1],$$

and, for $k \in S(i)$,

$$\hat{h}_i \leq h_{ji} \Rightarrow \delta_{ji} \in [0,1]. \tag{7.10}$$

If $\#[S(i)] = 1$, then δ_{ji} has a single value, *i.e.*, $\delta_{ji} = \hat{h}_i$.

On the other hand, if $\#[S(i)] > 1$ for any $i \in \{1, \ldots, M\}$ and $j \in S(i)$, then the maximum value that δ_{ji} can have is $\delta_{ji\,\max} = \hat{h}_i$. Let δ_{ji_n} and δ_{ji_m} denote the minimum and maximum admissible values of δ_{ji} as dictated by (7.10). Now let

$$\delta_{jv} = \max_{i=1,\ldots,M} (\delta_{ji}),$$

and

$$\delta_{ju} = \min_{i=1,\ldots,M} (\delta_{ji}).$$

When (7.8) is satisfied, δ_j must belong to $[\delta_{ju}, \delta_{jv}]$. Let us select a $\delta_j \in [\delta_{ju}, \delta_{jv}]$ and suppose $R_j = \{i | i \in \{1, \ldots, M\}\}$. We now present the condition of dependence of $\hat{h}$ on the set of fuzzy column vectors in the following theorem.

Theorem 7.3 *A fuzzy column vector $\hat{\boldsymbol{h}}$ is dependent on $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N)$, a set of fuzzy column vectors, if and only if*

$$R_1 \vee R_2 \vee \ldots \vee R_N = \{1, \ldots, M\}.$$

The following corollary immediately follows from Theorem 7.3.

Corollary. A fuzzy column vector $\hat{\boldsymbol{h}} \in A^n$ is independent of a set of fuzzy column vectors $\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n\}$ with $\boldsymbol{h}_i \in A^n$, if $\#[S(i)] = 1$ for any $i \in \{1, \ldots, M\}$.

Now, if a set of column vectors $\boldsymbol{g}_i$, $i = 1, \ldots, n$, is given, a complete set of independent fuzzy vectors $f_i, i = 1, \ldots, L$, can be selected such that the subsemimodule generated by $(f_1, \ldots f_m)$ contains g_i's.

7.6.4.4 *Application*

We now describe an application to the automated inference of fuzzy grammar in character recognition. Each of the classes of alphabetic characters belonging to a language has been coded in the form of a string over $V_T = \{a, b, c, d\}$. Linguistic analysis is carried out for only a small zone of the pattern where the structural dissimilarity among training patterns representing different pattern classes is maximum. For structural analysis these zones are represented by strings of pattern primitives. All the strings of a particular pattern class are next associated with a generative grammar that is not known *a priori.* The grammar corresponding to each class of patterns is next constructed using the inference procedure described above. It may be noted at this point that the positive sample set $R^+L(G)$, ($L(G)$ is the language corresponding to a pattern class whose grammar is G) must be structurally complete with respect to G. Otherwise if a new string not hitherto included in R is accepted by the automaton, the set R is enhanced to include it and the fuzzy Hankel matrix is modified accordingly. The repetition of this procedure continues until the sample set R^+ is complete. We now consider a positive sample set

$$R^+ = \{0.8ab, 0.8aabb, 0.3ab, 0.2bc, 0.9abbc\}.$$

Once the set of independent column vectors is extracted, the next step is to find the matrices $\mu(x), x \in V_T$. For finding the matrices $\mu(x)$, $x \in V$, initially the expression xF has to be completed for $x = a, b, c$ and $i = 1, \ldots, 7$. The matrices $\mu(a)$, $\mu(b)$ and $\mu(c)$ are as follows:

$$\begin{array}{c} \\ \lambda \\ a \\ ab \\ abc \\ bc \\ abb \\ abbc \\ aa \\ aab \\ aabb \end{array}
\begin{array}{c}
\begin{array}{ccccccccccc} \lambda & ab & b & abc & bc & c & abbc & bbc & aabb & abb & bb \end{array} \\
\begin{bmatrix}
0 & .8 & 0 & .3 & .2 & 0 & .9 & 0 & .8 & 0 & 0 \\
0 & 0 & .8 & 0 & .3 & 0 & 0 & .9 & 0 & .8 & 0 \\
.8 & 0 & 0 & 0 & .9 & .3 & 0 & 0 & 0 & 0 & 0 \\
.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & .9 & 0 & 0 & 0 & 0 & 0 \\
.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .8 \\
0 & 0 & .8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}$$

$$\mu(a) = \begin{array}{c|ccccccc} & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 \\ \hline S_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_2 & 0 & 0 & .3 & 0 & 1 & 0 & .8 \\ S_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_5 & 0 & 0 & 0 & 0 & 0 & .8 & 0 \\ S_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\mu(b) = \begin{array}{c|ccccccc} & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 \\ \hline S_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ S_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ S_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ S_7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\mu(c) = \begin{array}{c|ccccccc} & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 \\ \hline S_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

The α_i's can be computed from the relationship

$$\alpha = F_1(\lambda), \ldots, F_m(\lambda),$$

where the vector corresponds to the entries in the set of independent columns $F_1, \ldots, F_m$ for the row in $\boldsymbol{H}(r)$ labeled by λ. Thus

$$\alpha = [0 \;\; 0.9 \quad 0.2 \quad 0 \quad 0 \quad 0 \quad 0]$$

and

$$\beta = [1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$

because F_1 is an independent column. Once α, β, $\mu(a)$, $\mu(b)$, and $\mu(c)$ have been determined, the fuzzy automaton can be constructed as per the method already described. The fuzzy automaton that accepts the strings is shown in Fig. 7.4.

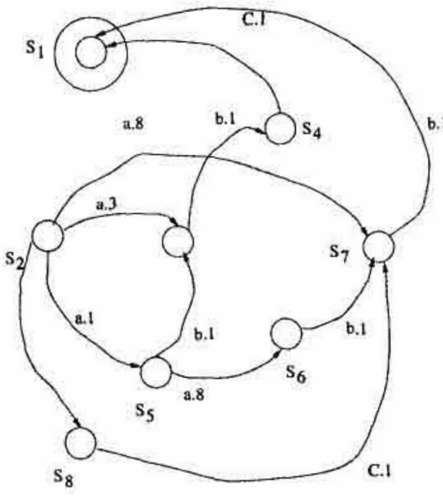


Fig. 7.4 Inferred fuzzy automaton

7.7 Recognition of ill-formed patterns: error-correcting grammars

We have so far discussed syntactic procedures for analysis of perfect or imprecisely-formed pattern strings. However, in many practical applications, patterns are ill-formed and distorted. Various error-correcting grammars have been proposed for analyzing such non-ideal patterns. Three types of errors, namely, substitution, deletion and insertion, have been considered.

7.7.1 Error-correcting procedure

In any syntactic pattern recognition problem, a grammar G describes a class of patterns represented by strings in $L(G)$. The corresponding automaton will accept only those strings which are perfectly formed. An imperfectly-formed string belonging to a pattern class, caused by either a deformation or by noise, will be rejected by the automaton. An error-correcting scheme has been suggested for processing such erroneous strings. A pattern grammar is first inferred from a set of training samples belonging to a particular pattern class. A noisy input string which is rejected by the corresponding automaton is fed to an error-correcting process, which attempts to transform the noisy string into an error-free string so that it may be accepted by the inferred automaton.

The error-correcting procedure uses a discriminant function which indicates the distance between two pattern strings. The computation of the discriminant function may have to be preceded by an adequate stretching operation when the length of the given string is smaller than the length of the sample strings belonging to a particular pattern class. Given a fuzzy regular automaton M_k, inferred from a set R_k of training samples from the kth pattern class, and a set of test strings, this approach uses the procedure STRETCH to recursively modify M_k when a test pattern, even after stretching, is not accepted by M_k. The error correcting procedure is given below:

```
Step 1. Get a new test pattern string $X$.
        Set $i = 1$.
        If $X$ is not accepted by $M_k$ go to step 2, else repeat Step 1.
Step 2. Stretch $X$ to $X_i$ with respect to the training sample $Y_{ik}$ of the kth
        pattern class using STRETCH.
Step 3. Compute $d(X_i, Y_{ik})$.
        If $d(X_i, Y_{ik}) \leq threshold$ go to Step 4, else go to step 5.
Step 4. If $M_k$ accepts $X_i$ go to step 1, else go to Step 5.
Step 5. Set $i = i + 1$.
        If $i \leq N$ go to Step 2, else go to Step 6.
Step 6. Compute $Disc(X, Y_k) = \min [d(X_1, Y_{1k}), \ldots, d(X_N, Y_{NK})]$.
        If $Disc(X, Y_k) \leq Threshold$ go to Step 7, else go to Step 8.
Step 7. Modify $M_k$ with $R_k = R_k \cup \{X\}$. Go to Step 1.
Step 8. The pattern $X$ does not belong to class $k$. Go to Step 1.
```

Any new string of a particular pattern class which is not accepted by the automaton will first be fed to the error-correcting procedure and, if necessary, the inferred grammar corresponding to the said language will be modified accordingly by including the error-corrected string in the language.

7.7.2 Error correction based on similarity enhancement

This error-correcting procedure is based on a similarity criterion using discriminant function, indicating the distance between two pattern strings. Before defining the distance between two strings, we denote the distance between two terminal alphabets (*i.e.*, pattern primitives) a_i and a_j, by $|a_i - a_j|$, $a_i, a_j \in V_T$. The discriminant function between two strings X and $Y \in V_T{}^*$ of equal length N gives a measure of the distance between the two strings and is given by

$$d(X,Y) = \frac{1}{N}\left[\sum_{i=1}^{N}(X_i - Y_i)^2\right]^{\frac{1}{2}}$$

where $X_i, Y_i \in V_T$ are the ith terminals of the strings X and Y respectively. In order to apply this definition of the discriminant function for strings of unequal length, we first elongate the string having smaller length by the following stretching procedure. If X_i and X_j are two strings of lengths p and q respectively ($p < q$) corresponding to ith and jth pattern classes, and if $X_i{}^k = X_j{}^k$, where $X_i{}^k$ denotes the kth primitive of X_i, then we insert $X_j{}^{k+1}$ at $(k+1)$th position in the ith string also, *i.e.*, after insertion, $X_i{}^{k+1} = X_j{}^{k+1}$. The above operation produces a stretched string and those of the sample strings of the same pattern class are minimized. The proposed stretching procedure results in similarity intensification of a set of samples from a particular pattern class. The algorithm for stretching a string is given below.

The process of stretching may continue until the lengths of the two strings become equal, or may terminate at any intermediate stage, if no such matching is possible. Once the stretching procedure is over, the discriminant function between the two strings is computed.

In case the two strings remain unequal even after stretching, the discriminant function is computed after appending the worst-case primitives to the string with smaller length. Thus for two strings X_i and X_j belonging to the ith and jth pattern classes with lengths p and q ($q > p$), $(q-p)$ primitives may have to be appended to the string of length p when X_i cannot be stretched at all with respect to X_j. The worst-case primitives to be appended always result in maximum distance between the two strings.

Consider the fuzzy automaton that accepts R^+ as shown in Fig. 7.4. When a new noisy string $0.5cbcc$ belonging to the same pattern class is fed to the automaton, it is initially rejected by the automaton. After sufficient error correction it is again fed to the inference procedure and the inferred fuzzy automaton is next modified accordingly. The modified fuzzy automaton is shown in Fig. 7.5. Another string cbc, after stretching operation, is transformed into a string $cbbc$ and is directly accepted by the modified automaton.

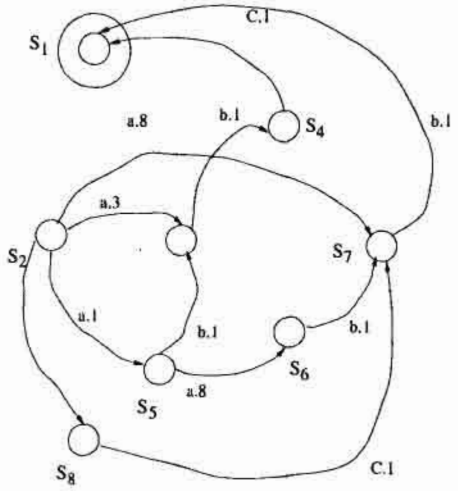


Fig. 7.5 Error-corrected fuzzy automaton

7.7.3 Error-correcting tree grammar

Lu and Fu [35] have proposed an error-correcting parser which utilizes the following tree editing operations:

- insertion of a labelled node in between two nodes in a tree
- insertion of a node to the left of one node and to the right of it
- deletion of a node which has a maximum of one successor node
- label substitution of a leaf node

Zhang and Shasha [59] were the first to report an algorithm for computing distances between two strings in polynomial time. In a related work, Oommen and Kashyap [41] have investigated a similarity measure between two trees. A major direction of research involves inferring a tree automaton which recognizes a class of patterns belonging to a specific tree language. For this purpose, it is important to compute the distance between an unknown pattern represented by a tree and a given tree automaton.

References

[1] K. Aizawa and A. Nakamura, "Quadtree adjoining grammar," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 13, pp. 573–588, 1999.

[2] J. Baldwin and S. Zhou, "A fuzzy relational inference language," Tech. Rep. EM/FS/32, University of Bristol, Bristol, UK, 1982.

[3] A. W. Biermann and J. A. Feldman, "On the synthesis of finite-state acceptors," Stanford Artificial Intelligence Project Memo no. AIM-114, Stanford University, Stanford, CA, 1970.

[4] A. W. Biermann and J. A. Feldman, "A survey of results in grammatical inference," International Conference on Frontiers of Pattern Recognition, Honolulu, Hawaii, 1971.

[5] A. W. Biermann and J. A. Feldman, "On the synthesis of finite state machine from samples of their behaviour," *IEEE Transactions on Computers*, vol. 21, pp. 592–597, 1972.

[6] P. Biswas, *A fuzzy hybrid model for pattern classification with application to recognition of handprinted Devanagari*. PhD thesis, Jawaharlal Nehru University, New Delhi, 1980.

[7] A. Blumenkraans, "Two-dimensional object recognition using a two-dimensional polar transform", *Pattern Recognition*, vol. 24, pp. 879–890, 1991.

[8] E. Bribiesca, "A new chain code," *Pattern Recognition*, vol. 32, pp. 235–251, 1999.

[9] W. Chomsky, *Syntactic Structures*. The Hague, The Netherlands: Monton, 1957.

[10] G. F. DePalma and S. S. Yau, "Fractionally fuzzy grammars with application to pattern recognition", in *Fuzzy Sets and their Applications to Cognitive and Decision Processes* (L. A. Zadeh, K. S. Fu, K. Tanaka and M. Shimura, eds.), pp. 329–352, New York: Academic Press, 1975.

[11] S. Eilenberg, *Automata, Languages and Machines*, vol. A. New York: Academic, 1974.

[12] S. Ewert and A. van der Walt, "Generating pictures using random permitting context," *International journal of Pattern Recognition and Artificial Intelligence*, vol. 13, pp. 339–355, 1999.

[13] J. Feder, "Plex languages," *Information Sciences*, vol. 3, pp. 225–241, 1971.

[14] J. Feldman, "First thoughts on grammatical inference." Artificial Intelligence Memo no. 55, Computer Science Dept., Stanford University, Stanford, CA, 1967.

[15] J. A. Feldman, J. Gips, J. J. Horning and S. Reder, "Grammatical complexity and inference", Tech. Rep. no. CS-TR-69-125, Computer Science Dept., Stanford University, Stanford, CA, 1969.

[16] M. Fliess, "Sur divers produits de series formellers," *Bull. Soc. Math., France*, vol. 102, pp. 181–191, 1974.

[17] H. Freeman,"On the encoding of arbitrary geometric configurations," *IEEE Transactions on Electronic Computers*, vol. EC-10, pp. 260–268, 1961.

[18] K. S. Fu, *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[19] K. S. Fu and B. K. Bhargava, "Tree systems for syntactic pattern recognition," *IEEE Transactions on Computers*, vol. C–22, no. 12, pp. 1087–1099, 1973.

[20] K. S. Fu and T. L. Booth, "Grammatical inference : introduction and survey, part I and part II," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 95-11, no. 5, pp. 409–423, 1975.

[21] K. S. Fu and T. Huang, "Stochastic grammars and languages", *International Journal of Computer and Information Sciences*, vol. 1, pp. 135–170, 1972.

[22] E. M. Gold, "Language identification in the limit," *Information and Control*, vol. 10, pp. 447–474, 1967.

[23] R. C. Gonzalez and M. G. Thomason, "On the inference of tree grammars for syntactic pattern recognition", Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Dallas, TX, 1974.

[24] R. C. Gonzalez, J. J. Edwards and M. G. Thomason, "An algorithm for the inference of tree grammars", *International Journal of Computer*

and Information Sciences, vol. 5, pp. 145-164, 1976.

[25] R. C. Gonzalez and M. G. Thomason, *Syntactic Pattern Recognition*. Reading, MA: Addison-Wesley, 1978.

[26] J. E. Hopcroft and J. D. Ullman, *Formal Languages and their Relation to Automata*, Reading, MA: Addison-Wesley, 1969.

[27] J. Horning, "A study of grammatical inference," Tech. Rep. CS-139, Computer Science Dept., Stanford University, Stanford, CA, 1969.

[28] K. S. Fu and T. Huang, "Stochastic grammars and languages," *International Journal of Computer and Information Sciences*, vol. 1, pp. 135–170, 1972.

[29] A. K. Joshi and Y. Schabes, "Tree adjoining grammars," in *Beyond Words: Handbook of Formal Languages* (G. Rozenberg and A. Salomaa, eds.), vol. 3, pp. 69–123, Berlin: Springer-Verlag, 1997.

[30] A. Kandel and S. Lee, *Fuzzy Switching and Automata: Theory and Applications*. New York: Crase Russak, 1979.

[31] W. J. M. Kickert and H. Koppelaar, "Application of fuzzy set theory to syntactic pattern recognition of handwritten capitals", *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-6, pp. 148–151, 1976.

[32] R. S. Ledley *et al.*, "FIDAC: Film input to digital automatic computer and associated syntax-directed pattern-recognition programming system," in *Optical and Electro-optical Information Processing* (J. T. Tippet, D. Beckowitz, L. Clapp, C. Koester and A. Vanderburgh, Jr., eds.), pp. 591–613, Cambridge, MA: MIT Press, 1965.

[33] E. T. Lee and L. A. Zadeh, "Note on fuzzy languages," *Information Sciences*, vol. 1, pp. 421–434, 1969.

[34] P. Lopez, "Extended partial parsing for lexicalized tree grammars", in *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pp. 159–170, Trento, Italy, 2000.

[35] H. R. Lu and K. S. Fu, "A general approach to inference of context-free programmed grammars", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 14, pp. 191–202, 1984.

[36] F. J. Maryanski and T. L. Booth, "Inference of finite-state probabilistic grammars," *IEEE Transactions on Computers*, vol. C-26, pp. 521–536, 1977.

[37] D. L. Milgram and A. Rosenfeld, "Array automata and array grammars," in *Proceedings of the IFIP Congress*, pp. 166–173, North Holland, 1971.

[38] B. Moayer and K. S. Fu, "A tree system approach to finger print pattern recognition," *IEEE Transactions on Computer*, vol. C-25, pp. 262–274, 1976.

[39] M. Mizumoto, J. Toyoda and K. Tanaka, "Some considerations on fuzzy automata", *Journal of Computer and System Sciences*, vol. 3, pp. 409–422, 1969.

[40] R. Narasimhan, "A linguistic approach to pattern recognition", Tech. Rep. no. 121, Department of Computer Science, University of Illinois at Urbana-Champaign. Urbana, IL, 1962.

[41] B. J. Oommen and R. L. Kashyap, "Optimal and information theoretic syntactic pattern recognition for traditional errors", in *Advances in Structural and Syntactic Pattern Recognition*, (P. Perner, P. Wang, and A. Rosenfeld, eds.), New York: Springer, pp. 11–20, 1996.

[42] A. Pathak and S. K. Pal, "Fuzzy grammars in syntactic recognition of skeletal maturity from X-rays", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, pp. 657–667, 1984.

[43] T. Pao, "A solution of the syntactical induction inference problem for a nontrivial subset of context free languages," Tech. Rep. no. 69–16, Moore School of Electrical engineering, University of Pennsylvania, Philadelphia, 1969.

[44] J. L. Pfaltz, "Web grammar and picture description," *Computer Graphics and Image Processing*, vol. 1, pp. 193–220, 1972.

[45] M. Richetin and F. Vernadat, "Efficient regular grammatical inference for pattern recognition," *Pattern Recognition*, vol. 17, pp. 245–250, 1984.

[46] A. Salomaa and M. Soittola, *Automata Theoretic Aspects of Formal Power Series*. New York: Springer-Verlag, 1978.

[47] E. S. Santos, "Fuzzy automata and languages," US-Japan Seminar on Fuzzy Sets and their applications, Berkeley, 1974.

[48] E. S. Santos, "Regular fuzzy expressions", in *Fuzzy Automata and Decision Processes* (M.M. Gupta, G. N. Saridis and B. R. Gaines, eds.), New York: North–Holland, pp. 169–176, 1977.

[49] M. P. Schutzenberger, "On definition of a family of automata," *Information and Control*, vol. 4, pp. 245–270, 1961.

[50] A. C. Shaw, "The formal picture description scheme as a basis for picture processing systems," *Information and Control*, vol. 14, pp. 9–52, 1969.

[51] G. S. Sidhu and R. T. Boute, "Property encoding applications in binary picture encoding and boundary following," *IEEE Transactions on Computers*, vol. C-21, pp. 1206–1216, 1972.

[52] R. J. Solomonoff, "A new method for discovering the grammars of phrase structure languages ," *Information Processing*, New York: UNESCO, 1959.

[53] W. Stallings, "Approaches to Chinese character recognition", *Pattern Recognition*, vol. 8, pp. 87–98, 1976.

[54] M. G. Thomason, "Syntactic pattern recognition: stochastic languages", in *Handbook of Pattern Recognition and Image Processing*, (T. Y. Young and K. S. Fu, eds.), Orlando, FL: Academic Press, pp. 119–142, 1986.

[55] M.G.Thomason and P. Marinos, "Deterministic acceptors of regular and fuzzy languages," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-4, pp. 228–230, 1974.

[56] W. H. Tsai and K. S. Fu, "Subgraph error-correcting isomorphisms for syntactic pattern recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, pp. 48-62, 1983.

[57] W. Wee and K. S. Fu, "A formulation of fuzzy automata and its application as a model of learning systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 5, pp. 215–223, 1969.

[58] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[59] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM Journal of Computing*, vol. 18, pp. 1245–1262, 1989.

Chapter 8

FUZZY SETS AS A LOGIC CANVAS FOR PATTERN RECOGNITION

W. Pedrycz[*] and N. Pizzi[†]

[*] *University of Alberta*
Department of Electrical and Computer Engineering
Edmonton, AB, T6G 2G7, CANADA
e-mail: *pedrycz@ee.ualberta.ca*

[†] *National Research Council Canada*
Institute for Biodiagnostics
Winnipeg, MB, R3B 1Y6, CANADA
e-mail: *Nicolino.Pizzi@nrc.ca*

Abstract

Imprecision and uncertainty are hallmarks of many pattern recognition problems. Although often impervious to crisp reasoning methods, these problems may be open to approximate ones. For this reason, the field has become a fertile and active domain with which to exercise fuzzy set-based modes of reasoning. For example, fuzzy sets support and enhance the design of logic-driven learning methods by: (i) formalizing prior domain knowledge; (ii) promoting hierarchical and modular architectures that exploit the most appropriate level of information granulation; (iii) providing an intuitive mapping between information couched in imprecise linguistic terminology and a well-defined contextual space. We focus this investigation on fuzzy set-based generalizations of logic processors and Petri nets as well as the role fuzzy sets may play in context-based partially supervised modes of clustering.

8.1 Introduction: fuzzy sets and pattern recognition

Pattern recognition, with its underlying philosophy, fundamental methodology, and suite of advanced algorithms, has established itself as a mature, well-developed, information technology. Pattern recognition is a heterogeneous area exploiting a diversity of processing paradigms such as probability, statistics, neural networks and fuzzy sets. Fuzzy sets entered the pattern recognition arena quite early, for example, [2, 20] and they have a lot to offer in this domain especially as its research agenda is in line with the key pursuits of pattern recognition. Not attempting in any way to be exhaustive in the coverage of the area [7, 8, 14, 19, 23] (a somewhat futile task) we simply wish to emphasize the main trends, as summarized below:

- Fuzzy sets contribute to pattern recognition as a vehicle formalizing any prior domain knowledge about the classification environment. This facilitates decisions about the classifiers structure, enhances learning methods (by avoiding cumbersome learning from scratch), and supports the design of logic-driven, transparent classifier architectures that are easier to train and easier to comprehend. In other words, fuzzy sets act a conceptual canvas for logic-based and designer/user-oriented classifiers.
- Fuzzy sets promote a notion of partial membership to a class. This is in accordance with many classification tasks where class assignment may not be binary.
- By emphasizing the role of information granulation, fuzzy sets promote a hierarchical and modular approach to pattern recognition, which becomes indispensable when handling complex classification problems in a modular fashion. In particular, this manifests itself in the form of hierarchical architectures with a clearly delineated knowledge-based layer and gives rise to the hierarchy of low-end, more specialized local classifiers located at the lower end. The role of the upper layer of the overall hierarchy (which is implemented with the aid of fuzzy sets and fuzzy logic) is to schedule activities of the classifiers at the lower end and

summarize the findings there and eventually repair some discrepancies. The modularization of the classifier has strong evidence in practice. In general, one may develop individual, local classifiers that cope with regions in the feature space that are different as to the separability level of the patterns belonging to different classes. We have to deal with regions of the feature space where classes are well separated as well as regions where the classes overlap. In the first instance, a linear classifier may be a good option. On the other hand, a nearest neighbor classifier is a good option for the regions where the classes overlap.

- Traditionally, pattern recognition techniques are divided into two main groups: supervised and unsupervised classification schemes. While generally useful, this simplified taxonomy is often not in rapport with real-world classification scenarios. Quite often we may encounter situations that fall in-between these two fundamental scenarios. For instance, there could be a mixture of labeled and unlabeled patterns. This calls for unsupervised learning (clustering) that comes with a certain level of partial supervision.

The objective of this study is to investigate the role of fuzzy sets in pattern recognition along the main points highlighted above. In particular, we discuss several topologies in which the logic-based facet is inherent due to the incorporation of fuzzy sets. The focus will be logic-oriented classifiers (built around a basic architecture of a logic processor) and fuzzy Petri nets. Next, we analyze various aspects of designing interactions between the modes of unsupervised and supervised learning by showing how domain knowledge about class membership can be reconciled with the internal structure of the data. To be more specific, this type of essential interaction is devised in the setting of the well-known FCM-like clustering methods [3, 6]. As a prerequisite to fuzzy set-based computing, we adhere to triangular norms and co-norms (t- and s-norms), viewed as basic models of logic operators on fuzzy sets and relations.

8.2 Fuzzy set-based transparent topologies of the pattern classifier

Neural networks [16] occupy a significant niche in pattern recognition mainly in the capacity of nonlinear classifiers. While capable of carrying out various learning schemes, neural networks are inherently "black-box" architectures. This has two important implications: meaningful interpretations of the network's structure are difficult to derive; any prior knowledge we may possess about the classification problem, which would reduce the total learning effort, is difficult to directly "download" onto the network (the common practice is to "learn from scratch"). A solution to the problem would be to construct networks that are similar to neural networks in terms of their learning abilities while at the same time composed of easily interpretable processing elements. To this end, a logic-oriented processing style is definitely an asset. Bearing this in mind, [11] proposed two general classes of fuzzy AND and OR neurons. They serve as generalizations of standard AND and OR digital gates. The AND neuron is a static n-input single output processing element $y = \mathrm{AND}(\mathbf{x}; \mathbf{w})$ constructed with the use of fuzzy set operators (t- and s-norms),

$$y = \mathop{\mathrm{T}}_{i=1}^{n} (x_i s w_i) \tag{8.1}$$

Here $\mathbf{w} = (\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_n})'$ denotes a weight vector (connections) of the neuron. In light of the boundary conditions of the triangular norms, we obtain

- If $w_i = 1$, then the corresponding input has no impact on the output. Moreover, the monotonicity property holds: higher values of w_i reduce the impact of x_i on the output of the neuron.
- If the elements of $\mathbf{w}$ assume values equal to 0 or 1, then the AND neuron becomes a standard AND gate. The OR neuron, denoted as $y = \mathrm{OR}(\mathbf{x}; \mathbf{w})$ is described in the form

$$y = \mathop{\mathrm{S}}_{i=1}^{n} (x_i t w_i) \tag{8.2}$$

As with AND neurons, the same general properties hold; the boundary conditions are somewhat complementary: the higher the connection value, the more evident the impact of the associated input on the output.

The fundamental Shannon's expansion theorem [17] states that any Boolean function can be represented as a sum of minterms (or equivalently, a product of maxterms). The realization is a two-layer digital network: the first layer has AND gates (realizing the required minterms); the second consists of OR gates that carry out OR-operations on the minterms.

Fuzzy neurons operate in an environment of continuous variables. An analogy of the Shannon theorem (and the resulting topology of the network) can

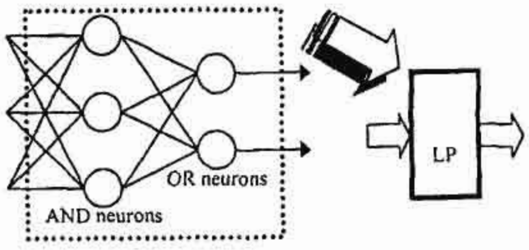


Fig. 8.1 Logic processor (LP): a general topology; the inputs involve both direct and complemented inputs

be realized in the form illustrated in Fig. 8.1. Here, AND neurons form a series of generalized minterms. The OR neurons serve as generalized maxterms. This network *approximates* experimental continuous data in a *logic-oriented* manner. In contrast, note that the sum of minterms *represents* Boolean data. The connections of the neurons equip the network with the required parametric flexibility. Alluding to the nature of approximation accomplished here, we will refer to the network as a logic processor (LP). The logic-based nature of the LP is crucial to a broad class of pattern classifiers. Fig. 8.2 portrays a common situation encountered in pattern classification. Patterns belonging to the same class (say, w) occupy several groups (regions). For the sake of discussion, we consider only two features over which we define several fuzzy sets (granular landmarks) and through which we "sense" the patterns. The AND neurons build (combine) the fuzzy sets defined for individual variables. Structurally, these neurons develop fuzzy relations, regions where the patterns are concentrated. Then all these regions are aggregated using a single OR neuron. The connections help capture the details of the regions (AND neuron) and contribution of each region to the classification decision (OR neuron). In this setting, one may refer to radial basis function (RBF) neural networks that resemble the above logic structure yet the latter do not carry any strong logic connotation. The relationship between the components of the networks coming from these two groups is summarized in Table 8.2.

8.2.1 Fuzzy generalization of the Petri net

Let us briefly recall the basic concept of a Petri net. A Petri net is a finite graph with two types of nodes, known as places, P, and transitions, T. More

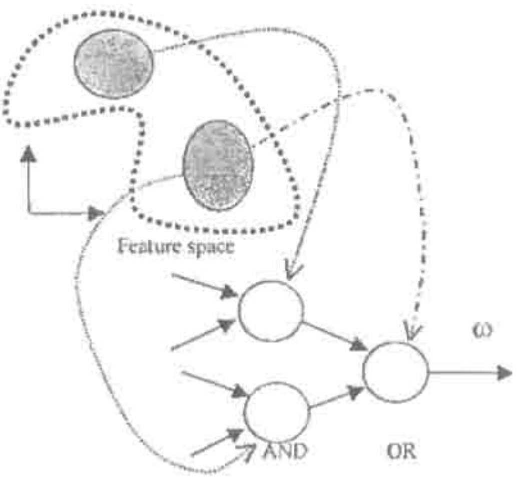


Fig. 8.2 LP as a fundamental logic-driven classification structure

formally, the net can be viewed as a triplet (P, T, F) where,

$$\begin{aligned} &P \cap T = \emptyset \\ &P \cup T = \emptyset \\ &F \subseteq (P \times T) \cup (T \times P) \\ &\text{domain}(F) \cup \text{codomain}(F) = P \cup T \end{aligned} \tag{8.3}$$

In (8.4), F is the flow relation. The elements of F are the arcs of the Petri net. Each place comes equipped with some tokens that form a marking of the Petri net. The flow of tokens in the net occurs through firings of the transitions; once all input places of a given transition have a nonzero number of tokens, this transition fires. Subsequently, the tokens are allocated to the output places of the transition. Simultaneously, the number of tokens at the input places is reduced. The effect of firing the transitions is binary: the transition either fires or does not fire.

An important generalization of the generic model of the Petri net is to relax the Boolean character of the firing process of the transition. Instead of subscribing to the firing-no firing dichotomy, we propose to view the firing mechanism as a gradual process with a continuum of possible numeric values of the firing strength (intensity) of a given transition. Subsequently, the flow

Table 8.1 Logic processor and RBF neural networks: a comparative analysis

Logic Processor	**RBF Neural Network**
AND neurons: form the fuzzy relations of features in the feature space. Identify separate concentrations of pattern regions belonging to a given class. Learn neuron connections in order to model the details of these regions.	RBF (receptive fields) are used to identify regions in the feature space that are homogeneous in terms of patterns belonging to the class of interest. RBF may assume various functional forms (*e.g.*, Gaussians) and could be adjusted by modifying its parameters (usually modal value and spread).
OR neurons: summarize the separate concentrations of pattern regions. Use connections to model impact of pattern regions on final classification decision. Some regions (outputs of AND neurons) need to be discounted, especially if they are heterogeneous and include patterns belonging to other classes.	A linear summation: the outputs of the RBFs (activation levels of these fields) are processed through a weighted sum where the weights are used to cope with the impact of the receptive fields on the final classification outcome.

of tokens can also take this continuum into consideration. Evidently, such a model is in rapport with a broad class of real-world phenomena including pattern classification. The generalization of the net along this line calls for a series of pertinent realization details. In what follows, we propose a construct whose functioning adheres as much as possible to the logic fabric delivered by fuzzy sets. In this case, a sound solution is to adopt the ideas of fuzzy logic as the most direct way of implementation of such networks.

8.2.1.1 *The architecture of the fuzzy Petri net*

Cast in the framework of pattern classification, the topology of the fuzzy Petri net is portrayed in Fig. 8.3. As it will be shown further on, this setting correlates well with the classification activities encountered in any process of pattern

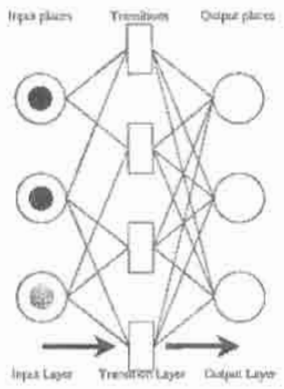


Fig. 8.3 A general three-layer topology of the fuzzy Petri net

recognition.

The network constructed in this manner comprises three layers:

- An input layer composed of n input places;
- A transition layer composed of hidden transitions;
- An output layer consisting of m output places.

The input place is marked by the value of the feature (we assume that the range of the values of each feature is in the unit interval). These marking levels are processed by the transitions of the network whose levels of firing depend on the parameters associated with each transition such as their threshold values and the weights (connections) of the incoming features. Subsequently, each output place corresponds to a class of patterns distinguished in the problem. The marking of this output place reflects a level of membership of the pattern in the corresponding class. The detailed formulas of the transitions and output places rely on the logic operations encountered in the theory of fuzzy sets. The i^{th} transition (more precisely, its activation level z_i) is governed by the expression

$$z_i = \operatorname*{T}_{j=1}^{n} \left[w_{ij} s(r_{ij} \rightarrow x_j) \right] \tag{8.4}$$

where

- w_{ij} is a weight (connection) between transition i and input place j;

- r_{ij} is a marking level threshold for input place j and transition i;
- x_j is the marking level of input place j;
- T denotes a t-norm and s denotes an s-norm.

The implication operator, $\rightarrow$, is expressed in the form

$$a \rightarrow b \;=\; \sup\{c \in [0,1] \mid a\,t\,c \leq b\} \tag{8.5}$$

where a and b are the arguments of the implication operator confined to the unit interval. Note that the implication is induced by a certain t-norm. In the case of two-valued logic, (8.5) returns the same truth values as the commonly-known implication operator, namely,

$$\begin{aligned} a \rightarrow b &= \begin{cases} b \text{ if } a > b \\ 1 \text{ otherwise} \end{cases} \\ &= \begin{cases} 0 \text{ if } a = 1, \quad b = 0 \\ 1 \text{ otherwise} \end{cases} \end{aligned} \tag{8.6}$$

Output place j (more precisely, its marking y_j) summarizes the levels of evidence produced by the transition layer and performs a nonlinear mapping of the weighted sum of the activation levels of these transitions, z_i, and the associated connections v_{ji},

$$y_j = f\left(\sum_{i=1}^{hidden} v_{ji} z_i\right) \tag{8.7}$$

where f is a nonlinear monotonically increasing mapping from $\Re$ to [0,1]. The role of the connections of the output places is to modulate the impact exhibited by the firing of the individual transitions on the accumulation of the tokens at the output places (*viz.*, the membership value of the respective class). The negative values of the connections have an inhibitory effect, that is, the value of the class membership gets reduced. Owing to the type of aggregation operations used in the realization of the transitions, their interpretation sheds light on the way in which the individual features of the problem are treated. In essence, the transition produces some higher level, synthetic features out of those originally encountered in the problem and represented in the form of the input places. The weight (connection) expresses a global contribution of feature j to transition i: the lower the value of w_{ij}, the more significant the contribution of the feature to the formation of the synthetic aggregate feature formed at the transition level. The connection itself is weighted uniformly regardless of the numeric values it assumes. The more selective (refined) aggregation mechanism is used when considering threshold values. Referring to (8.4), one finds that the thresholding operation returns 1 if x_j exceeds the value of the threshold r_{ij}. In other words, depending on this level of the threshold, the level of marking of the input place becomes "masked" and the threshold operation returns 1. For the lower values of the marking, such levels are processed by the implication operation and contribute to the overall level of the firing of the transition.

One should emphasize that the generalization of the Petri net proposed here is in full agreement with the two-valued generic version of the net commonly encountered in the literature. Consider, for instance, a single transition (transition node). Let all its connections and thresholds be restricted to $\{0,1\}$. Similarly, the marking of the input places is also quantified in a binary way. Then the following observations are valid:

- Only those input places are relevant to the functioning of transition i for which the corresponding connections are set to 0 and whose thresholds are equal to 1. Denote a family of these places by P.
- The firing level of transition i is described by

$$Z_i \;=\; \overset{n}{\underset{j\in\mathcal{P}}{\mathrm{T}}}\,(r_{ij} \rightarrow x_j) \tag{8.8}$$

It becomes apparent that the firing level is equal to 1 if and only if the marking of all input places in P assumes the value 1; the above expression for the transition is an *and*-combination of the marking levels of the places in P,

$$Z_i \;=\; \overset{n}{\underset{j\in\mathcal{P}}{\mathrm{T}}}\, x_j \tag{8.9}$$

(Let us recall that any t-norm can be used to model the and operation; moreover all t-norms are equivalent when operating on the 0-1 truth-values). Fuzzy Petri nets deliver another resource-based insight into the way in which pattern classifiers function. The role of resources is assumed by the features. Input places relate to fuzzy sets defined in the feature space. The higher the marking (satisfaction) of the input places, the higher the transition level, which translates into the output place's firing level (class assignment).

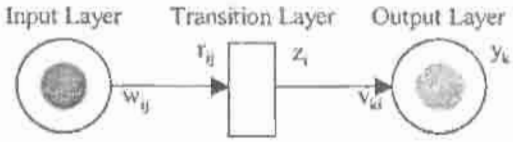


Fig. 8.4 Optimization in the fuzzy Petri net; a section of the net outlines all notation being used in the learning algorithm

8.2.1.2 *The learning procedure*

Learning in the fuzzy Petri net is parametric in nature, meaning that it focuses on changes (updates) of the parameters (connections and thresholds) of the net, its structure is unchanged. These updates are carried out in such a way that a predefined performance index is minimized. To concentrate on the detailed derivation of the learning formulas, it is advantageous to view a fully annotated portion of the network as illustrated in Fig. 8.4. The performance index to be minimized is viewed as a standard sum of squared errors. The errors are expressed as differences between the levels of marking of the output places of the network and their target values. The considered on-line learning assumes that the modifications to the parameters of the transitions and output places occur after presenting an individual pair of the training sets, say marking of the input places (denoted by x) and the target values (namely, the required marking of the output places) expressed by t. Then the performance index for the input-output pair is

$$Q = \sum_{k=1}^{m}(t_k - y_k)^2 \tag{8.10}$$

The connection updates are governed by the standard gradient-based method

$$param(iter+1) = param(iter) - \alpha \nabla_{param} Q \tag{8.11}$$

where $\nabla_{param} Q$ is a gradient of the performance index Q taken with respect to the parameters of the fuzzy Petri net. The iterative character of the learning scheme is underlined by the parameter vector regarded as a function of successive learning epochs. The learning intensity is controlled by the positive learning rate, α. In the above scheme, the vector of parameters, $param$, is used to encapsulate the elements of the structure to be optimized. A complete description of the update mechanism will be described below. With the quadratic performance index ((8.10)) in mind, the following holds:

$$\nabla \boldsymbol{param} Q = -2\sum_{k=1}^{m}(t_k - y_k)\nabla \boldsymbol{param} y_k \tag{8.12}$$

Refer to Fig. 8.4 for the following derivations. The nonlinear function associated with the output place ((8.7)) is a standard sigmoid nonlinearity

$$y_k = \frac{1}{1+e^{-z_k}} \tag{8.13}$$

For the connections of the output places we obtain

$$\frac{\partial y_k}{\partial v_{ki}} = y_k(1-y_k)z_1 \tag{8.14}$$

where $k = 1, 2, \cdots, m$ and $i = 1, 2, \cdots, hidden$. Observe that the derivative of the sigmoidal function is equal to $y_k(1-y_k)$. Similarly, the updates of the threshold levels of the transitions of the net are expressed in the form

$$\frac{\partial y_k}{\partial r_{ij}} = \frac{\partial y_k}{\partial z_i}\frac{\partial y_i}{\partial r_{ij}} \tag{8.15}$$

where $i = 1, 2, \cdots, n$. We then obtain

$$\frac{\partial y_k}{\partial z_i} = y_k(1-y_k)v_{kj} \tag{8.16}$$

and

$$\begin{aligned}\frac{\partial y_k}{\partial r_{ij}} &= A\frac{\partial}{\partial r_{ij}}(w_{ij} + (r_{ij} \to x_j) - w_{ij}(r_{ij} \to x_j)) \\ &= A(1-w_{ij})\frac{\partial}{\partial r_{ij}}(r_{ij} \to x_j)\end{aligned} \tag{8.17}$$

where the new expression, denoted by A, is defined by taking the t-norm over all the arguments except j, as follows:

$$A = \sum_{\substack{l=1 \\ l\neq 1}}^{n}[w_{ij}s(r_{ij} \to x_1)] \tag{8.18}$$

The calculations of the derivative of the implication operation can be completed once we confine ourselves to some specific realization of the t-norm that is involved in its induction. For the product (being a particular example of the t-norm), the detailed result reads as

$$\frac{\partial}{\partial r_{ij}}(r_{ij} \to x_j) = \frac{\partial}{\partial r_{ij}} \begin{cases} \frac{x_j}{r_{ij}} & \text{if } r_{ij} > x_j \\ 1 & \text{otherwise} \end{cases} \tag{8.19}$$

$$= \begin{cases} -\frac{x_j}{r_{ij}} & \text{if } r_{ij} > x_j \\ 0 & \text{otherwise} \end{cases}$$

The derivatives of the connections of the transitions (transition nodes) are obtained in a similar way

$$\frac{\partial y_k}{\partial w_{ij}} = \frac{\partial y_k}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} \tag{8.20}$$

where $k = 1, 2, \cdots, m$, $i = 1, 2, \cdots, hidden$, and $j = 1, 2, \cdots, n$. Subsequently,

$$\frac{\partial z_i}{\partial w_{ij}} = A\frac{\partial}{\partial w_{ij}}(w_{ij} + (r_{ij} \to x_i) - w_{ij}(r_{ij} \to x_i))$$

$$= A(1 - (r_{ij} \to x_j)) \tag{8.21}$$

There are two aspects of further optimization of the fuzzy Petri nets that need to be raised in the context of their learning:

- *The number of transition layer nodes.* The optimization of the number of the transition nodes of the fuzzy Petri net falls under the category of structural optimization that cannot be handled by the gradient-based mechanisms of the parametric learning. By increasing the number of these nodes, we enhance the mapping properties of the net, as each transition can be fine-tuned to fully reflect the boundaries between the classes. Too many of these transitions, however, could easily develop a memorization effect that is well known in neural networks.
- *The choice of t and s-norms.* This leads us to a semi-parametric optimization of the fuzzy Petri net. The choice of these norms does not impact upon the net architecture but in this optimization we cannot resort to gradient-based learning. A prudent strategy would be to confine oneself to a family of t and s-norms that can be systematically exploited.

Table 8.2 summarizes the main features of the fuzzy Petri nets and contrasts these with the structures with which the proposed constructs have a lot in common, namely, Petri nets and neural networks. Fuzzy Petri nets combine the advantages of both neural networks in terms of their learning abilities with the glass box-style processing (and architectures) of Petri nets.

Table 8.2 A comparative analysis of fuzzy Petri nets, Petri nets, and neural networks

	Learning	**Knowledge Representation**
Petri Nets	From nonexistent to significantly limited (depending on the type of Petri nets).	Transparent representation of knowledge arising as a result of mapping a given problem (specification) onto the net structure. Well-defined semantics of transitions and places.
Fuzzy Petri Nets	Significant learning abilities (parametric optimization of the net connections). Structural optimization can be exercised through a variable number of the transitions utilized in the net.	Transparent representation of knowledge (glass box processing style). Problem (its specification) is mapped directly onto the topology of the fuzzy Petri net. Fuzzy sets deliver an essential feature of continuity required to cope with continuous phenomena encountered in many problems.
Neural Networks	High learning abilities and a vast number of learning algorithms. Learning scheme properties are well-known (advantages and disadvantages).	Limited (black box style of processing) mainly due to the highly distributed topologies of the networks and homogeneous processing elements (neurons) used throughout the overall network. As a result, one has to provide additional interpretation mechanisms for these networks.

8.3 Supervised, unsupervised, and hybrid modes of learning

Supervised and unsupervised models of learning are commonly used taxonomies of pattern classifiers. Classifiers constructed in supervised learning mode exploit knowledge about membership of patterns to classes. In this way, the main task is to design a nonlinear mapping that discriminates between patterns belonging to different categories so that a classification error assumes a minimal value. The other criterion concerns an efficiency of learning/design of the classifier, which has to be high in case of large data sets. Obviously, a Bayesian classifier is a reference point for all constructs. Both neural networks and fuzzy set architectures along with their hybrids [10, 13, 15, 18] are commonly exploited here. Unsupervised learning assumes that we have a collection of patterns that do not carry labels (class assignment) and these need to be discovered in the process of learning or self-organization. Unsupervised learning models include well-known neural network architectures such as self-organizing maps [9] and the family of ART algorithms [4]. Fuzzy sets offer a wide family of FCM clustering methods and have been successfully exploited in a number of different applications [1, 5, 15]. In general, there are numerous clustering algorithms that differ in terms of their underlying optimization criteria, methods of handling data, and a format of results being produced (dendrograms, partition matrices, *etc.*). In particular, we are interested in objective function-based clustering where a structure in the given data set is determined (discovered) by minimizing a certain performance index (objective function). Our anticipation is that the minimized objective function relates with the optimal structure discovered in the analyzed data sets. The form of the objective function (Q) is usually given in advance and does not change during the optimization process. The results of clustering are represented in the form of a partition matrix, $\boldsymbol{U}$. Partition matrices store details regarding class membership of the respective patterns (data). Each row of the matrix corresponds to the respective class (group) identified in the data set. Fuzzy clustering gives rise to the rows that contain membership functions of the fuzzy sets. In the case of clustering we get characteristic functions of sets determined in the entire data set.

While the above taxonomy is sound, the reality of classification problems calls for an array of learning modes. Consider a handful of examples making this point evident and justifying a continuum of models of hybrid supervised-unsupervised learning:

- While most of the patterns in the mixture are not labeled, there is a fraction of selected patterns with labels attached. These patterns can serve as "anchor" points guiding a formation of the overall structure in the data. The situations leading to such mixtures are specific to the problems where there is a substantial cost of classifying the patterns, the job of class assignment is tedious and, finally, the data set is huge. For instance, a collection of handwritten characters falls under this category. The size of the data set is evidently very large (as the characters are readily available). The labeling of all of them is not feasible. Yet a portion of the data set can be manually inspected and the patterns labeled.

- The labeling may not be fully reliable. This could be a result of errors made by the individual making class assignment, complexity of the classification problem, a lack of sharp class boundaries. This yields patterns whose membership to some classes may be questionable and therefore need to be treated as such in the construction of the classifier.

An example of a preprocessing strategy that compensates for the possible imprecision of class labels is fuzzy class label adjustment [15]; using training vectors, robust measures of location and dispersion are computed for each class center. Based on distances from these centers, fuzzy sets are constructed that determine the degree to which each input vector belongs to each class. These membership values are then used to adjust class labels for the training vectors.

The aspect of partial supervision can manifest itself through various ways of utilizing domain knowledge about the problem or incorporating the preferences of the designer as to structure determination cast in a certain setting (context).

In what follows, we discuss two examples of clustering algorithms that fall under this category. The first one, called context-based clustering, is concerned with clustering guided by some context variable (more precisely, a fuzzy set defined therein). The second one deals with clustering in the presence of some auxiliary pattern labeling (where we are concerned with a "reconciliation" of these labels with the structure in the data set itself).

As we will be working with the FCM method and view it as a starting point, it is instructive to set up necessary notation. For the data set $\boldsymbol{X}$, consisting of N records, of dimensionality n, in which we are seeking c clusters (groups), the respective partition matrix is comprised of N columns and c rows. If we are interested in determining Boolean clusters, then the entries of $\boldsymbol{U}$ are 0–1 values. For fuzzy clusters, the entries of the partition matrix assume values in the unit interval; the ik^{th} entry u_{ik} denotes a degree of membership of pattern

k to the cluster i. In addition to the partition matrix, the clustering algorithm returns a set of prototypes– centroids describing the clusters in terms of single representatives.

The optimization problem is governed by the objective function

$$Q = \sum_{j=1}^{c}\sum_{k=1}^{N} u_{ik}^{m} \|x_k - v_i\|^2 \tag{8.22}$$

The partition matrix, $\boldsymbol{U}$, satisfies a number of extra conditions; these properties are captured by restricting the optimization to the class of partition matrices $\boldsymbol{U}$, say, $\boldsymbol{U} \in \mathcal{U}$ and is usually referred to as the Fuzzy c-Means algorithm (FCM) [3] The partition matrix $\boldsymbol{U}$ satisfies a number of extra conditions; these properties are captured by restricting the optimization to the class of partition matrices $\boldsymbol{U}$, say, $\boldsymbol{U} \in \mathcal{U}$,

$$\mathcal{U} = \left\{ u_{ik} \in [0,1] | \sum_{i=1}^{c} u_{ik} = 1 \text{ and } 0 < \sum_{k=1}^{N} u_{ik} < N \right\} \tag{8.23}$$

where $i = 1, 2, \cdots, c$. The other elements of the clustering method are:

- Prototypes (centroids) of the clusters are representatives of the determined groups of elements (data);
- The distance function, $\|\cdot\|$, describes the distance between the individual patterns and the prototypes;
- The fuzzification coefficient, $m > 1$, controls the level of fuzziness assigned to the clusters. By default, we assume that m is equal to 2.

In spite of differences resulting from the way in which the optimization task is formulated at the level of the objective function, the underlying idea remains the same: we intend to reveal a general structure in the data set.

8.3.1 Context-based fuzzy clustering

The essence of context-based clustering [12] is to search for structure within the data while making the search more focused by applying a context. More profoundly, our intent is to concentrate on a search of structure with a certain variable (classification variable) in mind. One should stress that generic clustering is relation-driven, meaning that all variables play the same role in the overall design of the clusters. In the variable of interest, we distinguish a number of so-called contexts. The context itself is an information granule [21,

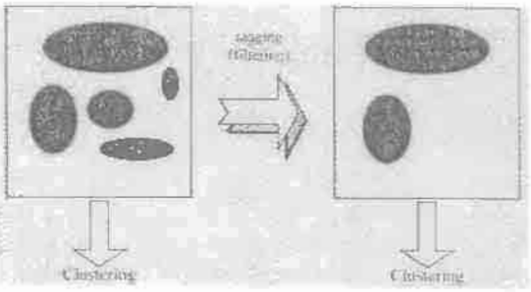


Fig. 8.5 An example of the tagging (logic filtering) effect provided by a linguistic context A; note that some groups of data present in the original data set have been "filtered out" by the use of the fuzzy set in the context variable

22] (captured by fuzzy sets) that is defined in one or several attributes whereby the search is focused for structure in the data. In other words, the formulation

- "Reveal a structure in data $\boldsymbol{X}$",

is reformulated as,

- "Reveal a structure in data $\boldsymbol{X}$ in context A",

where A denotes an information granule of interest (context of clustering). For instance, a task may be formulated as,

- "Reveal a structure in $\boldsymbol{X}$ for context equal to *high* temperature",

with *high* temperature being a fuzzy set defined in the context space. The context space (variable) alludes to the continuous classification variable.

Note that the selected information granule (context) directly impacts upon the resulting data to be examined. As a matter of fact, the context can be regarded as a window (or a focal point) of the clustering pursuit. On a technical side, the introduced linguistic context [21] provides us with a certain tagging of the data. Fig. 8.5 illustrates this phenomenon.

The conditioning aspect (context sensitivity) of the clustering mechanism is introduced into the algorithm by taking into consideration the conditioning variable (context) assuming the values $f_1, f_2, \cdots, f_N$ on the corresponding patterns. More specifically, f_k describes a level of involvement of x_k in the assumed context, $f_k = A(x_k)$. The way in which f_k can be associated with or allocated among the computed membership values of x_k, say, $u_{1k}, u_{2k}, \cdots, u_{ck}$,

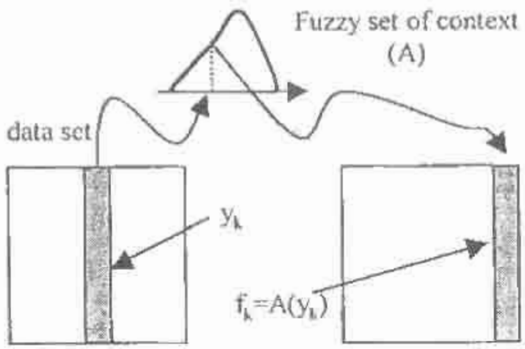


Fig. 8.6 From context fuzzy set to the labeling of data to be clustered (shadowed column in the data set consists of membership values of the context fuzzy set A)

can be realized in the form,

$$\sum_{i=1}^{c} u_{ik} = f_k \tag{8.24}$$

that holds for all data points, $k = 1, 2, \cdots, N$. The way in which the fuzzy set of context, A, is obtained will be discussed in a subsequent section. However, Fig. 8.6 highlights the way we proceed from data to clusters and the corresponding labeling by the membership values of the context variable. It is worth noting that the finite support of A "eliminates" some data points (for which the membership values are equal to zero) thus leaving only a certain subset of the original data to be used for further clustering. Bearing this in mind, we modify the requirements for the partition matrices and define the family,

$$\mathcal{U}(A) = \left\{ u_{ik} \in [0,1] | \sum_{i=1}^{c} u_{ik} = f_k \ \forall k \text{ and } 0 < \sum_{k=1}^{N} u_{ik} < N \ \forall i \right\} \tag{8.25}$$

Note that the standard normalization condition where the membership values sum up to 1 is replaced by the involvement (conditioning) constraint. The optimization problem may now reformulated accordingly,

$$\min_{U, v_1, v_2, \cdots, v_c} Q, \tag{8.26}$$

subject to

$$U \in \mathcal{U}(A). \tag{8.27}$$

Let us proceed with deriving a complete solution to this optimization problem. Essentially, it may be divided into two separate subproblems:

- Optimization of the partition matrix $\boldsymbol{U}$;
- Optimization of the prototypes.

As these tasks may be handled independently of each other, we start with the partition matrix. Moreover, we notice that each column of $\boldsymbol{U}$ can be optimized independently, so let us fix the index of the data point, k. Rather than being subject to (8.27), (8.26) is now subject to the constraint

$$\sum_{i=1}^{c} u_{ik} = f_k. \tag{8.28}$$

In other words, having fixed the data index, we must solve N independent optimization problems. To make the notation more concise, we introduce the notation d_{ik} for the distance between the pattern and prototype,

$$d_{ik}^2 = \|\boldsymbol{x}_k - \boldsymbol{v}_i\|^2. \tag{8.29}$$

As the above is an example of optimization with constraints, we can easily convert this into unconstrained optimization by using the technique of Lagrange multipliers. The overall algorithm is summarized as the following sequence of steps (where the fuzzification parameter m (> 1) affects the form of the membership functions describing the clusters):

Given the number of clusters, c, select the distance function, $\|\cdot\|$, termination criterion, e (> 0), and initialize partition matrix $\boldsymbol{U} \in \mathcal{U}(A)$. Select the value of the fuzzification parameter, m (> 1) (the default is $m = 2.0$).

(1) Calculate centers (prototypes) of the clusters

$$\boldsymbol{v}_i = \sum_{k=1}^{N} u_{ik}^m \boldsymbol{x}_k \Big/ \sum_{k=1}^{N} u_{ik}^m \qquad (i = 1, 2, \cdots, c).$$

(2) Update the partition matrix to

$$u'_{ik} = \frac{f_k}{\displaystyle\sum_{j=1}^{c} \left(\frac{\|\boldsymbol{x}_k - \boldsymbol{v}_i\|}{\|\boldsymbol{x}_k - \boldsymbol{v}_j\|} \right)^{\frac{2}{m-1}}} \quad i = 1, 2, \cdots, c, \; j = 1, 2, \cdots, N$$

(3) Compare $\boldsymbol{U}'$ to $\boldsymbol{U}$. If termination criterion $\|\boldsymbol{U}' - \boldsymbol{U}\| < e$ is satisfied then stop, else return to step (1) with $\boldsymbol{U}$ equal to $\boldsymbol{U}'$.

Result: partition matrix and prototypes.

8.3.2 Clustering with partial supervision: reconciling structural and labeling information

In the simplest case, we envision a situation where there is a set of labeled data (patterns), yet their labeling may not be perfect (as being completed by an imperfect teacher). Bearing this in mind, we want to confront this labeling with the "hidden" structure in the data and reconcile it with this inherent structure. Another interesting situation occurs when an expert provides a granulation of a concept or a variable formulated in terms of fuzzy sets or relations and we intend to confront (reconcile) it with the data. In any case, we anticipate that the linguistic labels should have enough experimental justification behind them and in this way make them more sound. This reconciliation is an example of building constructs on the basis of knowledge (coming here in the form of fuzzy sets) and an array of numeric data. Another version would involve labeling coming from some other clustering mechanisms and presenting results of clustering carried out for the data.

The algorithmic aspects can be laid down in many ways. Incorporation of the labeling information is reflected in the augmented objective function

$$Q = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m}\|x_k - v_i\|^2 + \alpha\sum_{i=1}^{c}\sum_{k=1}^{N}(u_{ik} - f_{ik})^m b_k\|x_k - v_i\|^2 \qquad (8.30)$$

The first term is the objective function, (8.22), used by the original FCM method; the second term reflects the domain knowledge introduced in the form of the given partition matrix $\boldsymbol{F} = ((f_{ik}))$, $i = 1, 2, \ldots, c$, $k = 1, 2, \ldots, N$. The vector $\boldsymbol{b} = [b_k]$ that assumes values in the unit interval, helps control a balance between the structural aspects coming from the clustering itself and the class membership coming from the external source. In addition, the weight, set to 1, excludes the corresponding data point as not being externally labeled. Put it differently: higher entries of $\boldsymbol{b}$ stipulate a higher level of confidence associated to the labeling coming from the external source (say, expert). And conversely, lower entries of $\boldsymbol{b}$ discount the labeling (class assignment) as not being reliable enough. The weight coefficient (α) plays a similar role as the previous vector but it operates at the global level and does not distinguish between the individual patterns.

As a simple illustrative example, consider two-dimensional data shown in

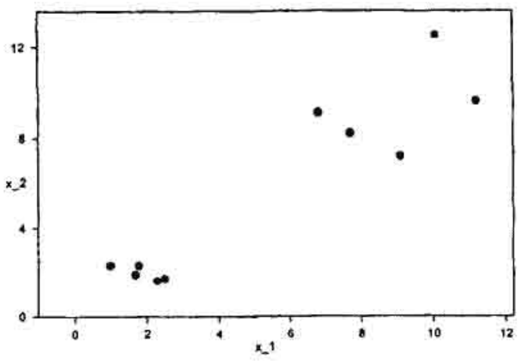


Fig. 8.7 The set of two-dimensional patterns used in the experiment

Fig. 8.7. The auxiliary partition matrix $\boldsymbol{F}$ is shown in Table 8.3 as well. The size of this matrix stipulates the partition of the data into $c = 2$ clusters. Moreover, $\boldsymbol{b} = \mathbf{1}$. So we assume that the extra labeling works equally well with all the patterns. The fuzzification factor, m, was set to 2.

Table 8.3 Two-dimensional data with additional labeling represented by the auxiliary partition matrix $\boldsymbol{F}$.

Pattern	F
(1.00 2.30)	(1.00 0.00)
(1.70 1.90)	(0.90 0.10)
(2.30 1.60)	(0.40 0.60)
(2.50 1.70)	(0.30 0.70)
(1.80 2.30)	(0.80 0.20)
(6.80 9.10)	(0.05 0.95)
(7.70 8.20)	(0.10 0.90)
(9.10 7.20)	(0.00 1.00)
(10.10 12.50)	(0.00 1.00)
(11.20 9.60)	(0.40 0.60)

For different values of α we obtain a set of results reported in terms of the partition matrices (Table 8.4(i)) and the prototypes (Table 8.4(ii)) of the clusters. For $\alpha = 0.0$, there is no impact on the clustering process coming from the auxiliary partition matrix, $\boldsymbol{F}$. In this case, as expected, there are two clearly delineated clusters; note that the entries of the partition matrix are close to 1 or 0. When increasing the values of α, the partition matrix modifies its entries according to the values of $\boldsymbol{F}$. For high values of α (in this case, $\alpha = 6.0$), the impact of $\boldsymbol{F}$ becomes profound and the patterns tend to follow the values in $\boldsymbol{F}$ irrespective of their location in the feature space.

Table 8.4 Partition matrices (i) and prototypes (ii) of clusters for selected values of α.

$\alpha = 0.0$	$\alpha = 0.5$	$\alpha = 1.5$	$\alpha = 6.0$
(i) Partition Matrices			
0.99214 0.00786	0.89927 0.10073	0.82237 0.17764	0.75163 0.24837
0.99964 0.00036	0.95400 0.04600	0.91487 0.08513	0.87906 0.12094
0.99690 0.00310	0.90990 0.09011	0.83770 0.16230	0.76872 0.23128
0.99524 0.00476	0.88993 0.11008	0.80308 0.19692	0.71878 0.28123
0.99896 0.00104	0.90923 0.09077	0.83477 0.16524	0.76308 0.23692
0.06400 0.93600	0.09262 0.90738	0.11661 0.88340	0.14568 0.85432
0.03883 0.96117	0.16894 0.83106	0.27259 0.72741	0.37803 0.62197
0.05055 0.94945	0.19630 0.80370	0.31157 0.68843	0.42259 0.57741
0.05899 0.94101	0.15370 0.84630	0.23210 0.76790	0.30100 0.69900
0.03386 0.96614	0.21306 0.78694	0.35742 0.64258	0.48977 0.51023
(ii) Cluster Prototypes			
1.87844 1.97932	8.99748 9.30629	2.11223 2.20003	8.77845 9.27113
2.57500 2.62513	8.40672 9.06494	3.25580 3.24709	7.84665 8.66043

8.4 Conclusions

We have presented a motivation for the use of fuzzy sets as a logic canvas for pattern recognition problems possessing imprecise or vague information. Fuzzy set-based logic processors and Petri nets are natural extensions of their crisp binary counterparts. The former carry much stronger logic connotations than architectures such as radial basis function neural networks. The latter possess learning characteristics similar to neural networks without a reduction in the knowledge representation capabilities of the binary Petri net. Finally, context-based partially supervised extensions to the fuzzy c-means clustering algorithm were also presented. By adjusting the optimization function to take into account a fuzzy set-based context, confidence (as determined by an external source such as a domain expert) of assigned class labels may be exploited to more accurately determine the intrinsic structure of data.

Acknowledgment

Support from the Natural Science and Engineering Research Council of Canada (NSERC) is gratefully acknowledged.

References

[1] E. Backer, *Computer Assisted Reasoning in Cluster Analysis.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[2] R. E. Bellman, R. Kalaba, and L. A. Zadeh, "Abstraction and pattern recognition," *Journal of Mathematical Analysis and Applications*, vol. 13, pp. 1–7, 1966.

[3] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms.* New York: Plenum Press, 1981.

[4] S. Grossberg, "Adaptive pattern classification and universal recoding 1. parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121–134, 1976.

[5] F. Hoppner, F. Klawonn, R. Kruse, and T. Runkler, *Fuzzy Cluster Analysis.* New York: Wiley, 1999.

[6] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* London: Wiley, 1988.

[7] A. Kandel, *Fuzzy Mathematical Techniques with Applications.* New York: Addison-Wesley, 1986.

[8] G. J. Klir and T. A. Folger, *Fuzzy sets, Uncertainty, and Information.* Englewood Cliffs: Prentice Hall, 1988.

[9] T. Kohonen, *Self-Organization and Associative Memory.* New York: Springer-Verlag, 1989.

[10] S. K. Pal and S. Mitra, *Neuro-Fuzzy Pattern Recognition*. New York: Wiley, 1999.

[11] W. Pedrycz and A. Rocha, "Fuzzy-set based models of neurons and knowledge-based networks," *IEEE Transactions on Fuzzy Systems*, vol. 1, pp. 254–266, 1993.

[12] W. Pedrycz, "Conditional fuzzy c-means," *Pattern Recognition Letters*, vol. 17, pp. 625–632, 1996.

[13] W. Pedrycz, "Conditional fuzzy clustering in the design of radial basis function neural networks," *IEEE Transactions on Neural Networks*, vol. 9, pp. 601–612, 1998.

[14] W. Pedrycz and F. Gomide, *An Introduction to Fuzzy Sets*. Cambridge: MIT Press, 1998.

[15] N. Pizzi, "Bleeding predisposition assessments in tonsillectomy/adenoid/-ectomy patients using fuzzy interquartile encoded neural networks." Artif. Intell. Med. (in press).

[16] B. Ripley, *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press, 1996.

[17] C. Shannon, "A symbolic analysis of relay and switching circuits," *Transactions of the American Institute of Electrical Engineers*, vol. 57, pp. 713–723, 1938.

[18] H. Takagi and I. Hayashi, "NN-driven fuzzy reasoning," *International Journal of Approximate Reasoning*, vol. 5, pp. 191–212, 1991.

[19] T. Terano, K. Asai, and M. Sugeno, *Fuzzy Systems Theory and Its Applications*. Boston: Academic Press, 1992.

[20] L. Zadeh, K. Fu, K. Tanaka, and M. Shimura, *Fuzzy Sets and Their Applications to Cognitive and Decision Processes*. London: Academic Press, 1975.

[21] L. A. Zadeh, "Fuzzy sets and information granularity," in *Advances in Fuzzy Set Theory and Applications* (M. Gupta, R. K. Ragade, and R. R. Yager, eds.), pp. 3–18, Amsterdam: North Holland, 1979.

[22] L. A. Zadeh, "Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic," *Fuzzy Sets and Systems*, vol. 90, pp. 111–117, 1997.

[23] H. J. Zimmermann, *Fuzzy Set Theory and Its Applications*. Boston: Kluwer, 1991.

Chapter 9

FUZZY PATTERN RECOGNITION BY FUZZY INTEGRALS AND FUZZY RULES

M. Grabisch

Laboratoire d'Informatique de Paris VI
UPMC
8, rue du Capitaine Scott
75015 Paris, FRANCE
e-mail: *michel.grabisch@lip6.fr*

Abstract

We give an overview of the application of fuzzy rules and fuzzy integrals in classification, presenting the general methodology and illustrating it by giving real applications. Fuzzy rules have been most of the time devoted to fuzzy control, using the so-called "Mamdani rules". Here we present a broader view of the topic in the framework of possibility theory. It is seen that uncertainty rules are the best-suited ones for modeling human knowledge in pattern recognition. On the other hand, fuzzy integrals, by assigning weights to groups of attributes, permit to define non linear classifiers, with powerful performances.

9.1 Introduction

Since the introduction by Zadeh of the concept of fuzzy sets in 1965 [35], and later of the concept of possibility measure in 1978 [36], that is, a non-additive representation of uncertainty, many methods in pattern recognition have been proposed, based on these two seminal ideas. We should cite in this respect the pioneering work of Ruspini [30], who gave the basis of fuzzy clustering in 1969, an area which has known an important development later due to the work of Bezdek [3].

It would be a difficult task to cite here all important contributions to the field of pattern recognition done by researchers in fuzzy logic (but the reader can consult the edited volume of Bezdek and Pal [4] for a selected collection of papers on this topic), and we will limit ourself in this chapter to the presentation of only two methodologies, which are representative of the two seminal ideas of Zadeh cited above. The first one is based on fuzzy sets, as a means to model imprecise concepts, and uses fuzzy rules to represent the classes of interest. The second one is based on non-additive representations of uncertainty, namely fuzzy measures [32], which are yet more general than possibility (and also probability) measures.

The approaches described here are based on the work done by the author. Related works will be also indicated. In the sequel, $\wedge, \vee$ denote min and max respectively.

9.2 Classification by fuzzy rules

9.2.1 Fuzzy rules

The following exposition is based on possibility theory (see [8] for basic definitions). The typology of fuzzy rules we use below is based on [9].

9.2.1.1 *Fuzzy logic implications*

We call multivalued implication any operation I from $[0,1]$ to $[0,1]$ which extends the material implication of classical binary logic, *i.e.*,

$$I(0,0) = 1 \;\; I(0,1) = 1 \tag{9.1}$$

$$I(1,0) = 0 \;\; I(1,1) = 1. \tag{9.2}$$

The material implication can be written as

$$I(a,b) = \neg a \vee b. \tag{9.3}$$

The first way to extend I on $[0,1]$ is to replace usual binary operations by their corresponding fuzzy operations, *i.e.*,

$$I(a,b) = n(a) \perp b$$

where $\perp$ is a t-conorm and n a strong negation (*i.e.*, strictly decreasing, involutive, and such that $n(0) = 1$). These implications are called S-implications since S often denotes t-conorms. Examples of S-implications are those of Kleene-Dienes, Reichenbach and Łukasiewicz, defined by

$$I_{\mathrm{KD}}(a,b) = (1-a) \vee b \tag{9.4}$$

$$I_{\mathrm{R}}(a,b) = 1 - a + ab \tag{9.5}$$

$$I_{\text{Ł}}(a,b) = (1 - a + b) \wedge 1. \tag{9.6}$$

The second way of considering an implication is related to sets. We say that $p \to q$ if in all worlds where p is true, then q is true, *i.e.*,

$$P \subset Q$$

where P (resp. Q) denotes the set of worlds where p (resp. q) is true. But inclusion is a particular order relation, and in lattice theory, residuated operations are defined for lattices having a structure of semi-group. Using a t-norm $\top$ for the semi-group operation, we can define

$$I(a,b) = \sup\{c \in [0,1] | a \top c \leq b\}.$$

These implications are called residuated implications or R-implications. Examples of R-implications are those of Łukasiewicz and Gödel, the last one being defined by

$$I_{\mathrm{G}}(a,b) = \begin{cases} 1, & \text{if } a \leq b \\ b, & \text{otherwise.} \end{cases}$$

Using these implications, two types of rules can be defined [9]:

Uncertainty rules (or *uncertainty-qualifying rules*) – They have the following semantical interpretation:

The more X is A, the more certain Y is B

i.e., $\mu_A(x) \leq C(B)$, where $C(B)$ is the certainty degree of having B when the input is x, and μ_A is the membership function of the fuzzy set A. It can be shown that S-implications are suitable for this case.

Gradual rules (see [10]) (or *truth-qualifying rules*) – They have semantically the following meaning:

`The more` x `is` A`, the more` y `is` B

where A, B are fuzzy sets with membership functions μ_A, μ_B. This kind of rule expresses a relation between x and y in the sense that as X goes closer to A, Y is constrained to go closer to B. This constraint can be expressed by $\mu_A(x) \leq \mu_B(y)$. The least specific solution (*i.e.*, leading to the greatest possibility distribution) is the Gaine-Rescher implication defined by

$$I_{\mathrm{GR}}(a,b) = \begin{cases} 1, & \text{if } a \leq b \\ 0, & \text{otherwise} \end{cases}$$

since, if (x, y) is such that $A(x) \leq B(y)$, then $I(A(x), B(y)) = 1$ and 0 otherwise. In fact, all R-implications are suitable for modeling gradual rules.

9.2.2 The generalized modus ponens

The classical modus ponens is the following reasoning process:

$p \rightarrow q$ (rule)
p (fact)

---

q (conclusion)

The extension to fuzzy sets becomes:

$X \in A \rightarrow Y \in B$ (rule)
$X \in A'$ (fact)

$Y \in B'$ (conclusion)

The conclusion B' is computed as follows. We consider the rule as a conditional possibility distribution $\pi_{Y|X}(x, y) = I(A(x), B(y))$, and the possibility distribution of y after inference is obtained by the combination/projection principle [9]

$$\pi_Y(y) = \mu_{B'}(y) = \sup_x(\mu_{A'}(x) \wedge \pi_{Y|X}(x, y)).$$

In the case of a precise input x_0 (*i.e.*, a distribution reduced to a single value), the above expression reduces to:

$$\pi_Y(y) = I(\mu_A(x_0), \mu_B(y)).$$

Let us now consider the case of several parallel rules $R_1, \ldots, R_l$. Each rule R_j is expressed by a possibility distribution $\pi^j_{X|Y}(x, y)$. It is known from [9] that inferred possibility distributions have to be aggregated by a minimum operator. Moreover, it is better (*i.e.*, more informative) to aggregate all the rules in one single rule before performing the inference, than aggregating the inferred result of each rule, thus:

$$\sup_x \left(\pi_X(x) \wedge \left[\bigwedge_j \pi^j_{Y|X}(x,y) \right] \right) \leq \bigwedge_j \sup_x (\pi_X(x) \wedge \pi^j_{Y|X}(x,y)).$$

9.2.3 Classification by fuzzy rules

Let $C_1, \ldots, C_m$ be a set of classes, which can be described by a set of features or attributes X_i, $i = 1, \ldots, n$, *i.e.*, a given pattern to classify is an element $x = (x_1, \ldots, x_n)$ of $X_1 \times \cdots \times X_n$, where x_i is the value taken by attribute i for this pattern. In the sequel, X_i will indicate either the attribute (or variable) itself or its set of values, while x_i indicate possible values of X_i. Fuzzy rules can be used in two different respects:

- gradual rules express relations between variables, and can be used to compute approximately the value of an attribute which could be difficult or costly to obtain or compute in an exact mathematical way. Such situations often occur, when it is sufficient to have a rough approximation of the value of an attribute for classification. More specifically, let X_j be such an attribute, whose value depends on variables $Z_1, \ldots, Z_p$, which could be attributes among $X_1, \ldots, X_n$, or additional variables. Instead of giving an explicit model of the form $x_j = f(z_1, \ldots, z_n)$, with f being a deterministic function, the model is given under a set of gradual rules of the form:

  The more Z_1 is A_1 and ... and the more Z_p is A_p,
  the more X_j is B_j

  where $A_i, \ldots, A_p, B_j$ are fuzzy sets on the respective universes. This kind of rule expresses a constraint between X_j and $Z_1, \ldots, Z_p$: as

$Z_1, \ldots, Z_p$ go nearer the core of the fuzzy sets $A_1, \ldots, A_p$, the value of X_j is constrained to go nearer the core of B_j. A gradual equivalence (*i.e.*, a gradual rule $p \longrightarrow q$ together with its converse $\neg p \longrightarrow \neg q$) can be used as well if necessary. In this case, there will be similar constraints on the negations of B_j.
Note that the result of a gradual rule is a fuzzy set, even if the input is crisp. Depending on the application, one should decide to keep the result in the form of a fuzzy set, and to forward it in subsequent layers of the reasoning, or to extract from it the most *representative* or *plausible* value. We avoid here the term "defuzzification", which is so often associated with the center of gravity method. If such a method can be justified in the domain of fuzzy control (where fuzzy rules are Mamdani-type rules, not belonging to the taxonomy given in Section 9.2.1.1), it has no meaning in the context of possibility theory, where the representative value should be of maximal degree of possibility.
See an example of application of gradual rules in classification in [2].

- uncertainty rules express relations between variables and classes. More specifically, let us introduce a new variable Y defined on the set of classes $\{C_1, \ldots, C_m\}$, which we can call the *classification variable*. A classification rule will have the form:

  The more X_1 is A_1, and ... and the more X_n is A_n,
  the more certain Y is B

  where $A_1, \ldots, A_n$ are fuzzy sets expressing fuzzy domains of values for attributes, and B is the possibility distribution on classes, *i.e.*, $B(C_i)$ expresses to which degree it is possible that C_i is the right class when the pattern matches perfectly the fuzzy sets $A_1, \ldots, A_n$.
  In practice, several such rules will be used in parallel. It is to be noted that, since the aggregation of rules is made in a conjunctive way, if one of the rules concludes that the possibility degree of class C_i is 0, then no other rule can "save" this conclusion, *i.e.*, the class C_i will be considered as impossible.

So basically, the classification is performed by a set of uncertainty rules, with possibly some additional gradual rules to compute some attributes. The result of the inference is thus a possibility distribution π_Y indicating the possibility degree for each class. If $\pi_Y(C_i) = 0$, then the rule has inferred that the observed object cannot belong to class C_i. If $\pi_Y(C_i) = 1$, then it is totally possible (but not certain) that the object belongs to class C_i. The question is now how to exploit this possibility distribution in order to draw a conclusion on the class of the pattern in consideration.

Recall that a possibility distribution π induces a possibility measure Π defined by $\Pi(A) = \sup_{x \in A} \pi(x)$ and a necessity measure $N(A) = 1 - \Pi(\overline{A})$. Then:

- if there is a unique class C_i such that $\pi_Y(C_i) = 1$, class C_i is the most certain class since $N(C_i) > 0$ and $N(C_j) = 0$ for all $j \neq i$.
- if several C_i are such that $\pi_Y(C_i) = 1$, we cannot decide between these classes, which are the most certain: the rule base has not enough knowledge to discriminate. However, if the classes with possibility degree equal to 1 correspond to a meaningful subset of classes, then the certainty of this subset is strictly positive. If $\pi_Y(C_i) = 1$ everywhere, then there is total uncertainty about the class.
- if there is no class C_i such that $\pi_Y(C_i) = 1$, then no class matches the pattern perfectly. It means that the pattern could belong to a class which is not included into the initial set of classes. It could also mean that there is some contradiction in the rule base.

Further processing can be done, but it depends strongly on the application: see the example below with satellite images.

It is important to stress here the advantage of possibility theory over probabilistic methods for the representation of uncertainty. The forced normalization of probability measures prevents them from distinguishing between equicertainty and low certainty due to the presence of an unknown class.

This methodology has been successfully applied by the author on target classification [2], face recognition [13], and in satellite image analysis (unpublished). It is to be noted that in every case, no automatic learning method was used to derive the rules, but only expert knowledge. The following example, borrowed from the satellite image analysis application, illustrates this. The problem is concerned with the recognition of various natural and artificial zones on an image (decametric image). The expert in the field is able to tell the following:

- *water appears as regions of almost uniform texture. The grey level can be black (pure water) to middle grey (turbid water)*

- *forests form regions which are slightly more textured than water, and are generally dark (but less dark than pure water)*
- *fields on a small scale present almost no texture. The grey level varies from dark to light grey*
- *urban zones are highly textured and are bright.*

From this expertise, the following attributes seem to be relevant:

- mean of grey levels on a 3×3 window, denoted `Moy` ;
- variance of the grey levels on a 3×3 window, denoted `Var` ;
- "busyness" type II on a 3×3 window, denoted `Busy2`.

The busyness index has been proposed by Dondes and Rosenfeld [7, 29]. The type II busyness is defined as follows. Considering a 3×3 window:

a	b	c
d	e	f
g	h	i

we compute the 12 absolute differences $\delta_{ab}, \delta_{bc}, \delta_{de}, \delta_{ef}, \delta_{gh}, \delta_{hi}, \delta_{ad}, \delta_{dg}, \delta_{be}, \delta_{eh}, \delta_{cf}, \delta_{fi}$, avec $\delta_{ab} = |a - b|$, and similarly for the other quantities. The type II busyness index corresponds to the median of these 12 differences:

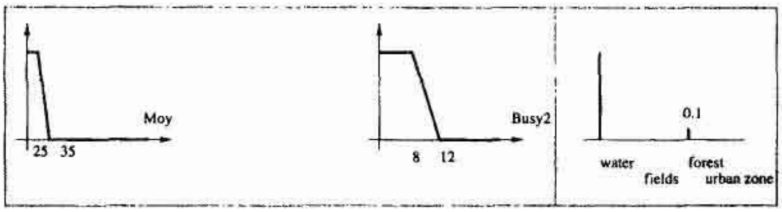
$$B_2 = \mathrm{med}(\delta_{ab}, \delta_{bc}, \delta_{de}, \delta_{ef}, \delta_{gh}, \delta_{hi}, \delta_{ad}, \delta_{dg}, \delta_{be}, \delta_{eh}, \delta_{cf}, \delta_{fi}).$$

Using these three attributes, we have built a fuzzy rule base, containing 7 fuzzy rules, all of the uncertainty type, with Kleene-Dienes implication. We give some examples of this rule base.

- **Fuzzy rule 1**

```
IF (Moy is VERY DARK) AND (Busy2 is LOW),
THEN it is almost surely WATER
```

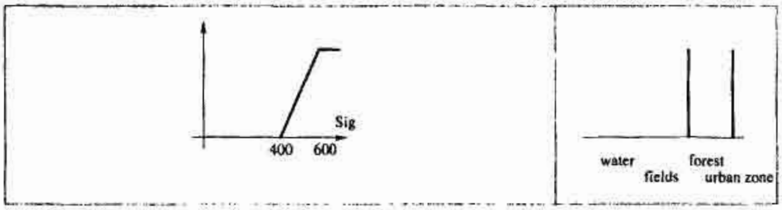
The fuzzy sets concerning this rule are shown below. The possibility distribution over the classes is represented by vertical strokes.



- **Fuzzy rule 5**

  ```
  IF (Sig is HIGH),
  THEN it can be neither FIELD nor WATER
  ```

  The fuzzy sets and possibility distribution concerning this rule are shown below.



As explained above, the result of inference is a possibility distribution over the 4 classes. The final decision is taken as follows. For each pixel p, we first assign the class of highest possibility degree. Then, we compute the difference $\Delta_{12}(p)$ between the class of maximal possibility degree and the second highest possibility degree. This gives the quality of the decision (the higher $\Delta_{12}(p)$ is, the better the quality), which is 0 if the 2 best classes have equal possibility degree. Considering a window around p, we compute the average of decisions in the window, weighted by the quality of the decision. If the average is 0 for each class, then the pixel is classified into the INDETERMINATE class.

Experimental results on various images show that this method, although without learning procedure, largely outperforms classical approaches such as neural nets, clustering and nearest prototype, and also fuzzy integrals, presented hereafter.

9.2.4 Other approaches based on fuzzy rules

Many authors have proposed fuzzy rule based approaches for classification, but to the knowledge of the author, all these approaches are more or less based on Mamdani-type rule or similar (*e.g.*, Sugeno rules), *i.e.*, rules where the implication is, in fact, a conjunction, namely the minimum operator for Mamdani's rules, and the product operator for Sugeno's rule. For this reason, let us call them "conjunctive rules". This type of rule, although widely used, cannot be called properly a rule in the logical sense of the term, but it performs a smooth interpolation between fuzzy domains. In this respect, conjunctive rules can be viewed as a means to "fill in" the holes in a partial description of the relation between attributes and classes, given under the form of (fuzzy) examples. For this reason, these methods often lack the concise form of the rule base which can be obtained with the previous approach: for example, the rule base in [2] contains only 19 rules for a 4-class and 16-attribute problem, and the above example uses 7 rules. On the other hand, some methods based on conjunctive rules use nearly as many rules as learning samples in the database!

In fact, most of the work devoted to the learning of fuzzy rules is concerned rather with the modeling of a continuous function (*e.g.*, for prediction) than classification (see the monograph of Glorennec [14] for a thorough survey of this topic), and are not completely appropriate. Few methods are available for constructing fuzzy rules from learning data that include the structure of the fuzzy rule base. Such a complete method, including a detailed analysis of performance, has been proposed by Mandal *et al.* [24, 25].

9.3 Classification by fuzzy integrals

9.3.1 Background on fuzzy measures and integrals

Let X be a finite index set $X = \{1, \ldots, n\}$.

Definition 9.1 A fuzzy measure μ defined on X is a set function $\mu : \mathcal{P}(X) \longrightarrow [0,1]$ satisfying the following axioms:

(i) $\mu(\emptyset) = 0,\ \mu(X) = 1.$
(ii) $A \subseteq B \Rightarrow \mu(A) \leq \mu(B)$

$\mathcal{P}(X)$ indicates the power set of X, *i.e.*, the set of all subsets of X. ■

A fuzzy measure on X needs 2^n coefficients to be defined, which are the values of μ for all the different subsets of X.

Fuzzy integrals are integrals of a real function with respect to a fuzzy measure, by analogy with Lebesgue integral which is defined with respect to an ordinary (*i.e.*, additive) measure. There are several definitions of fuzzy integrals, among which the most representative ones are those of Sugeno [32] and Choquet [6].

Definition 9.2 Let μ be a fuzzy measure on X. The discrete Choquet integral of a function $f : X \longrightarrow \mathbb{R}^+$ with respect to μ is defined by

$$\mathcal{C}_\mu(f(x_1), \ldots, f(x_n)) := \sum_{i=1}^{n}(f(x_{(i)}) - f(x_{(i-1)}))\mu(A_{(i)}), \tag{9.7}$$

where $\cdot_{(i)}$ indicates that the indices have been permuted so that $0 \leq f(x_{(1)}) \leq \cdots \leq f(x_{(n)}) \leq 1$. Also $A_{(i)} := \{x_{(i)}, \ldots, x_{(n)}\}$, and $f(x_{(0)}) = 0$. ■

Definition 9.3 The discrete Sugeno integral of a function $f : X \longrightarrow [0, 1]$ with respect to μ is defined by

$$\mathcal{S}_\mu(f(x_1), \ldots, f(x_n)) := \bigvee_{i=1}^{n}(f(x_{(i)}) \wedge \mu(A_{(i)})), \tag{9.8}$$

with the same notation as before. ■

Choquet integral coincides with Lebesgue integral when the measure is additive, but this is not the case for the Sugeno integral.

9.3.2 Classification by fuzzy integral

9.3.2.1 *General methodology*

As before, let $C_1, \ldots, C_m$ be a set of given classes, and let patterns be described by a n-dimensional vector $x^T = [x_1 \cdots x_n]$. We have n sensors (or sources), one for each feature (attribute), which provide for an unknown sample x° a degree of confidence in the statement "x° belongs to class C_j", for all C_j. We denote by $\phi_i^j(x^\circ)$ the confidence degree delivered by source i (*i.e.*, feature i) of x° belonging to C_j.

The second step is then to combine all the partial confidence degrees in a consensus-like manner, by a fuzzy integral. It can be shown that fuzzy integrals constitute a vast family of aggregation operators including many widely used operators (minimum, maximum, order statistic, weighted sum, ordered weighted sum, *etc.*) suitable for this kind of aggregation [15]. In particular, fuzzy integrals are able to model some kind of interaction between features: this is the main motivation of the methodology (more on this in Section 9.3.4). Thus the global confidence degree in the statement "x° belongs to C_j" is given by:

$$\Phi_{\mu^j}(C_j; x^\circ) := \mathcal{C}_{\mu^j}(\phi_1^j(x^\circ), \ldots, \phi_n^j(x^\circ)) \tag{9.9}$$

(or similarly with the Sugeno integral). Finally, x° is put into the class of highest confidence degree. Here, the fuzzy measures μ^j (one per class) are defined on the set of attributes (or sensors), and express the importance of the sensors and groups of sensors for the classification. For example, $\mu^j(\{X_1\})$ expresses the relative importance of attribute 1 for distinguishing class j from the others, while $\mu^j(\{X_1, X_2\})$ expresses the relative importance of attributes 1 *and* 2 *taken together* for the same task. The precise way of how to interpret this will be given in Section 9.3.4.

The above presentation is very general and allows many methods to be used. However, it is interesting to embed this methodology in the fuzzy pattern matching methodology, a fundamental approach for classification in possibility theory, which is the counterpart of the Bayesian approach in probability theory. Due to its importance, we devote the next paragraph to the presentation of this methodology, its connection with fuzzy integral, and the Bayesian approach.

9.3.2.2 *The fuzzy pattern matching approach*

As for the section on fuzzy rules, we assume some familiarity of the reader with possibility theory (see [8] for this topic, and [12] for fuzzy pattern matching). Following previous notation, X_i denotes the universe of the ith attribute. Each class C_j is modeled by a collection of fuzzy sets $C_j^1, \ldots, C_j^n$ defined on $X_1, \ldots, X_n$ respectively, expressing the set of typical values taken by the attribute for the considered class. An observed datum x is modeled by a possibility distribution $\pi_x(x_1, \ldots, x_n)$, representing the distribution of possible locations of the (unknown) true value of x in $\times_{i=1}^n X_i$. If attributes are considered to be non-interactive, then $\pi_x(x_1, \ldots, x_n) = \wedge_{i=1}^n \pi_i(x_i)$. Now the possibility and necessity degrees that datum x matches class C_j with respect to attribute i is given by

$$\Pi_{\pi_i}(C_j^i) := \sup_{x_i \in X_i} (C_j^i(x_i) \wedge \pi_i(x_i))$$
$$N_{\pi_i}(C_j^i) := \inf_{x_i \in X_i} (C_j^i(x_i) \vee (1 - \pi_i(x_i))).$$

The first quantity represents the degree of overlapping between typical values of the class and possible value of the datum, while the second one is an inclusion degree of the set of possible values of x_i into C_j^i. If x is a precise datum, π_x

reduces to a point, and the two above quantities collapse into $C_j^i(x_i)$, which corresponds to $\phi_i^j(x^\circ)$.

The next step is the aggregation of these matching degrees, according to the way the class C_j is built. If, for example, the class is built by the conjunction of the attributes, *i.e.*, $x \in C_j$ *if* $(x_1 \in C_j^1)$ *and* $(x_2 \in C_j^2)$ *and* $\cdots$ *and* $(x_n \in C_j^n)$, then it can be shown that, by letting $C_j := C_j^1 \times \cdots \times C_j^n$,

$$\Pi_\pi(C_j) = \bigwedge_{i=1}^{n} \Pi_{\pi_i}(C_j^i)$$
$$N_\pi(C_j) = \bigwedge_{i=1}^{n} N_{\pi_i}(C_j^i).$$

Similarly, if the class is built by a disjunction of the attributes, a weighted conjunction, or a weighted disjunction, the above result still holds, replacing the minimum by a maximum, a weighted minimum or a weighted maximum respectively. More generally, if we consider that C_j is built by a Sugeno integral with respect to a given fuzzy measure μ, a construction which encompasses all previous cases*, $\Pi_\pi(C_j)$ and $N_\pi(C_j)$ are also obtained by the (same) Sugeno integral. More specifically:

Proposition 1 *Let μ be a fuzzy measure, and consider that class C is expressed by a Sugeno integral, i.e., $C(x_1, \ldots, x_n) = \bigvee_{i=1}^{n} \left[C^{(i)}(x_{(i)}) \wedge \mu(A_{(i)})\right]$. Then, the possibility and necessity degrees that a datum x belongs to class C_j are given by*

$$\Pi_\pi(C) = \mathcal{S}_\mu(\Pi_{\pi_1}(C^1), \ldots, \Pi_{\pi_n}(C^n))$$
$$N_\pi(C) = \mathcal{S}_\mu(N_{\pi_1}(C^1), \ldots, N_{\pi_n}(C^n)).$$

For a proof, see [18].

This method can be viewed also under the Bayesian point of view. Let $p(x|C_j)$, $j = 1, \ldots, m$ be the probability densities of classes, and $p(x_i|C_j)$, $i = 1, \ldots, n$, $j = 1, \ldots, m$, the marginal densities of each attribute. The Bayesian inference approach is to minimize the risk (or some error cost function), which amounts, in the case of standard costs, to assigning x to the class maximizing

*Sugeno integrals, as shown by Marichal [26], represent a wide class of Boolean polynomials, *i.e.*, polynomials formed uniquely by a combination of $\vee$ and $\wedge$.

the following discriminating function:

$$\Phi(C_j|x) = p(x|C_j)P(C_j).$$

where $P(C_j)$ is the *a priori* probability of class C_j. If the attributes are statistically independent, the above formula becomes :

$$\Phi(C_j|x) = \prod_{i=1}^{n} p(x_i|C_j)P(C_j). \tag{9.10}$$

If the classes have equal a priori probability, formulae (9.8) and (9.10) are similar: in probability theory and in the case of independence, the product operator takes place of the aggregation operator.

9.3.3 Learning of fuzzy measures

We give now some insights on the identification of the fusion operator, that is, the fuzzy integral, using training data. We suppose that the ϕ_i^j have already been obtained by some parametric or non-parametric classical probability density estimation method, after suitable normalization: possibilistic histograms (that is, transformations of probability density functions into possibility distributions, see [11]), Parzen windows, Gaussian densities, *etc.*

The identification of the fusion operator reduces to the identification (or learning) of the fuzzy measures μ^j, that is, $m(2^n - 2)$ coefficients. Several approaches have been tried here, corresponding to different criteria. We restrict ourselves to the most interesting, and state them in the two-class case ($m = 2$) for the sake of simplicity. We suppose to have $l = l_1 + l_2$ training samples labeled $x_1^j, x_2^j, \ldots, x_{l_j}^j$ for class C_j, $j = 1, 2$. The criteria are the following:

- the squared error (or quadratic) criterion, *i.e.*, minimize the quadratic error between expected output and actual output of the classifier. This takes the following form :

$$J = \sum_{k=1}^{l_1} (\Phi_{\mu^1}(C_1; x_k^1) - \Phi_{\mu^2}(C_2; x_k^1) - 1)^2 + \sum_{k=1}^{l_2} (\Phi_{\mu^2}(C_2; x_k^2) - \Phi_{\mu^1}(C_1; x_k^2) - 1)^2.$$

  It can be shown that this reduces to a quadratic program with $2(2^n - 2)$ variables and $2n(2^{n-1} - 1)$ constraints in the case of Choquet integral (see full details in [19, 20]).

- the generalized quadratic criterion, which is obtained by replacing the term $\Phi_{\mu^1} - \Phi_{\mu^2}$ by $\Psi[\Phi_{\mu^1} - \Phi_{\mu^2}]$ in the above, Ψ being any increasing function from $[-1,1]$ to $[-1,1]$. Ψ is typically a sigmoid type function $\Psi(t) = (1 - e^{-Kt})/(1 + e^{-Kt})$, $K > 0$. With suitable values of K, differences between good and bad classifications are enhanced. This is no more a quadratic program, but a constrained least mean squares problem, which can also be solved with standard optimization algorithms when the Choquet integral is used. In fact, this optimization problem requires huge memory and CPU time to be solved, and happens to be rather ill-conditioned since the matrix of constraints is sparse. For these reasons, the author has looked towards heuristic algorithms better adapted to the peculiar structure of the problem and less greedy [16]. A satisfying algorithm has been found (called hereafter HLMS) which, although suboptimal, reduces the computing time by a factor of 200.

9.3.3.1 *Performance*

We give some experimental results of classification performed on real and simulated data. We have tested the Choquet integral with the quadratic criterion minimized with the Lemke method (QUAD), the generalized quadratic criterion minimized by a constrained least square algorithms (CLMS), and by our algorithm (HLMS), and compared with classical methods. Table 9.1 (top) gives the results obtained on the Iris data of Fisher (3 classes, 4 attributes, 150 data), and on the cancer data (2 classes, 9 attributes, 286 data), which are highly non-Gaussian. The results by classical methods come from a paper of Weiss and Kapouleas [34]. The good performance of HLMS on the difficult cancer data is to be noted.

The bottom part of the table gives another series of results, obtained on simulated data (3 classes, 4 non-Gaussian attributes, 9000 data, one attribute is the sum of two others). These results show that if the Choquet integral-based classifier is not always the best one, it is nevertheless always among the best ones.

In [27], an experiment has been conducted on a problem of bank customer segmentation (classification). In the first step, we have performed a classification on a file of 3068 customers, described by 12 qualitative attributes, and shared among 7 classes. Classical methods in this context are linear regression,

Table 9.1 Classification rates on various data sets

Method	iris (%)	cancer (%)
linear	98.0	70.6
quadratic	97.3	65.6
nearest neighbor	96.0	65.3
Bayes independent	93.3	71.8
Bayes quadratic	84.0	65.6
neural net	96.7	71.5
PVM rule	96.0	77.1
QUAD	96.7	68.5
CLMS	96.0	72.9
HLMS	95.3	77.4

Method	Classification rate (%)
Bayes linear	82.6
linear pseudo-inverse	84.4
cluster	86.9
adaptive nearest neighbor	87.8
Bayes quadratic	90.3
k nearest neighbor	90.4
tree	96.2
CLMS	90.7
HLMS	89.2

sequential scores, and polytomic scores. The problem happened to be very difficult, since no method (including fuzzy integrals), was able to go beyond 50% of correct classification (see Table 9.2, top). However, in many cases, the quantities $\Phi_{\mu^j}(C_j;x)$ were very near for two classes, showing that the decision of the classifier was not clear-cut. In the second step, we have taken into account the "second choice", considering that the classification was also correct when the second choice gave the correct class, provided the gap between the two greatest $\Phi_{\mu^j}(C_j;x)$ was below some threshold (here 0.05). Performing this way, the classification rate climbed to 65%. We have tried to apply the

Table 9.2 Segmentation of customers

File 1

Methods	Classification rate
Regression	45 %
Sequential scores	46.4 %
Fuzzy integral (HLMS)	47.1 %
Polytomic scores	50 %
Polytomic scores (2nd choice)	54%
Fuzzy integral (HLMS) (2nd choice)	65 %

File 2

Methods	Classification rate
Regression	29.9 %
Sequential scores	27.9 %
Fuzzy integral (HLMS)	31.1 %
Polytomic scores	32.2 %
Polytomic scores (2nd choix)	36%
Fuzzy integral (HLMS) (2nd choix)	50 %

same approach to classical methods, but without good results, since there were very few cases where first and second choices were very near. Even taking systematically the two first choices, the rate obtained was at best 54%. A second experiment was performed on a second file of 3123 customers, described by 8 qualitative attributes, and shared among 7 classes. The results corroborate the fact that the classifier based on the fuzzy integral, when allowing the second choice in case of doubt, largely outperforms the other methods.

This fact is again an evidence of the advantage of dealing with possibility distributions[†] rather than probability distributions as explained in Section 9.2.3.

[†]Strictly speaking, $\Phi_{\mu^j}(C_j;x)$ is a possibility distribution only if a Sugeno integral is used. Nevertheless, there is no normalization of Φ such as $\sum_j \Phi_{\mu^j}(C_j;x) = 1$.

9.3.4 Importance and interaction of attributes

As said before, the fuzzy measures μ^j contain all the information about the importance of all individual attributes (or features) and all groups of attributes for distinguishing class C_j from the others. Let us drop index j for simplicity, and denote by $X = \{1, \ldots, n\}$ the set of attributes, and by $X_1, \ldots, X_n$ the corresponding axes. We can reasonably state the following.

- a feature i is important if the values of $\mu(A)$ are high whenever $i \in A$. Clearly, it is not enough to look solely at the value of $\mu(\{i\})$, but also at $\mu(\{i,j\})$, $\mu(\{i,j,k\})$, *etc.* But it seems very difficult to extract from these coefficients the *contribution* of i *alone*.
- if $\mu(\{1\})$ and $\mu(\{2\})$ are high (say 0.7), and $\mu(\{1,2\})$ is not very different (say 0.75), then clearly the importance of features 1, 2 taken together is much the same as 1 or 2 taken separately, and we have no interest in considering them both. We will speak here of negative synergy or redundancy. On the contrary, if $\mu(\{1\})$ and $\mu(\{2\})$ have low values (say, 0.1) and $\mu(\{1,2\})$ is large (say, 0.6), then although features 1 and 2 are unimportant when considered separately, they become very important when taken together. We speak then of positive synergy, or complementarity. Of course, as above we must consider the value of all coefficients $\mu(A)$ with $\{1,2\} \subset A$, to see the effect of adding either i, j or $\{i,j\}$ to a coalition.

Based on these ideas, it is possible to compute a global importance index and an interaction index. We define them in the next section.

9.3.4.1 *Shapley value and interaction index of a fuzzy measure*

Definition 9.4 Let μ be a fuzzy measure on X. The *importance index* or Shapley index of element i with respect to μ is defined by:

$$v_i = \sum_{K \subset X \setminus i} \gamma_{|K|}(\mu(K \cup i) - \mu(K)), \tag{9.11}$$

with $\gamma_k = \frac{(n-k-1)!k!}{n!} = \frac{1}{\binom{n-1}{k}n}$, $|K|$ indicating the cardinality of K, and $0! = 1$ as usual. The Shapley value of μ is the vector $v = [v_1 \cdots v_n]$. ■

This definition has been proposed by Shapley on an axiomatic basis in cooperative game theory [31], and possesses all suitable properties for representing importance of indices. In particular, the Shapley value has the property that $\sum_{i=1}^{n} v_i = 1$. It is convenient to scale these indices by a factor n, so that an importance index greater than 1 indicates a feature more important than the average.

Definition 9.5 The interaction index between two elements i and j with respect to a fuzzy measure μ is defined by:

$$I_{ij} = \sum_{K \subset X \setminus \{i,j\}} \xi_{|K|}(\mu(K \cup \{i,j\}) - \mu(K \cup i) - \mu(K \cup j) + \mu(K)), \qquad (9.12)$$

with $\xi_k = \frac{(n-k-2)!k!}{(n-1)!} = \frac{1}{\binom{n-2}{k}(n-1)}$. ■

It is easy to show that the maximum value of I_{ij} is 1, reached by the fuzzy measure μ defined by $\mu(K \cup i, j) = 1$ for every $K \subset X$, and 0 otherwise. Similarly, the minimum value of I_{ij} is -1, reached by μ defined by $\mu(K \cup i) = \mu(K \cup j) = 1$ for any $K \subset X$ and 0 otherwise. This definition has been proposed by Murofushi and Soneda [28], by using concepts of multiattribute utility theory, and is very similar to the Shapley value. In fact, Grabisch has shown that both can be embedded into a general interaction index, defined for any coalition [17]. A positive (resp. negative) value of the index corresponds to a positive (resp. negative) synergy. Let us apply these indices to the iris data set. Figs. 9.1 and 9.2 give the histograms of every feature for every class, as well as projections of the data set on some pairs of features. In these figures, samples of class 1 (resp 2, 3) are represented by squares (resp. triangles, circles).

Tables 9.3 give importance index and interaction indices computed from the result of learning by HLMS (classification rate is 95.3%). We can see that the Shapley value reflects the importance of features which can be assessed by examining the histograms and projection figures. Clearly, X_1 and X_2 are not able to discriminate among the classes, especially for classes 2 and 3. In contrast, X_3 and X_4 taken together are almost sufficient.

The interaction index values are not always so easy to interpret. However, note that X_1 and X_2 are complementary for class 1: the projection figure on these two axes shows effectively that they are almost sufficient for distinguishing class 1 from the others, although X_1 or X_2 alone were not. In contrast, these two features taken together are not more useful than X_1 or X_2 for classes 2 and 3 (redundancy). The fact that I_{14} for class 2 is strongly negative can be explained as follows. Looking at the projection figure on X_1, X_4, we

Table 9.3 Indexes of importance and interaction for the iris data set

index of importance v_i (scaled)			
feature	class 1	class 2	class 3
1	0.759	0.670	0.416
2	0.875	0.804	0.368
3	1.190	1.481	1.377
4	1.176	1.045	1.839

index of interaction I_{ij}			
features	class 1	class 2	class 3
1,2	0.128	-0.159	-0.065
1,3	0.051	0.281	0.052
1,4	0.054	-0.257	0.010
2,3	-0.009	0.114	0.002
2,4	-0.007	0.036	0.059
3,4	-0.051	0.132	-0.238

can see that X_1 (horizontal axis) brings no better information than X_4 for discriminating class 2 from the others, so that the combination $\{X_1, X_4\}$ is redundant. Concerning X_3 and X_4, the examination of the projection figure shows that they are rather complementary for classes 2 and 3. Although I_{34} is positive for class 2 as expected, it is strongly negative for class 3.

9.3.5 Related work

To our knowledge, the fuzzy integral has been first applied to classification in the beginning of the nineties, independently by Tahani and Keller [33], and by Grabisch and Sugeno [21, 22]. Later, much work has been done in image processing and character recognition (see [23] for a thorough survey on this topic, with a detailed bibliography; see also the work of Arbuckle *et al.* [1]). Also, we mention the use of fuzzy integrals as a means to combine the output of several classifiers (see, *e.g.*, Cho [5]).

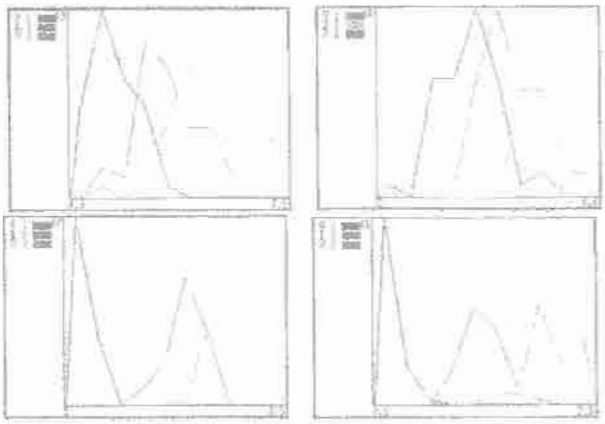


Fig. 9.1 Histograms of the iris data (from left to right: features 1 to 4)

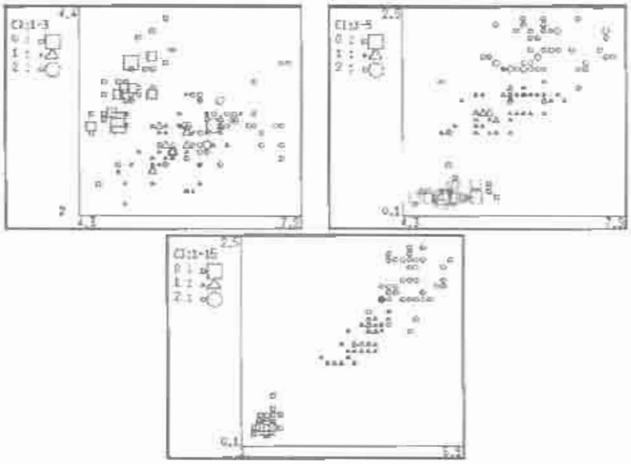


Fig. 9.2 Projections of the iris data (from left to right: on features 1 and 2, 1 and 4, 3 and 4 resp.)

References

[1] T. Arbuckle, E. Lange, T. Iwamoto, N. Otsu, and K. Kyuma. Fuzzy information fusion in a face recognition system. *Int. J. of Uncertainty,*

Fuzziness and Knowledge-Based Systems, 3(3):217–246, 1995.

[2] A. Ayoun and M. Grabisch. Tracks real-time classification based on fuzzy rules. *Int. J. of Intelligent Systems*, 12:865–876, 1997.

[3] J.C. Bezdek. *Pattern recognition with fuzzy objective function algorithms*. New York: Plenum Press, 1981.

[4] J.C. Bezdek and S.K. Pal (eds). *Fuzzy Models for Pattern Recognition*. New York: IEEE Press, 1992.

[5] S.B. Cho. Combining multiple neural networks by fuzzy integral for robust classification. *IEEE Tr. on Systems, Man and Cybernetics*, 25(2):380–384, 1995.

[6] G. Choquet. Theory of capacities. *Annales de l'Institut Fourier*, 5:131–295, 1953.

[7] P.A. Dondes and A. Rosenfeld. Pixel classification based on gray level and local busyness. *IEEE Tr. on Pattern Analysis and Machine Intelligence*, 4(1):79–84, 1982.

[8] D. Dubois and H. Prade. *Possibility Theory*. New York: Plenum Press, 1988.

[9] D. Dubois and H. Prade. Fuzzy sets in approximate reasoning, part 1: inference with possibility distributions. *Fuzzy Sets and Systems*, 40:143–202, 1991.

[10] D. Dubois and H. Prade. Gradual inference rules in approximate reasoning. *Information Sciences*, 61:103–122, 1992.

[11] D. Dubois, H. Prade, and S. Sandri. On possibility/probability transformations. In R. Lowen and M. Roubens, editors, *Fuzzy Logic, State of the Art*. New York: Kluwer Academic, 1993.

[12] D. Dubois, H. Prade, and C. Testemale. Weighted fuzzy pattern matching. *Fuzzy Sets & Systems*, 28:313–331, 1988.

[13] J. Figue, M. Grabisch, and M.-P. Charbonnel. A method for still image interpretation relying on a multi-algorithms fusion scheme, application to human face characterization. *Fuzzy Sets and Systems*, 103:317–337, 1999.

[14] P.Y. Glorennec. *Algorithmes d'apprentissage pour systèmes d'inférence floue*. Hermès, 1999.

[15] M. Grabisch. Fuzzy integral in multicriteria decision making. *Fuzzy Sets & Systems*, 69:279–298, 1995.

[16] M. Grabisch. A new algorithm for identifying fuzzy measures and its application to pattern recognition. In *Int. Joint Conf. of the 4th IEEE*

Int. Conf. on Fuzzy Systems and the 2nd Int. Fuzzy Engineering Symposium, pages 145–150, Yokohama, Japan, March 1995.

[17] M. Grabisch. k-order additive discrete fuzzy measures and their representation. *Fuzzy Sets and Systems*, 92:167–189, 1997.

[18] M. Grabisch. Fuzzy integral for classification and feature extraction. In M. Grabisch, T. Murofushi, and M. Sugeno, editors, *Fuzzy Measures and Integrals — Theory and Applications*, pages 415–434. Heidelberg:Physica Verlag, 2000.

[19] M. Grabisch, H.T. Nguyen, and E.A. Walker. *Fundamentals of Uncertainty Calculi, with Applications to Fuzzy Inference.* New York: Kluwer Academic, 1995.

[20] M. Grabisch and J.M. Nicolas. Classification by fuzzy integral — performance and tests. *Fuzzy Sets & Systems, Special Issue* on *Pattern Recognition*, 65:255–271, 1994.

[21] M. Grabisch and M. Sugeno. Fuzzy integral with respect to dual measures and its application to multi-attribute pattern recognition. In *6th Fuzzy Systems Symposium*, pages 205–209, Tokyo, Japan, September 1990. in japanese.

[22] M. Grabisch and M. Sugeno. Multi-attribute classification using fuzzy integral. In *1st IEEE Int. Conf. on Fuzzy Systems*, pages 47–54, San Diego, CA, March 1992.

[23] J.M. Keller, P.D. Gader, and A.K. Hocaoğlu. Fuzzy integrals in image processing and recognition. In M. Grabisch, T. Murofushi, and M. Sugeno, editors, *Fuzzy Measures and Integrals – Theory and Applications*, pages 435–466. Heidelberg: Physica Verlag, 2000.

[24] D.P. Mandal, C.A. Murthy, and S.K. Pal. Formulation of a multivalued recognition system. *IEEE Tr. on Systems, Man and Cybernetics*, 22:607–620, 1992.

[25] D.P. Mandal, C.A. Murthy, and S.K. Pal. Theoretical performance of a multivalued recognition system. *IEEE Tr. on Systems, Man and Cybernetics*, 24(7):1001–1021, 1994.

[26] J.L. Marichal. An axiomatic approach of the discrete Sugeno integral as a tool to aggregate interacting criteria in a qualitative framework. *IEEE Tr. on Fuzzy Systems*, to appear.

[27] O. Metellus and M. Grabisch. Une approche de la classification par filtrage flou — méthodologie et performances sur un problème de segmentation clientèle. In *Proc. Rencontres Francophones sur la Logique Floue et ses Applications (LFA)*, pages 215–220, Paris, France, November 1995.

[28] T. Murofushi and S. Soneda. Techniques for reading fuzzy measures (III): interaction index. In *9th Fuzzy System Symposium*, pages 693–696, Sapporo, Japan, May 1993. In Japanese.

[29] T.R. Reed and J.M. Hans du Buf. A review of recent texture segmentation and feature extraction techniques. *Image Understanding*, 57(3):359–372, 1993.

[30] E. H. Ruspini. A new approach to clustering. *Inform. Control*, 15(1):22–32, 1969.

[31] L.S. Shapley. A value for n-person games. In H.W. Kuhn and A.W. Tucker, editors, *Contributions to the Theory of Games, Vol. II*, number 28 in Annals of Mathematics Studies, pages 307–317. Princeton, NJ: Princeton University Press, 1953.

[32] M. Sugeno. *Theory of fuzzy integrals and its applications*. PhD thesis, Tokyo Institute of Technology, 1974.

[33] H. Tahani and J.M. Keller. Information fusion in computer vision using the fuzzy integral. *IEEE Tr. on Systems, Man, and Cybernetics*, 20(3):733–741, 1990.

[34] S.M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *11th IJCAI*, pages 781–787, 1989.

[35] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[36] L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets & Systems*, 1:3–28, 1978.

Chapter 10

NEURAL NETWORK BASED PATTERN RECOGNITION

V. David Sanchez A.

Advanced Computational Intelligent Systems
Pasadena, CA 91116-6130, U.S.A.
e-mail: *vdavidsanchez@earthlink.net*

Abstract

Connections between traditional methods and neural networks for pattern recognition are presented. For this purpose basic principles of statistics and pattern recognition are described as well as some of the most advanced neural network architectures and associated learning methods. To concretize the comparative assessment problem statements of key pattern recognition applications are introduced including the ones for classification, clustering, and regression.

10.1 Introduction

Pattern recognition has been an area of research for a few decades, pattern recognition solutions have been developed for different science and engineering fields. On the other hand, the use of neural networks in pattern recognition tasks has been actively pursued in recent years. An introductory presentation of pattern recognition and its subject of study can be found in [14, 15, 24, 32, 50, 71, 72]. Key relationships between the areas of neural networks and pattern recognition are summarized in [3, 21, 42, 51, 56, 62, 68]. Connections of neural networks to statistics are described in [9, 10, 34, 48]. Treatments of statistical pattern recognition include [12, 18, 40, 47, 77]. Pattern recognition using fuzzy and genetic algorithms can be found, *e.g.*, in [54, 55].

This chapter describes some key pattern recognition problems and neural network solutions. The relationship to other more conventional pattern recognition methods will be emphasized. This chapter is subdivided as follows. Section 10.2 describes the essence of pattern recognition. Section 10.3 presents advanced neural network architectures and their associated learning methods. Section 10.4 examines applications of neural pattern recognition. Finally, the conclusions are stated in Section 10.5.

10.2 The essence of pattern recognition

Pattern recognition is a well-defined field of research that investigates the analysis and design of systems capable of recognizing patterns in sensory data, notably in visual and sound data. A long-standing, unsolved goal has been the orientation-, location-, and scale-independent recognition of complex patterns. This area has been traditionally subdivided in statistical pattern recognition and syntactical pattern recognition. Statistical pattern recognition includes studies in discriminant analysis, feature extraction, and cluster analysis among others. Syntactical pattern recognition includes grammatical inference and parsing among others.

Areas of application for pattern recognition algorithms include image analysis, data mining, bioinformatics, optical character recognition, speech processing, man (medical) and machine diagnostics, financial trading, knowledge engineering, person identification, industrial inspection, among several others. More specific pattern recognition applications include fingerprint identification, handwriting recognition, X-ray image classification, DNA sequence analysis, internet search, stock option purchase decision support, target recognition, among many others.

10.2.1 Statistical pattern recognition

We first introduce some topics in probability theory and the Bayes rule to discuss the case of discriminant analysis from a statistical pattern recognition perspective. This provides a concrete example for the statistical approach of pattern recognition.

10.2.1.1 *Topics in probability theory and the Bayes rule*

Given two events E and F in a sample space S the conditional probability of E given F is defined by:

$$P(E|F) = \frac{P(EF)}{P(F)} \tag{10.1}$$

Given an event E and a partition F_i, $i = 1, \cdots, n$ of S, the events F_i being mutually exclusive, the averaging rule is defined as follows:

$$P(E) = \sum_{i=1}^{n} P(E|F_i) \cdot P(F_i) \tag{10.2}$$

Finally, for a given event E and a partition $F_i, i = 1, \cdots, n$ of S according to the Bayes rule:

$$P(F_i|E) = \frac{P(E|F_i) \cdot P(F_i)}{\sum_{j=1}^{n} P(E|F_j) \cdot P(F_j)} \tag{10.3}$$

Interpreting the Bayes rule in an environment where a model is refined by a new deta occurrence, the rule allows to compute the posterior model probabilities given the prior model probabilities and the new data occurrence. The posterior model probabilities are proportional to the likelihood, *i.e.*, the probability of the data given the model, times the prior model probabilities.

10.2.1.2 *The case of discriminant analysis*

Discriminant analysis identifies boundaries between groups of objects. The objects can then be classified into a number of criterion groups or classes C_j, $j = 1, \cdots, m$. Qualitative labels, the classification labels, are attached to the objects and constitute the criterion variables in discriminant analysis. The objects are measured on variables with quantitative values, called predictor variables gathered in the predictor vector $\vec{p}$, which are related to the object's membership in one criterion group. Regardless of group membership, all objects are measured on the same set of predictor variables. A function called the discriminant function is used to classify a given object, *e.g.*, $\vec{x} \in \mathbf{R}^N$, into a criterion group or class C_j. The discriminant function combines in a weighted form the values of all predictor variables.

From the introduction above, the determining parameters of the discriminant function are the weights of the predictor variables and the cutoff values for assigning objects into different criterion groups. Given data with a prior knowledge of the correct object-class assignment, called training data, the discriminant function is designed in such a way as to minimize classification errors while predicting the class membership of an object using the discriminant function. These errors can be visualized using the confusion matrix $\mathbf{C}^{m\times m}$ which tabulates the predicted versus the actual objects' group membership. $\mathbf{C}[i][j]$, $i, j = 1, \cdots, m$ states the number of objects which were predictively assigned to class i using the discriminant function when they are actually members of class j. If r is the total number of correct classifications and w the total number of incorrect classifications then:

$$r = \sum_{i=0}^{m} \mathbf{C}[i][i] \tag{10.4}$$

$$w = \sum_{i=0}^{m} \sum_{j=0, j\neq i}^{m} \mathbf{C}[i][j] \tag{10.5}$$

Two different approaches to classification either use data model assumptions or directly learn the conditional model. The first approach is called generative, the latter is called discriminative. Following a generative approach and using the Bayes rule defined in (10.3) the posterior probabilities $P(C_j|\vec{x})$ can be determined from the likelihood and prior probabilities. The expected loss of making a decision, *i.e.*, assigning an object $\vec{x}$ to class C_j is defined by:

$$l = \sum_{j=1}^{m} L_{ij} \cdot P(C_j|\vec{x}) \tag{10.6}$$

where L_{ij} are costs associated with making the decision of assignment to class i when j is the true class. According to the initial premise this expected loss needs to be minimized. Representatives of the generative approach to classification include mixture models and Hidden Markov Models (see, *e.g.*, [46] and [61] respectively). Representatives of the discriminative approach include, but are not restricted to linear classifiers, nearest neighbor algorithms, and neural networks which will be analyzed later in the discussion.

10.2.2 Syntactic pattern recognition

A highly introductory treatment of syntactical pattern recognition is included here mainly to establish the difference in character to the widely studied subfield of statistical pattern recognition. The basic principles of syntactical pattern recognition are included in [17, 20, 58]. Chapter 7 of this book also describes in detail various aspects of syntactic pattern recognition. The basis of this approach is to directly relate the structure of patterns to formal language syntax. Patterns are built in a hierarchical way from subpatterns, ..., all the way down to primitives, *i.e.*, the language alphabet. The rules for building patterns from primitives are dictated by the language grammar. Since the language grammar needs to be inferred from training sets, this approach requires a lot of computational power. Also, its conceptual basis does not explain how to detect primitives in noisy environments.

10.3 Advanced neural network architectures

A significant number of neural network architectures have been proposed in the literature for pattern recognition tasks, see *e.g.*, [26, 27, 28, 78]. To concretize our discussion one representative neural network architecture for supervised learning and one for unsupervised learning will be reviewed.

10.3.1 Supervised learning

In this section, for the purpose of a concrete presentation, we concisely review RBF (Radial Basis Function) networks and their associated learning methods.

10.3.1.1 *RBF networks*

RBF networks were introduced in [4, 49, 60] and in references cited therein. Extensions were presented among others in [8, 19, 29, 33, 52, 59, 57, 64, 65, 66]. RBF networks possess three layers: one input, one hidden, and one output layer. The hidden layer contains neurons realizing basis functions. The output layer contains one neuron for the approximation of functions $f : \mathbf{R}^n \rightarrow \mathbf{R}$. The approximation function g_m realized by a network architecture with m hidden units has the form in (10.7). The weights w_i, $i = 1, \cdots, m$ are parameters weighting the connections between each of the hidden neurons and the output neuron.

Table 10.1 Examples of RBF nonlinearities

Function name	Function expression $\phi(r, c) =$ (c=constant)
linear	r
cubic	r^3
thin plate spline	$r^2 \cdot \log r$
Gaussian	$\exp(-\frac{r^2}{c^2})$
multiquadric	$(r^2 + c^2)^{\pm 1/2}$

The basis functions $\phi_i : \mathbf{R}^n \rightarrow \mathbf{R}$ realized in the individual hidden units are parameterized scalar functions, which are built using a given nonlinearity $\phi : \mathbf{R} \rightarrow \mathbf{R}$. To parameterize this function the centers $\vec{z}_i$ and the widths σ_i for $i = 1, \cdots, m$ are used, see (10.8). Examples of nonlinearities for RBF networks are summarized in Table 10.1, one of the most commonly used is the Gaussian nonlinearity.

$$g_m(\vec{x}) = \sum_{i=1}^{m} w_i \cdot \phi_i(\vec{z}_i, \sigma_i, \vec{x}) \tag{10.7}$$

$$\phi_i(\vec{z}_i, \sigma_i, \vec{x}) = \phi(||\vec{x} - \vec{z}_i||, \sigma_i), \quad 1 \leq i \leq m \tag{10.8}$$

10.3.1.2 *Learning methods*

In a line of study, learning methods for RBF networks have been systematically developed. A learning method for weight determination was presented in [63]. A novel learning method for the automatic design of RBF networks for regression applications was introduced in [64]. Both of these methods handle noise-free and noisy data. A new robust learning method for RBF networks which handles outliers in the data as well was introduced in [65]. Further advances were presented in a special issue on RBF networks in [66].

To gain some insight here, the most basic learning method for weight determination is outlined. For the RBF networks realizing g_m in (10.7), fixed values for m, $\vec{z}_i, \sigma_i, i = 1, \cdots, m$, and a given training data set $\mathrm{Tr} = \{(\vec{x}_i, y_i), i = 1, \cdots, N\}$, the learning method needs to solve the optimization problem in (10.9) determining the weight vector $\vec{w} = (w_1, \cdots, w_m)^T$. This leads to the solution of a linear equation system or more specifically, of a linear least-squares problem and to the maximization of the memorization capability for a fixed number of hidden units m. The actual learning goal, *i.e.*, the optimization of the generalization capability, is dependent on m (model complexity) and, in practical settings, is based on a given test data set $\mathrm{Te} = \{(\vec{x}_k, y_k),\ k = 1, \cdots, M\}$, as

$$\min_{\vec{w}} e_{Tr},$$

where

$$e_{Tr} = \frac{1}{N} \cdot \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{m} w_j \cdot \phi_j(\vec{x}_i) \right)^2 \tag{10.9}$$

Recently, kernel methods and support vector machines (SVM) have become very popular for classification and regression, in particular when using RBF networks. Details on support vector algorithms and learning can be found in [11, 70, 74]. We will discuss the main aspects of the corresponding learning methods when we present neural network solutions to these pattern recognition tasks.

10.3.2 Unsupervised learning

In this section, for the purpose of a concrete presentation, we concisely review SOM (Self-Organizing Map) networks and their associated learning methods.

10.3.2.1 *SOM networks*

SOM networks have been extensively studied, *e.g.*, in [35, 36, 38, 53]. Extensions of more flexible map structures are included in [5, 45], of hierarchical approaches and speed-up in [39, 43], for closely related topographic models see [2, 16]. The neurons of a SOM network are allocated on a discrete lattice. Given an input $\vec{x}$, only the neuron which best represents the input together with its neighbors, is allowed to learn, thus realizing competitive learning and ordered representations within the lattice. Each neuron i of the lattice is represented by a reference vector $\vec{r}_i$, typically a two-dimensional vector. For a given input $\vec{x}$ the winner of the competition is the neuron $i(\vec{x})$ whose reference vector $\vec{r}_i$ is the nearest to $\vec{x}$ according to:

$$i(\vec{x}) = \arg\min_i ||\vec{x} - \vec{r}_i||^2 \tag{10.10}$$

10.3.2.2 *Learning methods*

The basic learning method for SOM networks is described in the following iterative updating scheme for the reference vectors:

$$\vec{r}_j(t+1) = \vec{r}_j(t) + k_{i(\vec{x}),j}(t)[\vec{x} - \vec{r}_j(t)] \tag{10.11}$$

k is a neighborhood kernel function which decreases with time and the distance to the origin. It is a scalar function of a two-dimensional argument. This function is used to control the influence of adaptation, in decreasing manner with the distance to the reference vector of the winning unit according to:

$$k_{i,j}(t) = k(||\vec{r}_i - \vec{r}_j||, t) \tag{10.12}$$

10.4 Neural pattern recognition

Three key pattern recognition applications are discussed: classification, clustering, and regression. For each of them, the problem is formally stated first. A neural network solution to the problem is then described making use of the neural networks previously reviewed in this chapter and discussing its relationship to more conventional pattern recognition methods.

10.4.1 Classification

10.4.1.1 *Problem statement*

To make things easier to present, a two-class classification problem is defined and their solution described. Given a set of labeled patterns consisting of the patterns, which are vectors $\vec{x}_i \in \mathbf{R}^n, i = 1, \cdots, N$, and their respective class labels $y_i \in \{-1, +1\}, i = 1, \cdots, N$ the classification problem consists in finding a decision function $f : \mathbf{R}^n \to \{-1, +1\}$ which accurately labels a given pattern $\vec{x}$ with the corresponding class label $y = f(\vec{x})$.

The given vector $\vec{x}$ can either be or not be in the original training set $\{(\vec{x}_i, y_i), i = 1, \cdots, N\}$. Typically, the probability distribution of the given training set on $\mathbf{R}^n \times \{-1, +1\}$ is unknown and a parameterized discriminant function $d : \mathbf{R}^n \to \mathbf{R}$ is used according to: $f(\vec{x}) = \mathrm{sgn}(d(\vec{p}; \vec{x}))$, where $\vec{p}$ represents the set of parameters, sgn is the 'sign' function.

10.4.1.2 *Neural network solution*

As previously stated, we present an RBF network solution with an associated SVM learning method. The structural risk minimization principle was introduced in [73] providing a bound for the risk function R in terms of the empirical risk R_e. The latter is computed using the given training data, the number of training data points N, and the VC-dimension h of the set of functions $\{f_\alpha\}$ used (α needed to parameterize the function) when attempting to minimize the risk to find the optimal decision function for a classification problem. The bound is stated in (10.13), $\forall\alpha$ with probability $\geq 1 - \eta$:

$$R(\alpha) \leq R_e(\alpha) + \Psi(N, \eta, h) \tag{10.13}$$

$$\Psi(N, \eta, h) = \sqrt{\frac{h(\log \frac{2N}{h} + 1) - \log(\frac{\eta}{4})}{N}} \tag{10.14}$$

For dichotomous classification (two-class problems) we use the decision function $\mathrm{sgn}(f(\vec{x}))$, *cf.* (10.15) and the RBF kernels given in (10.16).

$$f(\vec{x}) = b + \sum_{i=1}^{m} y_i \cdot \alpha_i \cdot K(\vec{x}_i, \vec{x}) \tag{10.15}$$

$$K(\vec{x_i}, \vec{x_j}) = e^{-\frac{||\vec{x_i} - \vec{x_j}||^2}{2\sigma^2}} \tag{10.16}$$

where $\{(\vec{x}_k, y_k), k = 1, \cdots, N\}$ is the given training data set. The bias b, the number of support vectors m, the support vectors themselves $\vec{x}_i$ and associated $y_i, i = 1, \cdots, m$ as well as the Lagrangian multipliers α_i need to be determined. To achieve this, the kernel learning method maximizes the Lagrangian objective function in (10.17) subject to the conditions in (10.18) $\forall i = 1, \cdots, m$.

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\vec{x_i}, \vec{x_j}) \tag{10.17}$$

$$0 \leq \alpha_i,$$

$$\sum_{i=1}^{m} \alpha_i \cdot y_i = 0 \tag{10.18}$$

The bias b of the decision function is computed using (10.19):

$$b = -\frac{1}{2} \cdot \left\{ \max_{i|y_i=-1} \sum_{j=1}^{m} y_j \cdot \alpha_j^* \cdot K(\vec{x}_i, \vec{x}_j) \; + \right.$$
$$\left. \min_{i|y_i=+1} \sum_{j=1}^{m} y_j \cdot \alpha_j^* \cdot K(\vec{x}_i, \vec{x}_j) \right\} \tag{10.19}$$

where α_j^* are the optimal values previously determined. To diminish the effect of noise soft constraints are incorporated according to: $0 \leq \alpha_i \leq C$ to trade-off the memorization and generalization capability of the RBF network making use of a test data set. This neural network solution to the posed classification problem can be interpreted as a linear maximal classifier in a higher dimensional space than the original, called feature space, after mapping the original data through a nonlinear function $\phi(\vec{x})$ implicitly defined by the inner product:

$$K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j}) \tag{10.20}$$

The relationship to more conventional pattern recognition techniques like a linear classifier whose decision function is given by $\mathrm{sgn}(f(\vec{x}))$, see (10.21), becomes transparent when using this framework. In the case of a linear classifier trained with separable data, the separating hyperplane satisfies:

$$f(\vec{x}) = b + \vec{w} \cdot \vec{x} = 0 \tag{10.21}$$

where $\vec{w}$ is a normal to the separating hyperplane. Two hyperplanes H_+ and H_- are defined by $b + \vec{w} \cdot \vec{x}_i \geq 1, \forall y_i = +1$ and $b + \vec{w} \cdot \vec{x}_i \leq -1, \forall y_i = -1$ respectively. These conditions can be unified in (10.22). The margin, *i.e.*, the distance between hyperplane H_+ and H_-, is given by $2/||\vec{w}||$. To find a solution the maximum margin is determined by minimizing $||\vec{w}||^2$ subject to the constraints in (10.22).

$$y_i \cdot (b + \vec{w} \cdot \vec{x}_i) - 1 \geq 0, \forall i = 1, \cdots, N \tag{10.22}$$

This problem can be shown to be equivalent to maximizing the Lagrangian, dual objective function in (10.23) subject to the conditions in (10.18).

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \tag{10.23}$$

The similarity with the problem formulation for the neural network solution using a SVM classifier, cf (10.17) is remarkable. The scalar product $\vec{x_i} \cdot \vec{x_j}$ in (10.23) is replaced by $K(\vec{x_i}, \vec{x_j})$ in (10.17) confirming that the neural network solution is a linear maximal classifier in the feature space. Formally $m = N$ (size of training data set), but the solution typically contains some values $\alpha_i = 0$ leading to $m < N$, where m is the actual number of supporting vectors.

10.4.2 Clustering

10.4.2.1 *Problem statement*

Given a set of points $\{\vec{x}_i \in \mathbf{R}^p, i = 1, \cdots, n\}$ the goal of clustering is to determine a set of clusters C_j each of them represented by its prototype $\{\vec{c}_j \in \mathbf{R}^p, j = 1, \cdots, m\}$ such that the distances between points of the same cluster are minimized and the distances between clusters are maximized.

This corresponds to central clustering as opposed to pairwise clustering. This will make things easier to present. In pairwise clustering pairwise distances are given $\{d_{i,j} \in \mathbf{R}, i, j = 1, \cdots, n\}$. The goal of clustering stated needs to be further specified, *e.g.*, in relation to the distance measures used. Table 10.2 and Table 10.3 show commonly used distance measures within and between clusters. $||\cdot||$ is the Euclidian norm, m_j is the number of points belonging to cluster $j = 1, \cdots, m$, and the centroid $\vec{c}_j$ of a cluster is given by:

$$\vec{c}_j = \frac{\sum\limits_{\vec{x}_i \in C_j} \vec{x}_i}{m_j} \tag{10.24}$$

The result of clustering is an element of the set of assignment matrices M which is a Boolean representation of data partitionings:

$$\mathcal{M}_{n,m} = \left\{ M \in \{0,1\}^{n \times m}, \sum_{j=1}^{m} M_{i,j} = 1, i = 1, \cdots, n \right\} \tag{10.25}$$

$M_{i,j} = 1$ and $M_{i,j} = 0$ indicate that the data point $\vec{x}_i$ belongs and does not belong to cluster C_j respectively.

Table 10.2 Distance Measures within Clusters

Expression	Description
$d_j = \dfrac{\sum_{i=1}^{m_j} \sum_{k=1}^{m_j} \lVert \vec{x}_i - \vec{x}_k \rVert}{m_j \cdot (m_j - 1)}$	average distance
$d_j = \dfrac{\sum_{i=1}^{m_j} \lVert \vec{x}_i - \vec{c}_j \rVert}{m_j}$	centroid distance
$d_j = \dfrac{\sum_{i=1}^{m_j} \min_k \lVert \vec{x}_i - \vec{x}_k \rVert}{m_j}$	nearest neighbor distance

Table 10.3 Distance Measures between Clusters

Expression	Description
$d_{C_j,C_l} = \min_{\vec{x}_i \in C_j, \vec{x}_k \in C_l} \lVert \vec{x}_i - \vec{x}_k \rVert$	single linkage distance
$d_{C_j,C_l} = \max_{\vec{x}_i \in C_j, \vec{x}_k \in C_l} \lVert \vec{x}_i - \vec{x}_k \rVert$	complete linkage distance
$d_{C_j,C_l} = \dfrac{\sum_{i=1}^{m_j} \sum_{k=1}^{m_l} \lVert \vec{x}_i - \vec{x}_k \rVert}{m_j \cdot m_l}$	average linkage distance
$d_{C_j,C_l} = \lVert \vec{c}_j - \vec{c}_l \rVert$	centroid linkage distance

10.4.2.2 *Neural network solution*

A neural network solution for clustering using the SOM network is presented and discussed in relation to a more conventional technique. For that purpose the criterion functions for the SOM network and the k-means clustering technique are compared. Additional material including specific treatment of clustering when using a SOM network was reported in [41, 76]. Other neural network approaches for clustering include the use of competitive networks as in [6]. More conventional pattern recognition algorithms for data clustering include methods based on mixed models proposed in [46], the k-means and ISODATA algorithms introduced in [44] and [22] respectively, as well as others studied, *e.g.*, in [1, 25, 31].

The criterion function used in k-means clustering for a given data set $\{\vec{x}_t, t = 1, \cdots, n\}$ is defined in (10.26) where $\vec{c}_l$ are the centroid vectors of the clusters $C_l, l = 1, \cdots, m$.

$$E = \sum_{l=1}^{m} \sum_{\vec{x}_t \in C_l} ||\vec{x}_t - \vec{c}_l||^2 \tag{10.26}$$

On the other hand, the criterion function for the SOM network [37] is given in (10.27) for the case of a discrete data set and a fixed neighborhood kernel k. n and m are the size of the data set and the number of neurons respectively. $\vec{r}_l$ is the reference vector associated to neuron $l = 1, \cdots, m$ of the SOM network and $i(\vec{x}_t)$ is the index of the neuron whose reference vector is the closest to $\vec{x}_t$.

$$E = \sum_{l=1}^{m} \sum_{t=1}^{n} k_{i(\vec{x}_t),j} ||\vec{x}_t - \vec{r}_l||^2 \tag{10.27}$$

From (10.26) and (10.27) we can observe that if the kernel function were to become non-zero only for the index of the neuron whose reference vector is the closest to $\vec{x}_t$ then the SOM clustering would reduce to the conventional k-means clustering. Otherwise, the reference vectors $\vec{r}_l$ associated to the neurons of the SOM network differ from the centroids $\vec{c}_l$ and are generated by local averaging of all data set vectors, their weighting is determined making use of the neighborhood kernel function k.

10.4.3 Regression

10.4.3.1 *Problem statement*

The linear multivariate regression case is described for convenience of presentation. The purpose of regression analysis can be summarized as to determine whether and which kind of a quantitative relationship (expressed by a regression equation) exists between two vector variables: the criterion variable $\vec{x} \in \mathbf{R}^n$ and the predictor variable $\vec{y} \in \mathbf{R}^m$ as well as to assess the prediction accuracy of the regression equation. Under zero-mean Gaussian noise $\vec{n} \in \mathbf{R}^m \sim \mathcal{N}(0, \mathbf{V}), \mathbf{V}$ is its covariance matrix, the data model is defined by the following regression equation: $\vec{y} = \vec{b} + \mathbf{A} \cdot \vec{x} + \vec{n}$, where $\vec{b} \in \mathbf{R}^m$ and $\mathbf{A} \in \mathbf{R}^{m \times n}$ represents a linear transformation.

In the case of a scalar predictor variable $y \in \mathbf{R}$ the matrix $\mathbf{A}$ becomes a row vector $\vec{w}$ and the decision function from the data model without considering the noise component becomes $f(\vec{x}) = b + \vec{w} \cdot \vec{x}$ where $b \in \mathbf{R}$ and $\| \cdot \|$ represents the scalar product. As in the classification case, a training data set $\{(\vec{x}_i, y_i), i = 1, \cdots, N\}$ is given to generate a solution, the difference is that y_i are real-valued. In the case of a nonlinear transformation $f : \mathbf{R}^n \rightarrow \mathbf{R}$ the data model becomes more complex and neural networks offer an efficient solution for that purpose.

10.4.3.2 *Neural network solution*

We present an RBF network solution with an associated SVM learning method. This regression solution is based on statistical learning theory similar to the neural network solution for classification. Additional, more specific regression-related material can be found, *e.g.*, in [13, 69, 75]. Theoretical foundations were reported in [67]. For regression the decision function is given in (10.28).

$$f(\vec{x}) = b + \sum_{i=1}^{m} y_i \cdot (\alpha_i - \beta_i) \cdot K(\vec{x}_i, \vec{x}) \tag{10.28}$$

For an ϵ-insensitive loss function:

$$L(x) = \begin{cases} 0, & |x| < \epsilon \\ |x|, & \epsilon \leq |x| \end{cases} \tag{10.29}$$

a quadratic optimization problem needs to be solved: the dual objective function to be minimized is given in (10.30) subject to the conditions in (10.31) $\forall i = 1, \cdots, m$.

$$\begin{aligned} W(\alpha, \beta) = \sum_{i=1}^{m} y_i(\alpha_i - \beta_i) - \epsilon \sum_{i=1}^{m} (\alpha_i + \beta_i) \cdots \\ - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} (\alpha_i - \beta_i)(\alpha_j - \beta_j) K(\vec{x_i}, \vec{x_j}) \end{aligned} \tag{10.30}$$

$$\begin{aligned} 0 \leq \alpha_i \leq C, \quad 0 \leq \beta_i \leq C, \\ \sum_{i=1}^{m} \alpha_i = \sum_{i=1}^{m} \beta_i \end{aligned} \tag{10.31}$$

The bias b is typically determined by averaging individual values which are gained from the Karush-Kuhn-Tucker conditions leading to $b = y_i - \vec{w} \cdot \vec{x}_i \pm \epsilon$, see *e.g.*, [7], where:

$$\vec{w} = \sum_{i=1}^{m} y_i(\alpha_i^* - \beta_i^*)\vec{x}_i \tag{10.32}$$

α_i^* and β_i^* are the optimal values previously determined. Similar quadratic optimization problems are generated when instead of the ϵ-insensitive loss function quadratic or robust loss functions [23, 30] are utilized. A robust learning method for RBF regression networks was introduced in [65]. In the case of regression, the relationship between this neural network solution and a more conventional pattern recognition approach like linear regression shows again the close connection: The linear regressor's decision function is given in (10.33). The optimization problem consists in the maximization of the functional given in (10.34) subject to the constraints in (10.31) as before, and the solution $\vec{w}$ is given in (10.35), whereas b is determined as in the case of the RBF network solution.

$$f(\vec{x}) = b + \vec{w} \cdot \vec{x} \tag{10.33}$$

$$\begin{aligned} W(\alpha, \beta) &= \sum_{i=1}^{m} y_i(\alpha_i - \beta_i) - \epsilon \sum_{i=1}^{m} (\alpha_i + \beta_i) \cdots \\ &\quad - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} (\alpha_i - \beta_i)(\alpha_j - \beta_j)\vec{x_i} \cdot \vec{x_j} \end{aligned} \tag{10.34}$$

$$\vec{w} = \sum_{i=1}^{m} (\alpha_i^* - \beta_i^*)\vec{x}_i \tag{10.35}$$

10.5 Conclusions

The explicit relationship between neural networks and more conventional methods for pattern recognition was discussed. RBF and SOM networks were reviewed for developing prototypical solutions to supervised and unsupervised learning problems respectively. Some of the most relevant pattern recognition tasks including classification, clustering, and regression were covered in-depth together with neural network solutions to show the concrete and deep relationship between pattern recognition and neural networks.

References

[1] M.R. Anderberg. *Cluster Analysis for Applications*. New York: Academic Press 1973.

[2] C.M. Bishop, M. Svensen, and C.K.I. Williams. *Developments of the Generative Topographic Mapping*. Neurocomputing 21 (1998), 203-224.

[3] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press 1999.

[4] D.S. Broomhead and D. Lowe. *Multivariable functional interpolation and adaptive networks*. Complex Systems 2 (1998), 321-355.

[5] J. Bruske and G. Sommer. *Dynamic cell structure learns perfectly topology preserving map*. Neural Computation 7 (1995), 845-865.

[6] J. Buhmann and H. Kühnel. *Complexity optimized data clustering by competitive neural networks*. Neural Computation 5 (1993), 75-88.

[7] C. Campbell. *An introduction to kernel methods*. In R.J. Howlett and L.C. Jain (Eds.): *Radial Basis Function Networks: Design and Applications*, Heidelberg: Physica-Verlag 2000, 155-192.

[8] S. Chen, C.F.N. Cowan, and P.M. Grant. *Orthogonal least squares learning algorithm for radial basis function networks*. IEEE Transactions on Neural Networks 2 (1991), 302-309.

[9] B. Cheng and D.M. Titterington. *Neural Networks: A review from a statistical perspective*. Statistical Science 9 (1994), 2-54.

[10] V. Cherkassky, J.H. Friedman, H. Wechsler (Eds.). *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*. Berlin: Springer 1994.

[11] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. New York: Cambridge University Press 2000.

[12] P.A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. London: Prentice-Hall 1982.

[13] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V. Vapnik. *Support vector regression machines*. in M. Mozer, M. Jordan, and T. Petsche (Eds.): Advances in Neural Information Processing Systems 9, Cambridge, MA: The MIT Press 1997, 155-161.

[14] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons 1973.

[15] R.O. Duda, P.E. Hart, and D.E. Stork. *Pattern Classification*. New

York: John Wiley & Sons, 2nd Edition, 2000.

[16] B. Fritzke. *Growing cell structures - A self-organizing network for unsupervised and supervised learning.* Neural Networks 7 (1994), 1441-1460.

[17] K.S. Fu. *Syntactic Pattern Recognition and Applications.* Englewood Cliffs, NJ: Prentice-Hall 1982.

[18] K. Fukunaga. *Introduction to Statistical Pattern Recognition*, 2nd Edition. New York: Morgan Kaufmann 1990.

[19] F. Girosi. *Some extensions of radial basis functions and their applications in artificial intelligence.* Computers and Mathematics with Applications 24 (1992), 61-80.

[20] R. Gonzalez and M. Thomason. *Syntactic Pattern Recognition An Introduction.* Reading, MA: Addison-Wesley 1978.

[21] I. Guyon and P.S.P. Wang (Eds.). *Advances in Pattern Recognition Systems using Neural Network Technologies.* Singapore: World Scientific 1994.

[22] D.J. Hall and G.B. Ball. *ISODATA: A novel method of data analysis and pattern classification.* Stanford Research Institute, Technical Report 1965.

[23] F.R. Hampel, E.M. Rochetti, P.J. Rousseeuw, and W.A. Stahel. *Robust Statistics.* New York: John Wiley & Sons 1986.

[24] D.J. Hand. *Discrimination and Classification.* Chichester: John Wiley & Sons 1981.

[25] J. Hartigan. *Clustering Algorithms.* New York: John Wiley & Sons 1975.

[26] S. Haykin. *Neural Networks: A Comprehensive Foundation.* New York: MacMillan 1994.

[27] R. Hecht-Nielsen. *Neurocomputing.* Reading, MA: Addison-Wesley 1990.

[28] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Reading, MA: Addison-Wesley 1991.

[29] R.J. Howlett and L.C. Jain (Eds.). *Radial Basis Function Networks: Design and Applications.* Heidelberg: Physica-Verlag 2000.

[30] P.J. Huber. *Robust Statistics.* New York: John Wiley & Sons 1981.

[31] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data.* Englewood Cliffs: Prentice-Hall 1988.

[32] M. James. *Pattern Recognition.* New York: John Wiley & Sons 1988.

[33] P.A. Jokinen. *A nonlinear network model for continuous learning.*

Neurocomputing 3 (1991), 157-176.

[34] J.W. Kay and D.M. Titterington (Eds.). *Statistics and Neural Networks – Advances at the Interface.* Oxford: Oxford University Press 1999.

[35] T. Kohonen. *Self-organized formation of topologically correct feature maps.* Biological Cybernetics 43 (1982), 59-69.

[36] T. Kohonen. *The Self-Organizing Map.* Proceedings of the IEEE 78 (1990), 1464-1480.

[37] T. Kohonen. *The self-organizing maps: Optimization approaches.* In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (Eds.). *Artificial Neural Networks,* Amsterdam: Elsevier Science 1991, 981-990.

[38] T. Kohonen, T.S. Huang (Ed.), and M.R. Schroeder (Ed.). *Self-Organizing Maps.* Berlin: Springer-Verlag, 3rd Edition, 2000.

[39] P. Koikkalainen. *Fast deterministic self-organizing maps.* in F. Soulie and P. Gallinari (Eds.): Proceedings of ICANN'95, Paris, France, vol. II, 63-68.

[40] P.R. Krishnaiah and L.N. Kanal (Eds.). *Handbook of Statistics 2: Classification, Pattern Recognition and Reduction of Dimensionality.* Amsterdam: North-Holland 1982.

[41] J. Lampinen and E. Oja. *Clustering properties of hierarchical self-organizing maps.* Journal of Mathematical Imaging and Vision 2 (1992), 261-272.

[42] C.G. Looney. *Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists.* Oxford: Oxford University Press 1997.

[43] S.P. Luttrell. *Hierarchical vector quantization.* IEE Proceedings 136 (1989), 405-413.

[44] J. MacQueen. *Some methods for classification and analysis of multivariate observations,* in Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability 1967, 281-297.

[45] T. Martinetz and K. Schulten. *Topology representing networks.* Neural Networks 7 (1994), 507-522.

[46] G.J. McLachlan and K.E. Basford. *Mixture Models.* New York: Marcel Dekker 1988.

[47] G.J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition.* New York: John Wiley & Sons 1992.

[48] D. Michie, D.J. Spiegelhalter, and C.C. Taylor (Eds.). *Machine Learn-*

ing, Neural and Statistical Classification. New York: Ellis Horwood 1994.

[49] J. Moody and C.J. Darken. *Fast learning in networks of locally-tuned processing units.* Neural Computation 1 (1989)1, 281-294.

[50] H. Niemann. *Pattern Analysis and Understanding,* Second Edition. Berlin: Springer-Verlag 1989.

[51] A. Nigrin. *Neural Networks for Pattern Recognition,* Cambridge, MA: The MIT Press 1993.

[52] P. Niyogi and F. Girosi. *On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions.* Neural Computation 8 (1996), 819-842.

[53] E. Oja and S. Kaski (Eds.). *Kohonen Maps.* Amsterdam: Elsevier Science 1999.

[54] S.K. Pal and P.P. Wang. *Genetic Algorithms for Pattern Recognition.* Boca Raton: CRC Press 1996.

[55] S.K. Pal and S. Mitra. *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing.* New York: John Wiley & Sons 1999.

[56] Y.-H. Pao. *Adaptive Pattern Recognition and Neural Networks.* Reading, MA: Addison-Wesley 1989.

[57] J. Park and I.W. Sandberg. *Approximation and radial-basis-function networks.* Neural Computation 5 (1993), 305-316.

[58] T. Pavlidis. *Structural Pattern Recognition.* Berlin: Springer-Verlag 1977.

[59] J. Platt. *A resource-allocation network for function interpolation.* Neural Computation 3 (1991), 213-225.

[60] T. Poggio and F. Girosi. *Networks for approximation and learning.* Proceedings of the IEEE 78 (1990) 9, 1481-1497.

[61] L.R. Rabiner. *A tutorial on hidden Markov models and selected applications in speech recognition.* Proceedings of the IEEE 77 (1989), 257-286.

[62] B.D. Ripley. *Pattern Recognition and Neural Networks.* New York: Cambridge University Press 1996.

[63] V. David Sánchez A., *On the Design of a Class of Neural Networks,* Journal of Network and Computer Applications 19 (1996), 111-118.

[64] V.D. Sánchez A., *Advances towards the automatic design of RBF networks.* International Journal of Knowledge-Based Intelligent Engineering Systems, 1 (1997), 168-174.

[65] V. David Sánchez A., *New Robust Learning Method.* International

Journal of Smart Engineering System Design 1 (1998), 223-233.

[66] V.D. Sánchez A., *Special Issue on RBF Networks.* Neurocomputing 19 (1998) 1-3 and 20 (1998) 1-3.

[67] J. Shawe-Taylor, S. Ben-David, P. Koiran, and R. Schapire. *Special Issue on Theoretical Analysis of Real-Valued Function Classes.* Neurocomputing 29 (1999) 1-3.

[68] R.J. Schalkoff. *Pattern Recognition: Statistical, Structural, and Neural Approaches.* New York: John Wiley & Sons 1992.

[69] B. Schoelkopf, P. Bartlett, A.J. Smola, and R. Williamson. *Shrinking the tube: A new support vector regression algorithm*=2E in M.S. Kearns, S.A. Solla, and D.A. Cohn (Eds.): Advances in Neural Information Processing Systems 11, Cambridge, MA: The MIT Press 1999, 330-336.

[70] B. Schoelkopf, C.J.C. Burges, and A.J. Smola (Eds.). *Advances in Kernel Methods – Support Vector Learning.* Cambridge, MA: The MIT Press 1999.

[71] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* San Diego, CA: Academic Press 1998.

[72] C.W. Therrien. *Decision, Estimation, and Classification: An Introduction to Pattern Recognition and Related Topics.* New York: John Wiley & Sons 1989.

[73] V. Vapnik. *Estimation of Dependencies Based on Empirical Data* (in Russian). English Translation: New York: Springer-Verlag 1982.

[74] V. Vapnik. *The Nature of Statistical Learning.* New York: Springer-Verlag 1995.

[75] V. Vapnik, S.E. Golowich, and A. Smola. *Support vector method for function approximation, regression estimation, and signal processing.* in M. Mozer, M. Jordan, and T. Petsche (Eds.): Advances in Neural Information Processing Systems 9, Cambridge, MA: The MIT Press 1997, 281-287.

[76] J. Vesanto and E. Alhoniemi. *Clustering of the Self-Organizing Map.* IEEE Transactions on Neural Networks 11 (2000), 586-600.

[77] A. Webb. *Statistical Pattern Recognition.* Oxford: Oxford University Press 1999.

[78] M. Zurada. *Introduction to Artificial Neural Systems.* Boston, MA: PWS Publishing 1992.

Chapter 11

PATTERN CLASSIFICATION BASED ON QUANTUM NEURAL NETWORKS: A CASE STUDY

N. B. Karayiannis*, R. Kretzschmar† and H. Richner††

*Department of Electrical and Computer Engineering
University of Houston
Houston, Texas 77204-4793, U.S.A.
e-mail: Karayiannis@UH.EDU

†Signal and Information Processing Laboratory
Swiss Federal Institute of Technology (ETH)
Zurich, SWITZERLAND
e-mail: kretzsch@isi.ee.ethz.ch

††Institute for Atmospheric Science
Swiss Federal Institute of Technology (ETH)
Zurich, SWITZERLAND
e-mail: richner@atmos.umnw.ethz.ch

Abstract

This chapter presents the development, testing and evaluation of pattern classifiers designed using quantum neural networks (QNNs) to remove bird-contaminated wind profiler data. QNNs are trainable feedforward neural networks inherently capable of capturing and quantifying uncertainty in the training data. In this application, the pattern classifiers were developed by training QNNs to identify and remove bird-contaminated data recorded by a 1290-MHz wind profiler using a set of input features computed from the wind profiler measurements. A series of experiments were designed to evaluate several sets of features extracted from wind profiler data, compare various QNNs and traditional feedforward neural networks (FFNNs) of different sizes, and rate the criteria employed for identifying birds in wind profiler data based on the outputs of the trained neural networks. This experimental study indicates that QNN-based pattern classifiers can remove up to 90% of bird-contaminated wind profiler data, while QNNs are strong competitors to traditional FFNNs for real-world pattern classification applications.

11.1 Introduction

Feedforward neural networks (FFNNs) have been a natural choice as trainable pattern classifiers because of their function approximation capability and generalization ability [1]. The function approximation capability allows them to form arbitrary nonlinear discriminant surfaces while the generalization ability allows them to respond consistently to data they were not trained with. FFNNs are trained to implement a desired input-output mapping from the sample information provided by the training data. As a result, they may be unable to recognize the structure inherent in the feature space which they are not explicitly trained to recognize. One of the major disadvantages of FFNNs is their inability to correctly assign class membership to data samples belonging to regions of the feature space where there is overlapping between classes. The reason for this is that FFNNs use sharp decision boundaries to partition the feature space. As a result, the outputs of trained FFNNs cannot generally be interpreted as membership values. This motivated the development of neuro-fuzzy systems by merging neural modeling with fuzzy-theoretic concepts [4, 5, 6, 7, 12, 13, 14, 15].

A theoretical study on the capacity of conventional FFNNs to deal with uncertainty led to the development of inherently fuzzy feedforward neural networks, known as quantum neural networks (QNNs) [5, 14, 15]. Conventional FFNNs and QNNs satisfy the requirements for universal function approximators [10]. In addition to their function approximation capabilities, QNNs have also been shown to be capable of representing and quantifying the uncertainty inherent in the training data. More specifically, QNNs can identify overlapping between classes due to their capacity of approximating any arbitrary membership profile up to any degree of accuracy. This chapter describes the application of QNNs in the classification and removal of bird-contaminated data recorded by a 1290-MHz wind profiler [8, 9]. This is a nontrivial real-world problem that provides a reliable basis for testing the capabilities of QNNs and comparing their performance with that of conventional FFNNs.

11.2 Quantum neural networks

QNNs are decision-making and inferencing tools, which are capable of obtaining an approximate classification for uncertain data without any restricting assumptions such as the availability of *a priori* information in the form of a "desired" membership profile, limited number of classes of data, convexity of the classes, and so on [5, 14, 15]. QNNs are designed to achieve this goal through multi-level partitioning of the feature space. The capacity of QNNs for autonomously forming multi-level partitions of the feature space arises from their ability to create graded internal representations of the sample information provided by the training data. The sample information is encoded into graded internal representations by choosing multi-level activation functions for the hidden units, instead of the conventional sigmoid activation functions. If all the activation functions of the hidden units have the ability to form 'graded' partitions, then these partitions can be 'collapsed-in' or 'spread-out' as required, using a suitable learning algorithm.

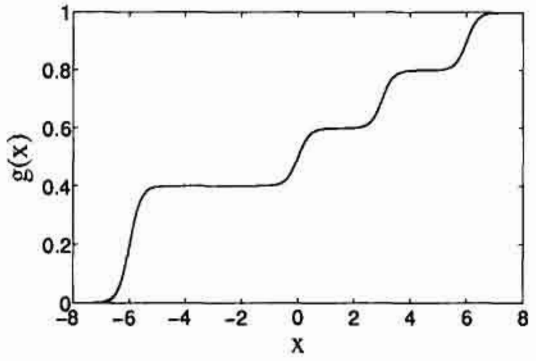


Fig. 11.1 Example of a multi-level activation function

11.2.1 QNN models

Consider a QNN consisting of n_i inputs, one layer of n_h multi-level hidden units, and n_o output units. The output units can be linear or sigmoidal. Let w_{ij} be the synaptic weight connecting the ith output unit to the jth hidden unit. Let the synaptic weight connecting the jth hidden unit to the ℓth input be $v_{j\ell}$. Suppose the data set $\mathcal{X}$ contains the feature vectors $\mathbf{x}_k = [x_{1,k}\ x_{2,k}\ \ldots\ x_{n_i,k}]^T$, $1 \leq k \leq M$. Then the input to the jth hidden unit from the kth feature vector $\mathbf{x}_k$ is $\bar{h}_{j,k} = \sum_{\ell=0}^{n_i} v_{j\ell}\, x_{\ell,k}$, with $x_{0,k} = 1, \forall k$. Suppose a multi-level hidden unit has n_s discrete quantum levels. Then its activation function can be written as a superposition of n_s sigmoid functions, each shifted by θ^r, *i.e.*,

$$g(x) = \frac{1}{n_s} \sum_{r=1}^{n_s} g_0(\beta_h(x - \theta^r)), \tag{11.1}$$

where $g_0(\cdot)$ is a sigmoid function, β_h is a slope factor, and $\{\theta^r\}$ define the jump-positions in the activation function. A typical example of a sigmoid function is the logistic function $g_0(x) = 1/(1 + \exp(-x))$, which was used in this work. Figs. 11.1 shows a multi-level activation function formed as the superposition of five logistic functions with jump-positions at $-6, -6, 0, 3, 6$ and a slope factor $\beta_h = 5$ each. The step widths of the multi-level activation function, called the quantum intervals, are determined by the jump-positions $\{\theta^r\}$. Therefore, the response of the jth multi-level hidden unit to the kth feature vector $\mathbf{x}_k$ can be written as

$$\tilde{h}_{j,k} = \frac{1}{n_s} \sum_{r=1}^{n_s} h_{j,k}^r = \frac{1}{n_s} \sum_{r=1}^{n_s} g_0(\beta_h(\bar{h}_{j,k} - \theta_j^r)). \tag{11.2}$$

If $n_s = 1$ and $\theta_j^1 = 0, \forall j$, then each multi-level activation function reduces to the sigmoid function $g_0(\cdot)$. In such a case, $\tilde{h}_{j,k} = g_0(\beta_h\, \bar{h}_{j,k})$ and the QNN model under consideration, denoted here as QNN n_i–$n_h(n_s)$–n_o, reduces to a conventional FFNN, which is denoted as FFNN n_i–$n_h(1)$–n_o. The input to the ith output unit from the kth feature vector $\mathbf{x}_k$ is $\bar{y}_{i,k} = \sum_{j=0}^{n_h} w_{ij}\, \tilde{h}_{j,k}$, with $\tilde{h}_{0,k} = 1, \forall k$. Therefore, the response of the ith output unit to the kth feature vector can be written as $\hat{y}_{i,k} = f(\bar{y}_{i,k})$, where $f(x) = g_0(\beta_o\, x)$ if the unit is sigmoidal and $f(x) = x$ if the unit is linear.

11.2.2 A gradient descent learning algorithm for QNNs

Let $\mathbf{y}_k = [y_{1,k}\; y_{2,k}\; \cdots\; y_{n_o,k}]^T$ be the desired output vector for the kth feature vector $\mathbf{x}_k$ and also let $\hat{\mathbf{y}}_k = [\hat{y}_{1,k}\; \hat{y}_{2,k}\; \cdots\; \hat{y}_{n_o,k}]^T$ be the actual output vector. A gradient-descent-based algorithm for learning the synaptic weights of the QNN can be derived by minimizing the error function

$$E_k = \frac{1}{2}\sum_{i=1}^{n_o}(y_{i,k} - \hat{y}_{i,k})^2, \tag{11.3}$$

sequentially for $k = 1, 2, \dots, M$. The synaptic weight w_{ij} connecting the ith output unit to the jth hidden unit can be updated as [14, 15]

$$w_{ij,k} - w_{ij,k-1} = \alpha\, \varepsilon^o_{i,k}\, \tilde{h}_{j,k}, \tag{11.4}$$

where $w_{ij,k-1}$ and $w_{ij,k}$ are the values of w_{ij} before and after the adaptation for the kth input, α is the learning rate, and

$$\varepsilon^o_{i,k} = f'(\bar{y}_{i,k})\,(y_{i,k} - \hat{y}_{i,k}), \tag{11.5}$$

with $f'(\bar{y}_{i,k}) = \beta_o\, \hat{y}_{i,k}\,(1 - \hat{y}_{i,k})$ if the ith output unit is sigmoidal and $f'(\bar{y}_{i,k}) = 1$ if the ith output unit is linear. The synaptic weight $v_{j\ell}$ connecting the jth hidden unit to the ℓth input can be updated as [14, 15]

$$v_{j\ell,k} - v_{j\ell,k-1} = \alpha\, \beta_h\, \varepsilon^h_{j,k}\, x_{\ell,k}, \tag{11.6}$$

where $v_{j\ell,k-1}$ and $v_{j\ell,k}$ are the values of $v_{j\ell}$ before and after the adaptation, and

$$\varepsilon^h_{j,k} = \left(\frac{1}{n_s}\sum_{r=1}^{n_s} h^r_{j,k}\,(1 - h^r_{j,k})\right)\sum_{i=1}^{n_o}\varepsilon^o_{i,k}\, w_{ij}. \tag{11.7}$$

If $n_s = 1$ and $\theta^1_j = 0$, $\forall j$, then (11.7) gives

$$\varepsilon^h_{j,k} = \tilde{h}_{j,k}\,(1 - \tilde{h}_{j,k})\,\sum_{i=1}^{n_o}\varepsilon^o_{i,k}\, w_{ij}, \tag{11.8}$$

with $\tilde{h}_{j,k} = g_0(\beta_h\, \bar{h}_{j,k})$, and the algorithm described above reduces to the well-known error backpropagation algorithm, which was developed for training conventional FFNNs.

The quantum intervals can be estimated by minimizing the class-conditional variances at the outputs of the hidden units. The variance of the output of the

jth hidden unit for the mth class $\mathcal{C}_m$ is given by

$$\sigma_{j,m}^2 = \sum_{\forall \mathbf{x}_k \in \mathcal{C}_m} (\langle \tilde{h}_{j,\mathcal{C}_m} \rangle - \tilde{h}_{j,k})^2, \tag{11.9}$$

where $\langle \tilde{h}_{j,\mathcal{C}_m} \rangle = (1/|\mathcal{C}_m|) \sum_{\forall \mathbf{x}_k \in \mathcal{C}_m} \tilde{h}_{j,k}$, and $|\mathcal{C}_m|$ denotes the cardinality of $\mathcal{C}_m$. The adaptation of the parameters $\{\theta_j^r\}$ is based on the minimization of the objective function formed by summing $\sigma_{j,m}^2$ over all the classes and all the hidden units, *i.e.*,

$$G = \frac{1}{2} \sum_{j=1}^{n_h} \sum_{m=1}^{n_o} \sigma_{j,m}^2 = \frac{1}{2} \sum_{j=1}^{n_h} \sum_{m=1}^{n_o} \sum_{\forall \mathbf{x}_k \in \mathcal{C}_m} (\langle \tilde{h}_{j,\mathcal{C}_m} \rangle - \tilde{h}_{j,k})^2. \tag{11.10}$$

The update equation for $\{\theta_j^r\}$ can be derived as [14, 15]

$$\Delta\theta_j^r = \eta \frac{\beta_h}{n_s} \sum_{m=1}^{n_o} \sum_{\forall \mathbf{x}_k \in \mathcal{C}_m} (\langle \tilde{h}_{j,\mathcal{C}_m} \rangle - \tilde{h}_{j,k}) (\langle \nu_{j,\mathcal{C}_m}^r \rangle - \nu_{j,k}^r), \tag{11.11}$$

where $\langle \nu_{j,\mathcal{C}_m}^r \rangle = (1/|\mathcal{C}_m|) \sum_{\forall \mathbf{x}_k \in \mathcal{C}_m} \nu_{j,k}^r$, and $\nu_{j,k}^r = h_{j,k}^r (1 - h_{j,k}^r)$.

The QNN is trained according to the algorithm described above in a sequence of adaptation cycles. Each adaptation cycle involves the adaptation of all the internal parameters of the network, that is, the synaptic weights and the locations $\{\theta_j^r\}$ of the shifted and superimposed sigmoid functions of the hidden units. Since the criterion employed for updating the parameters $\{\theta_j^r\}$ is based on all the input vectors from the training set, $\{\theta_j^r\}$ are updated after the presentation of all the inputs to the network and the corresponding adaptation of the synaptic weights.

11.3 Wind profilers

Wind profilers are vertical pulsed Doppler radar systems developed for measuring the three-dimensional wind field. Wind profilers are used for the general study of wind fields, research, and airport needs. The height coverage of a wind profiler strongly depends on the frequency used. A 1290-MHz wind profiler, such as that used for this study, covers a height range from about 100 m to 5000 m above ground level. Reliable winds can be obtained with a time resolution of 10 minutes to 60 minutes and a height resolution of 30 m to 1500 m. The total height range of the wind profiler is divided into sections, called range gates, that define the vertical resolution of the wind profiler. The pulse length limits the widths of the range gates. A pulse length of 400 ns corresponds to a vertical resolution of 60 m, while a pulse length of 2800 ns corresponds to 400 m. The operational mode corresponding to a 400 ns pulse is usually referred to as the low mode while the operational mode corresponding to a 2800 ns pulse is referred to as the high mode. For larger pulse lengths more power can be emitted and greater detection heights can be reached.

11.3.1 The bird removal problem

Erroneous wind profiler data can occur due to ground clutter, inhomogeneous wind fields, precipitation, heavy storms, external electromagnetic noise, and moving objects hitting beams or side lobes (e.g., migrating birds, traffic or trees moved by winds). When the wind profiler operates at a frequency of 1290 MHz, corresponding to a wavelength of 23.5 cm, birds can be considered as approximate Rayleigh scatterers with strong reflectivity due to their high water content [17]. According to several observations, migrating birds can affect data recorded by wind profilers operating in the 1000 MHz range [8, 11, 16, 17]. In presence of birds, the true atmospheric wind echoes are being distorted by the bird signals to an extent that only the birds' speed can be seen in the wind profiler data.

Bird contamination occurs during more than 160 nights per year in Central Europe. This leads to a contamination of about 10% of the annual half-hourly or hourly wind data [8].

11.3.2 Wind profiler data acquisition and processing

All wind data used in this project were recorded by a Radian LAP$^{(R)}$–3000 wind profiler at the site of Payerne, Switzerland. Bird contamination was verified by simultaneous measurements made during periods of heavy migration by the wind profiler and an Inframetrics LORIS IRTV–445L infrared system, operated by the Swiss Ornithological Institute at Sempach. The infrared system has a beamwidth of 1.4° and can detect birds up to about 3000 m above ground level. Fig. 11.2 shows a schematic beam configuration of the wind profiler and the infrared system employed in this project. Note that for all experiments in this study the wind profiler and the infrared were operated using a fixed beam direction. Only data from birds appearing in both observing systems were classified as bird data. Because birds typically migrate at night [2] and usually avoid rain [2, 17], measurements that contain virtually no birds were

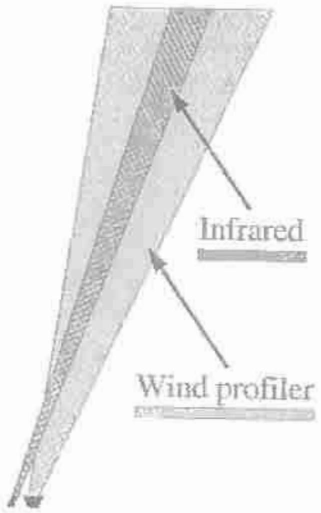


Fig. 11.2 Schematic beam configuration of the wind profiler and the infrared system employed for this study

recorded during day time and during rain events.

Processing of the wind profiler data begins with a over 128 pulses combined with DC removal, which is applied to increase the signal-to-noise ratio (SNR) and reduce the amount of data for the fast Fourier transform (FFT) which follows. After transforming the averaged time-domain data into the frequency domain data by FFT, a spectral averaging is done. The spectral average is usually calculated over about 50 spectra. In the next processing stage, the averaged noise level is calculated and ground clutter is removed. Ground clutter can be described as radar echo of objects in rest (*e.g.*, trees, houses) and appears in the spectral data as symmetric zero-velocity peaks. Data processing is completed by the selection of the most significant peak and the calculation of several characteristic features (*e.g.*, moments) of the most significant peak. Usually the most significant spectral peak is selected to be that with the highest power density. In operational mode, moments are calculated every 30 seconds. Using two pulse lengths for five beam directions, a full cycle where all directions and pulse lengths are alternately measured takes 5 minutes. Six or twelve cycles are averaged to produce half-hourly or hourly winds, respectively.

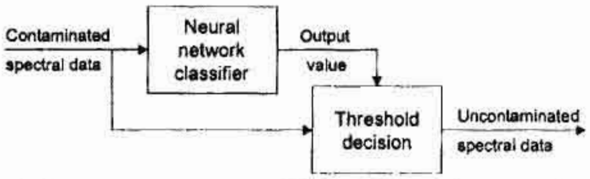

Contaminated spectral data
Neural network classifier
Output value
Threshold decision
Uncontaminated spectral data

Fig. 11.3 Block diagram representation of the bird removal system. A decision device identifies and removes single spectra by thresholding the response of a neural network trained to classify single spectra contaminated by birds

11.4 Formulation of the bird removal problem

Bird removal was attempted in this approach by rejecting time-averaged spectra. In such a case, all wind information contained in the rejected time period is lost. The amount of data lost through this process can be minimized by decreasing the averaging time. In this study, bird removal was accomplished by identifying and eliminating bird-contaminated single spectra, that is, spectra obtained by FFT after time-domain averaging over one-second intervals. The single spectra contaminated by migrating birds were identified in this study by thresholding the response of a neural network trained to separate birds from true wind data. All experiments were performed on a single beam direction of the wind profiler in an attempt to verify whether this formulation can provide the basis for developing a system operating on several beam directions. Following the removal of bird-contaminated single spectra, the remaining spectra can be averaged to obtain uncontaminated hourly or half-hourly winds. The bird removal approach is described by the block diagram shown in Fig. 11.3.

The removal of bird-contaminated spectra must deal effectively with uncertainty caused by ambiguous data samples that could correspond to birds or true winds. There are cases where a spectrum can not be identified by human experts to be exclusively caused by birds, precipitation, or turbulence. The existence of ambiguous data samples can be attributed to weather conditions and the observational tools employed for data recording and processing. Since the beamwidth of the infrared system ($1.4°$) is narrower than the beamwidth of the wind profiler ($6°$) (see Fig. 11.2), only data from birds crossing the infrared beam were selected to represent the class "birds." Moreover, only situations where virtually no bird was present in the air were selected to represent the class "no birds." Finally, the training set did not include birds traversing the wind profiler beam at other locations or birds being caught in side lobes.

11.4.1 Input Feature Selection

This study relied on a set of 14 potential input features, which was composed based on the results of previous studies [3, 8]. The 14 input features can be divided into the following groups:

Moments: The moments calculated from the spectral data produced by the wind profiler include the radar signal power, the spectral width, the skewness and kurtosis of the most significant peak. The frequency limits ν_1 and ν_2 of the integrals involved in the calculation of the moments were determined by the intersection of the noise level of the spectra and the envelope of the most significant peak.

- Radar signal power:

$$P = \frac{M_0}{N_F}, \tag{11.12}$$

where N_F is the number of spectral points within the spectrum, $M_0 = \int_{\nu_1}^{\nu_2} S'(\nu)\, d\nu$ is the area below the most significant peak, with $S'(\nu)$ obtained in terms of the spectral power density $S(\nu)$ and the noise level P_N as $S'(\nu) = S(\nu) - P_N$.

- Spectral width:

$$W_S = \sqrt{\frac{1}{M_0} \int_{\nu_1}^{\nu_2} (\nu - M_1)^2\, S'(\nu)\, d\nu}, \tag{11.13}$$

where $M_1 = \frac{1}{M_0} \int_{\nu_1}^{\nu_2} \nu\, S'(\nu)\, d\nu$ is the first moment of the most significant peak.

- Skewness:

$$S = \frac{1}{M_0\, W_S^3} \int_{\nu_1}^{\nu_2} (\nu - M_1)^3\, S'(\nu)\, d\nu. \tag{11.14}$$

- Kurtosis:

$$K = \frac{1}{M_0\, W_S^4} \int_{\nu_1}^{\nu_2} (\nu - M_1)^4\, S'(\nu)\, d\nu - 3. \tag{11.15}$$

The higher moments, skewness and kurtosis, are measures of the asymmetry and flatness of a spectral distribution (*i.e.*, of a spectral peak) relative to the Gaussian distribution.

Additional Echo Signal Features: The additional echo signal features extracted from the wind profiler data include the signal-to-noise ratio (SNR) and the height.

- Signal-to-noise ratio:

$$SNR = \frac{P}{P_N}, \tag{11.16}$$

where P is the radar signal power and P_N is the noise level.
- Height:

$$H = H_0 + n_G\, S_G, \tag{11.17}$$

where H_0 is the first gate height, n_G the gate number, and S_G is the vertical gate spacing.

Averaged Signal Power Variances: This set of features contains the signal power variances averaged in time and height.

- Time-averaged signal power variance:

$$\langle P_{\text{var}}\rangle_T = 1 - \frac{\left(\frac{1}{N_T}\sum_{t=1}^{N_T}|P(t)|\right)^2}{\frac{1}{N_T}\sum_{t=1}^{N_T}P^2(t)}, \tag{11.18}$$

where the time average was taken over $N_T = 5$ samples.
- Height-averaged signal power variance:

$$\langle P_{\text{var}}\rangle_H = 1 - \frac{\left(\frac{1}{N_H}\sum_{h=1}^{N_H}|P(h)|\right)^2}{\frac{1}{N_H}\sum_{h=1}^{N_H}P^2(h)}, \tag{11.19}$$

where the height average was taken over $N_H = 5$ samples.

Relative Features: This set of features contains the relative signal power differences in time and height.

- Time-relative signal power difference:

$$\langle P_{\text{diff}} \rangle_T = \frac{\frac{1}{N_T} \sum_{t=1}^{N_T} P^2(t)}{\overline{P}^2} - 1. \tag{11.20}$$

- Height-relative signal power difference:

$$\langle P_{\text{diff}} \rangle_H = \frac{\frac{1}{N_H} \sum_{h=1}^{N_H} P^2(h)}{\overline{P}^2} - 1. \tag{11.21}$$

Time Averages of Higher Moments: This set of features contains the time averages of the skewness and kurtosis.

- Time-averaged skewness:

$$\langle S \rangle_T = \frac{1}{N_T} \sum_{t=1}^{N_T} S(t). \tag{11.22}$$

- Time-averaged kurtosis:

$$\langle K \rangle_T = \frac{1}{N_T} \sum_{t=1}^{N_T} K(t). \tag{11.23}$$

Profiler-dependent Features: This set contains two features that describe the operating mode of the wind profiler, namely, the vertical gate width W_G, which is determined by the radiated signal pulse length, and the vertical gate spacing S_G, which is often set to be the same as the vertical gate width.

11.4.2 Training, testing and validation sets

The neural networks were trained and tested using a normalized version of the input features, produced by replacing each feature sample x by $\bar{x} = (x - \mu_x)/\sigma_x$, where μ_x and σ_x denote the mean and standard deviation of this feature calculated over 120000 single spectra representing all cases (*i.e.*, birds, rain, and clear air). Since there were only two classes of interest for this study, namely "birds" and "no birds," classification was performed by neural networks with only one output unit. The output value 1 was defined to represent the class "birds" and the output value 0 represented the class "no birds." A neural network classifier with a sigmoidal output unit produces continuous output values between 0 and 1. Thus, a final cut-off threshold has to be chosen to determine whether a bird is present or not (see Fig. 11.3).

The independent data sets composed for this project can be divided into three groups:

- Training set: Data used for training the neural networks.
- Validation set: Data used to select the best network size and to prevent overtraining.
- Testing set: Data set used to evaluate and to compare the performance of the different classifiers.

Table 11.1 shows the composition of the data sets used in the experiments. All neural networks were trained on the training set, which consisted of clear air, rain and bird-contaminated data recorded in both modes of the wind profiler. The bird-contaminated data were 10% of the total amount of data included in the training set while over 25% of the total amount of data in the training set were data recorded in rain. These percentages indicate that the bird-contaminated and rain data were clearly over-represented in the training set with respect to their annual distribution. The validation set originated from the same data source as the training set. In fact, both training and validation sets were formed from this data source by randomly selecting individual samples. The testing set contained data that originated from independent sources and was divided in three subsets. In contrast to the other data sets, the testing set 1 represents a collection of four independent data sets, namely clear air in high and low mode and rain in high and low mode. The testing set 2 represents a random selection of 200 individual birds identified by the infrared during a migration period. The testing set 3 represents two 200-second periods of wind profiler measurements recorded in high mode and low mode during a migration event. The number of samples included in the testing set 3 differ for the two modes because the high mode covers fewer range gates than the low mode.

11.5 Experimental results

This study began with a preliminary evaluation of various feature sets and neural networks trained to identify birds from wind profiler data recorded in high mode and low mode. Over 600 FFNNs and QNNs of different sizes were trained using the five sets of features summarized in Table 11.2. The feature set "14" includes all features described in Section 11.4. The feature set "10m"

Table 11.1 Composition of the training, validation and testing data sets. All values shown represent the number of wind profiler samples, except for the values corresponding to the "Testing Set 2" which represent the number of observed birds

	High Mode			Low Mode		
	Air	Rain	Birds	Air	Rain	Birds
Training Set	2500	1000	375	2500	1000	375
Validation Set	2500	1000	375	2500	1000	375
Testing Set 1	38000	21000		36000	21500	
Testing Set 2			100			100
Testing Set 3	2200			3000		

does not include any higher moments, while the set "10t" does not contain any time-averaged features. The feature set "6" consists of the radar signal power, the spectral width, and all profiler-dependent parameters. The feature set "5" contains the spectral moments or features that are directly related to the shape of the most significant peak.

The neural networks tested in these experiments were trained by gradient descent for 800 or 2000 adaptation cycles. In case of overtraining, the training process was terminated based on the performance of the trained networks on the validation set. The weights of all networks trained in these experiments were updated using a learning rate $\alpha = 0.07$ with momentum equal to 0.05. The quantum levels of the QNNs were determined by updating the jump-positions using a learning rate $\eta = 0.02$. The slope values β_h of the sigmoid functions were set equal to the number n_s of quantum levels in order to achieve a clear and equally scaled separation of the superimposed sigmoids.

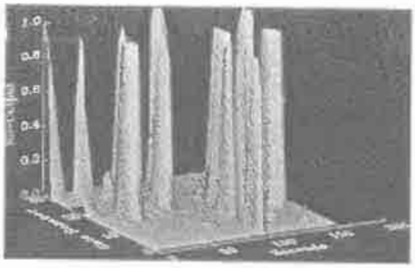
The best QNNs and FFNNs trained in these experiments produced similar classification rates when tested on the validation and testing sets. However, there were significant performance differences in the ability of trained QNNs and FFNNs to handle ambiguous data. These performance differences can be made clear by visualizing the responses of the trained neural networks in three or two dimensions.

Table 11.2 Composition of the input feature sets utilized for training the neural networks

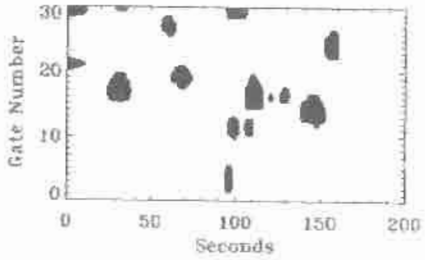
	Set Name				
Features	14	10m	10t	6	5
P	●	●	●	●	●
W_S	●	●	●	●	●
S	●		●		●
K	●		●		●
SNR	●	●	●	●	●
H	●	●	●	●	
$\langle P_{\text{var}} \rangle_T$	●	●			
$\langle P_{\text{var}} \rangle_H$	●	●	●		
$\langle P_{\text{diff}} \rangle_T$	●	●			
$\langle P_{\text{diff}} \rangle_H$	●	●	●		
$\langle S \rangle_T$	●				
$\langle K \rangle_T$	●				
S_G	●	●	●	●	
W_G	●	●	●	●	

11.5.1 Bird visualization

Fig. 11.4 allows the visualization of the classification results produced by the trained QNN 5–2(5)–1 on the low-mode data included in the testing set 3. According to the notation followed in this chapter, the QNN 5–2(5)–1 is a QNN with five inputs, two 5–level hidden units and one output unit. Note that the number of inputs (5 in this case) also reveals the set of features used for training. For example, the QNN 5–2(5)–1 was trained using the features included in the feature set "5." In the three-dimensional (3–D) visualization shown in Fig. 11.4(a), the axis toward the right is the time in seconds while the axis toward the left is the number of gates (*i.e.*, height). Note that in low mode, 10 gates correspond to 600 m above ground level. The axis perpendicular to the time-gate plane is the network output. The gray surface shown in Fig. 11.4(a) represents the network output as a function of time and height. In this particular case, there is a clear distinction between the two classes



(a)



(b)

Fig. 11.4 Bird visualization produced by the QNN 5–2(5)–1 on low-mode data: (a) Three-dimensional visualization of the outputs of the QNN 5–2(5)–1. (b) Two-dimensional bird visualization produced by thresholding the outputs of the QNN 5–2(5)–1 (threshold value: $\theta = 0.2$)

"birds" (corresponding to network output 1) and "no birds" (corresponding to network output 0). The responses of the trained neural networks can also be visualized in two dimensions after thresholding. Thresholding was performed individually on each sample (*i.e.*, each time-height point of Fig. 11.4) based on the response of the trained neural networks to the input features representing this sample. The threshold value was selected to guarantee the elimination of the largest possible amount of bird-contaminated data, while producing reliable bird visualizations. Fig. 11.4(b) allows the two-dimensional (2–D) visualization of the outputs of the QNN 5–2(5)–1 produced by thresholding its response. The outputs of the neural network exceeding a certain threshold value θ (0.2 in this particular case) are shown in black color.

According to the visualization procedure outlined above, birds appear as spikes in Fig. 11.4(a) and as patches in Fig. 11.4(b). A bird traversing the wind profiler is expected to produce a smooth pattern with respect to the time axis. Thus, the ability of trained neural networks to handle ambiguous data can be evaluated using as a criterion the smoothness and compactness of the bird-patches. The appearance of bird-patches looking "rough" or "random" reveals that the trained neural network is not capable of effectively dealing with ambiguous data. On the other hand, a neural network capable of handling ambiguous data not involved in its training is expected to produce smooth and compact bird-patches.

The visualization of birds was used to evaluate the feature sets used for classification and the performance of the neural networks trained in these experiments. Because it appeared to be more difficult to find appropriate thresholds for high mode than for low mode, the evaluation was initially restricted to high mode. The visualization of birds was evaluated on 200 seconds of bird-contaminated data recorded in high mode during a heavy migration period (*i.e.*, the high-mode data of the testing set 3). The experiments indicated that the neural networks trained using any time-averaged or height-averaged input features (such as those included in the feature sets "14," "10m," and "10t") tend to smear out birds and produce "blurred" visualization results. A typical example of such a visualization is shown in Fig. 11.5(a), which was produced by the FFNN 14–14(1)–1. The higher moments (skewness and kurtosis) seem to be essential for an accurate detection of birds, as indicated by the poor visualization produced by the neural networks using the feature sets "10m" and "6" that do not include these features. This is clear from Fig. 11.5(b), which shows the 2–D visualization produced by the FFNN 6–3(1)–1. Fig. 11.5(b) is a typical example of an "unreliable" visualization. The best visualization of birds in high mode was produced by neural networks trained using the feature set "5." Fig. 11.5(c) shows the 2–D visualizations of birds in high mode produced by the FFNN 5–3(1)–1, which exhibited the best performance among all the FFNNs trained in these experiments (including the FFNN 5–2(1)–1). The QNN 5–2(3)–1 and QNN 5–2(5)–1 produced the best visualization results in high mode among all the neural networks tested in these experiments. Fig. 11.5(d) shows the 2–D visualization produced by the QNN 5–2(5)–1, which achieved slightly higher classification rates on the validation set than the QNN 5–2(3)–1. Moreover, the QNN 5–2(5)–1 produced rounder and more compact bird-patches than those produced by the FFNN 5–3(1)–1 as indicated by comparing Figs. 11.5(d) and 11.5(c). It is also remarkable that there were significant performance differences among the FFNN 5–2(1)–1, the QNN 5–2(3)–1, and the QNN 5–2(5)–1, which were all trained using the same features and contained the same number of hidden units. Given that the FFNN 5–2(1)–1 can be considered as a QNN with a single quantum level per hidden unit, the superior performance of the QNN 5–2(3)–1 and the QNN 5–2(5)–1 can only be attributed to the fact that the QNNs consisted of multi-level hidden units whose jump-positions were specifically updated to capture the structure of the feature space.

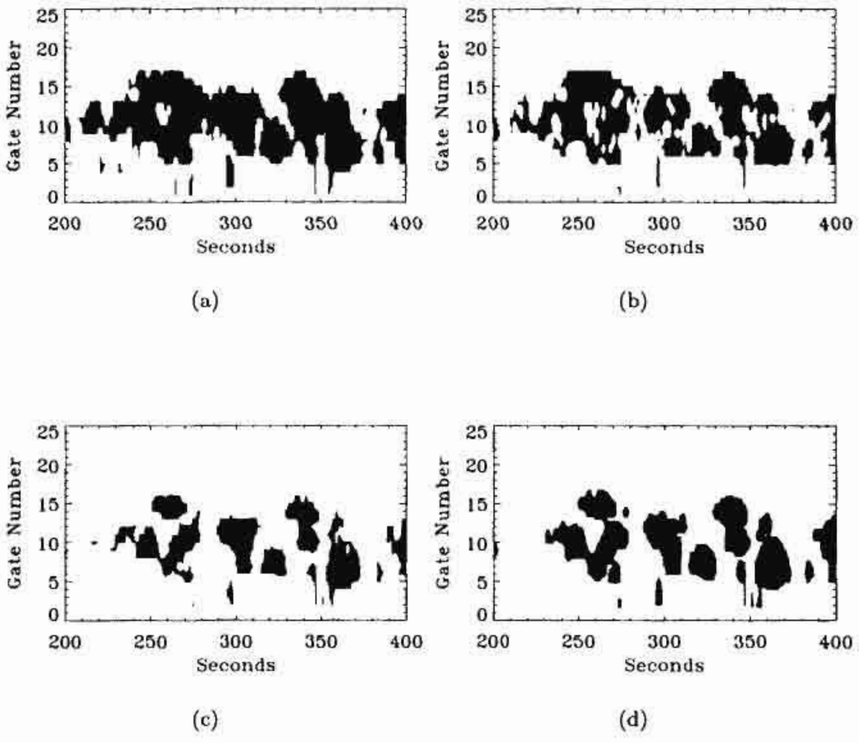


Fig. 11.5 Two-dimensional visualization of the outputs produced on high-mode data by: (a) the FFNN 14–14(1)–1 (threshold value: $\theta = 0.99$), (b) the FFNN 6–3(1)–1 (threshold value: $\theta = 0.99$), (c) the FFNN 5–3(1)–1 (threshold value: $\theta = 0.99$), and (d) the QNN 5–2(5)–1 (threshold value: $\theta = 0.91$)

Table 11.3 Detection and visualization of birds produced on the high-mode data included in the testing set 3 by thresholding the responses of the best FFNNs and QNNs

Network	Size	θ	Bird Data [%]	Bird Visualization
FFNN	5–3(1)–1	0.99	19.0	Average
		0.97–0.93	22.5–24.9	Average
FFNN	5–2(1)–1	0.99	7.5	Too few birds
		0.97–0.95	22.6–22.4	Average
QNN	5–2(3)–1	0.99	16.4	Good
		0.97–0.95	16.7–17.3	Very Good
QNN	5–2(5)–1	0.99–0.97	17.6–18.1	Good
		0.95–0.91	18.3–18.7	Very Good

11.5.2 Detecting birds in high mode

This evaluation began by selecting the trained neural networks that produced the best visualization of birds. Based on the previous evaluation of the feature sets, only networks trained using the feature set "5" were evaluated in this set of experiments. Table 11.3 shows the percentage of spectra removed due to the detection of single birds for different threshold values and the evaluation of the corresponding visualization of birds in high mode. Decreasing the threshold values increased the percentage of single spectra removed due to their classification as bird-contaminated. On the other hand, the visualization of birds became increasingly blurred as the threshold values decreased. The FFNN 5–3(5)–1, which produced the best visualization among all the FFNNs when tested with a threshold value of $\theta = 0.99$, removed 19% of single spectra as bird-contaminated. The QNN 5–2(3)–1 and QNN 5–2(5)–1 produced the best visualization among all the trained neural networks when tested with threshold values 0.97–0.95 and 0.95–0.91, respectively. Compared with the QNN 5–2(3)–1, the QNN 5–2(5)–1 removed a higher percentage of single spectra. The two QNNs that exhibited the best overall performance in high mode were also tested in terms of their performance on the high-mode data included in the testing set 1. Note that this particular set contains virtually no bird-contaminated data. The results of these experiments are summarized in

Table 11.4 Bird detection and classification rates produced on the high-mode data of the testing set 1 by thresholding the responses of the two neural networks that led to the best bird visualization in high mode

Network	Size	θ	Bird	HM [%]	
			Data [%]	Clear Air	Rain
QNN	5–2(3)–1	0.97	16.7	96.78	99.92
		0.95	17.3	96.73	99.92
QNN	5–2(5)–1	0.95	18.3	96.72	99.92
		0.91	18.7	96.67	99.91

Table 11.4. The two QNNs performed equally well on high-mode data recorded in rain but the QNN 5–2(3)–1 outperformed slightly the QNN 5–2(5)–1 on the high-mode data recorded in clear air. Table 11.4 indicates that both QNNs classified correctly almost all high-mode data recorded in rain and classified incorrectly about 3% of the high-mode data recorded in clear air.

11.5.3 Detecting birds in low mode

The detection of birds in low-mode data focused on the FFNN and QNN that achieved the best detection and visualization in high mode, namely the FFNN 5–3(1)–1 and the QNN 5–2(5)–1. Table 11.5 summarizes the percentage of single spectra selected for removal by the two networks tested with different thresholds on 200 seconds of wind profiler measurements during a migration period (*i.e.*, the low-mode data of the testing set 3). Table 11.5 also shows the classification rates achieved by the two neural networks on the low-mode data included in the testing set 1, which contained virtually no bird-contaminated data. The evaluation of the two neural networks listed in Table 11.5 indicated that the best overall performance was achieved when the FFNN 5–3(1)–1 and the QNN 5–2(5)–1 were both tested with a threshold value of $\theta = 0.2$. Testing the networks with a threshold value of $\theta = 0.1$ resulted in the identification of almost the same amount of bird-contaminated data but led to inferior visualization. Both neural networks achieved very high classification rates on low-mode data recorded in clear air regardless of the threshold value used for classification. However, the performance of both neural networks on low-mode data recorded in rain was significantly lower.

Table 11.5 Percentage of bird-contaminated data and classification rates produced on the low-mode data included in the testing set 1 by thresholding the responses of the best FFNN and QNN

Network	Size	θ	Bird	LM [%]	
			Data [%]	Clear Air	Rain
FFNN	5–3(1)–1	0.1	7.5	99.00	68.06
		0.2	6.8	99.23	70.91
QNN	5–2(5)–1	0.1	8.9	97.61	65.92
		0.2	6.7	99.30	71.62

11.5.4 The amount of bird-contaminated data

The ability of the QNN 5–2(5)–1 to reliably detect bird-contaminated spectra was evaluated by testing its performance on the testing set 2, which contained 200 birds observed by the infrared in high mode and low mode. The thresholds used were $\theta = 0.91$ and $\theta = 0.2$ for high-mode and low-mode data, respectively. The QNN 5–2(5)–1 identified 97% of the birds observed by the infrared, which indicates that it is a reliable tool for detecting bird-contaminated wind profiler data. In an attempt to evaluate the system under a "worst case scenario," the QNN 5–2(5)–1 classifier was tested on bird-contaminated data recorded during a night of heavy migration. The data recorded in both high and low mode were examined in 30-minute intervals for bird contamination at all gate ranges (*i.e.*, heights). The highest percentage of bird-contaminated data found in all 30-minute intervals at all heights was 76%. Note that this percentage refers to a single 30-minute interval and does not represent the average amount of bird-contaminated data. According to the evaluation on the testing tests, the QNN 5–2(5)–1 was capable of removing almost all bird-contaminated spectra while hardly affecting the data containing pure wind measurements. The wind profiler used for data acquisition and preprocessing computed single spectra every two seconds. This implies that the wind profiler would average about 900 single spectra every 30 minutes if none of the spectra was removed as bird-contaminated. In the worst case scenario, where the QNN 5–2(5)–1 removed almost 76% of bird-contaminated spectra, the wind profiler was allowed to average almost 200 single spectra in a time interval of 30 minutes. Averaging 200 single spectra every half hour is estimated to produce acceptable half-hourly

Table 11.6 Classification rates produced on the testing set 1 by thresholding the response of the QNN 5–2(5)–1 trained on the bird-enriched training set

Network	Size	θ	Clear Air [%]		Rain [%]	
			HM	LM	HM	LM
QNN	5–2(5)–1	HM: 0.3	92.25	98.83	95.76	82.20
		LM: 0.4	96.84	98.93	96.59	93.74

winds for most situations encountered in practice.

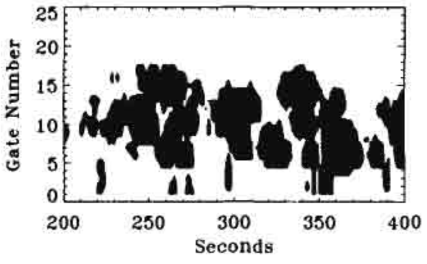
The overall classification rate reached by the QNN 5–2(5)–1 can be raised above 90% by applying the following procedure:

- check for birds in the high mode;
- if there is a bird detected in high mode, then check for birds in low mode.

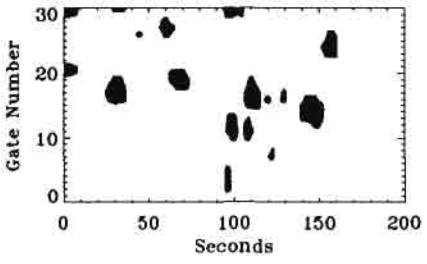
This procedure can be justified by the fact that the high mode of a wind profiler covers larger volumes than the low mode, while the high mode is generally more sensitive to birds [9]. As a result, it is not necessary to check for birds in low mode if there is no bird detected in the high mode. Given that birds usually avoid rain [2], the overall classification rate is not likely to be affected by the rather low classification rates on low-mode data recorded in rain. Even during periods of heavy migration, the QNN 5–2(5)–1 appeared to be able to produce a sufficient amount of clear air data for a reliable averaging. This implies that the system can be used to refine wind measurements recorded by the wind profiler even in periods of heavy migration.

11.5.5 Dealing with several weather conditions

The previous experiments indicated that the performance of all trained neural networks (including the QNN 5–2(5)–1) was relatively low for high-mode data recorded in clear air and low-mode data recorded in rain. It was hypothesized that the low performance of the QNN 5–2(5)–1 classifier is mainly due to a non-representative training set. This hypothesis was tested by expanding the training set and the validation set using a portion of the data from testing set 1. More specifically, the expanded training and validation sets included additional high-mode data recorded in clear air and low-mode data recorded in



(a)



(b)

Fig. 11.6 Bird visualization produced by QNN 5–2(5)–1 trained using the bird-enriched training and validation sets on (a) high-mode data and (b) low-mode data

rain. The QNN 5–2(5)–1 trained using the expanded training and validation sets performed well on bird-free data but failed to detect bird-contaminated spectra. This experimental outcome can be attributed to the fact that the expansion of both training and validation sets by 2500 bird-free spectra resulted in a reduced representation of bird-contaminated data in both sets. This was remedied by increasing the amount of bird-contaminated data in the training set and the validation set four times. Retraining the QNN 5–2(5)–1 using the bird-enriched training and validation sets resulted in a slight degradation of its performance on high-mode data recorded in clear air but improved significantly its performance on low-mode data recorded in rain (see Table 11.6). Figs. 11.6(a) and 11.6(b) show the 2–D bird visualization produced by testing the retrained QNN 5–2(5)–1 on high-mode and low-mode data, respectively.

Table 11.7 Classification rates produced on the testing set 1 by thresholding the response of the QNN 6s–3(5)–1 trained using the "6s" set of features on the bird-enriched training set

Network	Size	θ	Clear Air [%]		Rain [%]	
			HM	LM	HM	LM
QNN	6s–3(5)–1	HM: 0.6	94.95	99.41	98.75	97.41
		LM: 0.4	86.64	99.04	92.80	93.82

11.5.6 Improving the set of input features

The experiments indicated that the trained neural networks achieved their best performance when tested with different thresholds on high-mode and low-mode data. This experimental outcome indicates that bird detection is significantly affected by the operating mode of the wind profiler. Nevertheless, the feature set "5" that led to the best classifiers contains no features yielding information about the operating mode of the wind profiler. This observation motivated the use of the feature set "6s," which was constructed by adding to the feature set "5" the vertical gate width W_G, a quantity that depends heavily on the operating mode of the wind profiler. The feature set "6s" was utilized for training and testing a variety of neural networks on the bird-enriched training and validation sets described in the previous section. Table 11.7 summarizes the performance of the QNN 6s–3(5)–1, which achieved the best performance among the neural networks tested in these experiments using the feature set "6s." More specifically, Table 11.7 shows the classification rates obtained by testing the QNN 6s–3(5)–1 on the low-mode and high-mode data from the testing set 1. The QNN 6s–3(5)–1 outperformed the QNN 5–2(5)–1 when tested with appropriate thresholds on high-mode and low-mode data from the testing set 1. Figs. 11.7(a) and 11.7(b) show the 2–D bird visualization produced by the QNN 6s–3(5)–1 on wind profiler data from testing set 3 recorded in high mode and low mode, respectively. Comparison of Figs. 11.6 and 11.7 indicates that the QNN 6s–3(5)–1 produced comparable visualization of birds in high mode than the QNN 5–2(5)–1. In conclusion, an accurate detection of bird-contaminated data can be accomplished by utilizing the feature set "6s" and using well-balanced training and validation sets that represent all relevant weather conditions.

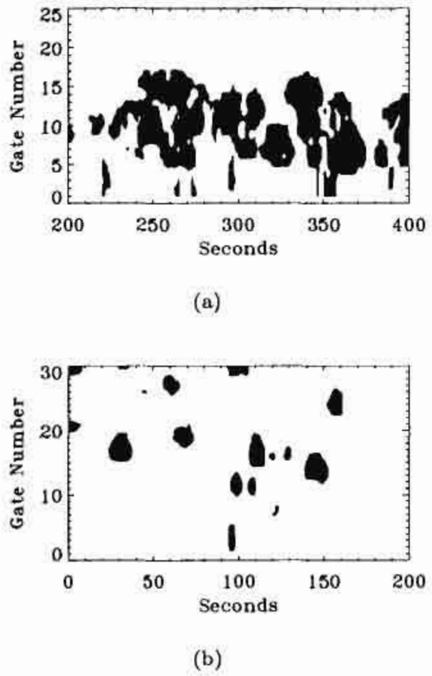


Fig. 11.7 Bird visualization produced by the QNN 6s–3(5)–1 trained using the "6s" set of features on (a) high-mode data and (b) low-mode data

11.6 Conclusions

This chapter presented the development, testing and evaluation of an automated bird removal system relying on QNNs. The study involved the selection of the best set of input features, the selection of the most reliable neural network models and sizes, and the criteria employed for identifying birds in wind profiler data recorded in high mode and low mode. It also focused on the selection of representative training, validation and testing sets and also evaluated the effect of weather conditions on the performance of the system.

The experimental study indicated that the most reliable feature set involved features directly related to the shape of the corresponding spectral peak and the vertical gate width of the wind profiler. In general, feature sets that did not include the skewness and kurtosis failed to produce reliable classification results. On the other hand, all feature sets including time or height averaged features tended to smear out the visualization of birds, with a negative impact on bird detection. The experiments revealed that the error rates on the training and validation sets are not reliable performance indicators for this particular application. The most reliable criterion for bird removal was found to be the visualization of birds in two dimensions, which can be used to assess the ability of trained neural networks to handle ambiguous data. The experiments indicated that bird visualization and detection in high mode and low mode required different thresholds. The best among the FFNNs and QNNs tested in this study achieved comparable classification on the training and validation sets. However, the best QNNs produced bird visualization results that were superior to those produced by the best FFNNs. Compared to FFNNs, the best among the QNNs tested in the experiments were found to be more reliable in detecting bird-contaminated data. These performance differences can be attributed to the inherent ability of QNNs to detect the presence of uncertainty in the training set and quantify the existing uncertainty by approximating any membership profile from sample data. The price to be paid for this performance gain is the extra computational effort for updating the jump-positions of the multi-level hidden units.

Acknowledgments

The authors express their gratitude to Dominique Ruffieux from MeteoSwiss, Switzerland, Felix Liechti, Thomas Steuri and Bruno Bruderer from the Swiss Ornithological Institute at Sempach, Switzerland, and Thomas Griesser from the Institute for Atmospheric Science at the ETH, Zurich, Switzerland.

References

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

[2] B. Bruderer, "The study of bird migration by radar, part 2: Major achievements," *Naturwissenschaften* **82**, 45–54 (1997).

[3] S. Haykin and C. Deng, "Classification of radar clutter using neural networks," *IEEE Transactions on Neural Networks* **2**, 589–600 (1991).

[4] H. Ishibuchi and H. Tanaka, "Approximate pattern classification using

neural networks," in *Fuzzy Logic: State of the Art*, R. Lowern and M. Roubens, Eds., Kluwer Academic, Dordrecht, The Netherlands, 1993.

[5] N. B. Karayiannis and G. Purushothaman, "Fuzzy pattern classification using feed-forward neural networks with multilevel hidden units," *Proceedings of IEEE International Conference on Neural Networks*, Orlando, FL, June 28–July 2, 1994, pp. 1577–1582.

[6] N. Kasabov and R. Kozma, (Eds), *Neuro-Fuzzy Techniques for Intelligent Information Processing*, Springer-Verlag, Heidelberg, 1999.

[7] R. Kozma *et al*, "Adaptive neuro-fuzzy signal processing system using structural learning with forgetting," *Intelligent Automation and Soft Computing* **1**, 389–404 (1995).

[8] R. Kretzschmar, *The Effect of Migrating Birds on 1290-MHz Wind Profiler Data*, Semester Thesis, Institute for Atmospheric Science (LAP-ETH), ETH Zurich, Switzerland, April 1997.

[9] R. Kretzschmar, *Quantum Neurofuzzy Bird Removal Algorithm (NEUROBRA) for 1290-MHz Wind Profiler Data*, Master's Thesis, Institute for Atmospheric Science (LAPETH), ETH Zurich, Switzerland, August 1998.

[10] M. Leshno *et al*, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks* **6**, 861–867 (1993).

[11] P. A. Miller *et al*, "The extent of bird contamination in the hourly winds measured by the NOAA profiler network: Results before and after implementation of the new bird contamination quality control check," *First Symposium on Integrated Observing Systems*, Long Beach, CA, February 2–7, 1997, pp. 138–144.

[12] S. K. Pal and S. Mitra, "Multilayer Perceptron, fuzzy sets, and classification," *IEEE Transactions on Neural Networks* **3**, 683–697 (1992).

[13] S. K. Pal and S. Mitra, *Neuro-Fuzzy Pattern Recognition*, Wiley, New York, 1999.

[14] G. Purushothaman and N. B. Karayiannis, "Quantum neural networks (QNNs): Inherently fuzzy feedforward neural networks," *IEEE Transactions on Neural Networks* **8**, 679–693 (1997).

[15] G. Purushothaman and N. B. Karayiannis, "Feed-forward neural architectures for membership estimation and fuzzy classification," *International Journal of Smart Engineering System Design* **1**, 163–185 (1998).

[16] J. M. Warnock *et al*, "Multiple frequency profiler studies of echoes

observed during bird migration," *Proceedings of the 27th Conference on Radar Meteorology*, Vail, CO, October 9–13, 1995, pp. 284–286.

[17] J. M. Wilczak *et al*, "Contamination of wind profiler data by migrating birds: Characteristics of corrupted data and potential solutions," *Journal of Atmospheric and Oceanic Technology* **12**, 449–467 (1995).

Chapter 12

NETWORKS OF SPIKING NEURONS IN DATA MINING

K. Cios and D. M. Sala

Department of Bioengineering
University of Toledo
Toledo, OH 43606-3390, U.S.A.
e-mail: *kcios@eng.utoledo.edu*

Abstract

The spiking neuron model, in spite of being a more realistic representation of the biological neuron, has not been widely used as a part of data mining tools for knowledge discovery that are based on artificial neural networks. This has been mainly on account of its complexity and the lack of learning rules and methods for adapting it to solving real problems. Spiking neurons have been used for finding clusters in data, without the usual requirement of specifying *a priori* the number of clusters, for finding associations in the data, and for solving some graph optimization problems. This has been made possible by the introduction of a new temporal correlation learning rule. Having solved graph optimization problems by networks of spiking neurons, we have highlighted the great potential and applicability of the approach to data mining.

12.1 Introduction

Artificial neural networks are one of the most widely used data mining tools in the process of knowledge discovery. The vast majority of them use average firing rate model of a neuron. On the other hand, the spiking neuron model that closely resembles its biological counterpart is usually used only for modeling purposes. This is due partly to its complexity as well as lack of learning rules and methods for using them in networks of spiking neurons for solving real problems. In this chapter, we describe how spiking neurons can be used for finding clusters in data, without the usual requirement of specifying a priori the number of clusters, for finding associations in the data, and for solving some graph optimization problems. We made it possible by introducing new temporal correlation learning rule that is explained in the chapter. Since artificial neural networks are usually represented in the form of a graph, the relationship between the two has been explored by some researchers, for example for solving the traveling salesman problem [8]. More recently, Bose and Liang [1] overviewed networks that use average firing rate neuron model and graph theory. With the rising interest in networks of spiking neurons, which take into consideration timing information, the relationship between graphs and neural networks can be seen from a different perspective, which we present in the chapter. We solve some graph optimization problems by using networks of spiking neurons. The results show their great potential and applicability to data mining. Below the building blocks of networks of spiking neurons are introduced.

12.1.1 Spiking neuron model

We use the integrate-and-fire spiking neuron model, described by the following differential equation defining the transmembrane potential [9]:

$$\tau \frac{dE}{dt} = -E + R(I^{syn} + I^{ext}), \tag{12.1}$$

where E is the transmembrane potential, I^{syn} is the synaptic current input, I^{ext} is the external current input, $t = RC$ is the membrane time constant, and R and C are, respectively, the resistance and capacitance of the membrane. Whenever the membrane potential E surpasses the threshold, the neuron fires. The interaction between neurons is described by postsynaptic potentials (PSPs). For each synapse between neurons i and j the postsynaptic potential is described by a delta function (other more complex functions can be considered) [13]:

$$psp_{ij}(t) = K\delta(t - \Delta t_{ij}), \tag{12.2}$$

where Δt_{ij} is the propagation time and K is a scaling factor. We assume that a synapse is activated whenever a spike generated by the pre-synaptic neuron reaches it. The total synaptic input of a neuron is obtained by summing up all postsynaptic potentials:

$$I_j^{syn}(t) = \Sigma_i w_{ij} \cdot psp_{ij}(t), \tag{12.3}$$

where w_{ij} represents the strength (weight) of the synapse between neurons i and j.

12.1.2 Synaptic learning rule for spiking neurons

Let us consider neuron j that receives input from n synapses with efficacies, or weights, $w_{ij}(t)$ $(1 \leq i \leq n)$. The relative timing between pre- and post-synaptic spikes induces the change of $w_{ij}(t)$. The quantity describing the time correlation between pre- and post-synaptic spikes is given by the correlation coefficient, defined below, that is also illustrated in Fig. 12.1 [14]:

$$c_{corr} = (1 + y)\exp(-k \cdot \frac{t_{ij}^2}{t_{corr}^2}) - y, \tag{12.4}$$

where t_{ij} is the time elapsed between the firing of neuron i and the firing of neuron j, $y \leq 1$ is the maximum value for the decay and $k = ln(1 + 1/y)$ is a scalar that assures that the zero crossing of c_{corr} takes place at t_{corr}, which is the time window for positive correlation. Whenever a postsynaptic neuron fires, the connection weights of all pre-synaptic neurons that excited it are modified according to the formula:

$$w_{ij} = \frac{w_{ij} + \alpha \cdot c_{corr}(t_{ij})}{\Sigma_i (w_{ij} + \alpha \cdot c_{corr}(t_{ij}))}, \tag{12.5}$$

where $w_{ij}(t)$ is the synaptic weight from neuron i to j, α is the learning rate (a small quantity), c_{corr} is the correlation value. We will refer to this rule as the temporal correlation rule. If the correlation value is positive, the synaptic strength increases and if it is negative it decreases. This type of dependence is supported by biological experiments [10, 11]. The rule is competitive as some neurons firing within a time interval are rewarded and other neurons are

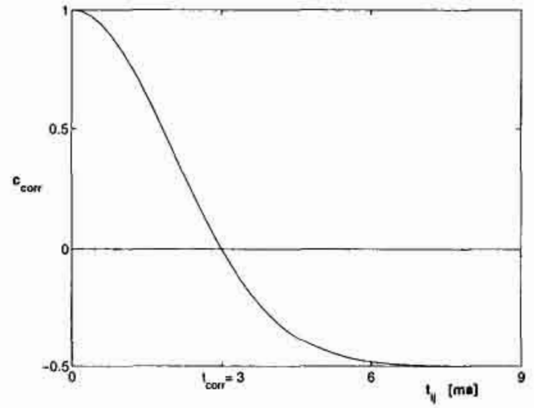


Fig. 12.1 Correlation value c_{corr} for $t_{corr} = 3$ ms and $y = 0.5$

penalized. Although the formula also works for negative t_{ij}, the utilization of the rule is time-order dependent. For neurons which are pre- and post-synaptically connected to each other, the order of spiking is essential as one connection will strengthen and the other weaken depending on which neuron fired first.

12.2 Graph algorithms

As mentioned above, artificial neural networks are usually represented in the form of a graph. Since networks of spiking neurons take timing information into consideration, the relationship can be seen in a different perspective. To that end, we introduce several optimization problems that are representable by graphs and whose solutions involve finding paths of a specific type. The spiking neuron model is the same as described above and the learning is performed by modifying the synaptic strength, w_{ij} according to our temporal correlation rule, given by (12.5).

12.2.1 Shortest path through a graph

The shortest path problem is one of the most commonly encountered problems among all classes of network optimization problems. Examples include transportation, communications, and production applications. Simply stated, the shortest path problem deals with finding the shortest route between a source and a destination in a network [4, 6]. Suppose that a transportation company has to deliver a package from one location to another in the shortest time. Assuming that the time depends only on the distance, the problem can be approached by representing the highway network as a graph. The nodes represent highway intersections and edges represent the interconnecting highways, as shown in Fig. 12.2. The solution to the problem is the shortest path through the graph from source to destination. The graph in Fig. 12.2 can be interpreted as a neural network where each circle represents a neuron node and each edge (we consider only positive edges) represents a connection between two neuron

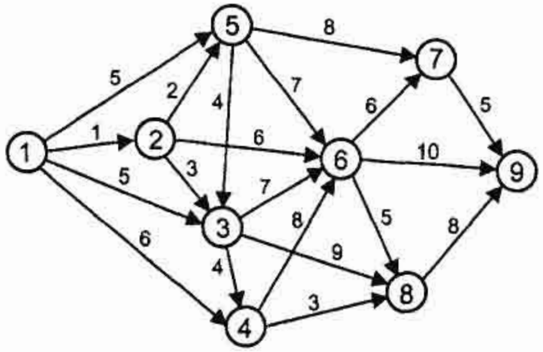


Fig. 12.2 Highway network represented as a graph. Node 1 is the source node and Node 9 is the destination node

nodes [14]. A neuron node consists of a pair of excitatory-inhibitory neurons arranged in such a way that each firing of the excitatory neuron activates the paired inhibitory neuron, which in turn inhibits the excitatory neuron for a period of time that is longer than the longest path in a given graph. With this arrangement each neuron node, henceforth called neuron, will fire only once. In the following we assume that a synapse is activated whenever a spike generated by the pre-synaptic neuron reaches it. The value shown on each edge represents the delay, t_{ij}, between the firing of neuron i and the time the synapse between neurons i and j is activated. Initially, the weights of all synapses are equal and sufficiently large so that a single postsynaptic potential can evoke a spike in the postsynaptic neuron. Learning is done in cycles according to the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \cdot c_{corr}(t_{ij}). \tag{12.6}$$

This equation is the same as (12.5) but without normalization. A cycle starts by initiating a spike in the input neuron, in our example the leftmost neuron, and ends when all the neurons in the network have fired. After a number of cycles, which depends on the value of the learning rate, some connections become stronger, up to the saturation value M, while other connections decrease and disappear altogether. Redrawing the network as shown in Fig. 12.3 we observe that the resultant network represents the initial network in which only the shortest paths from the input neuron to all other neurons have survived. Further examination shows that each neuron has only one entry that has survived. It

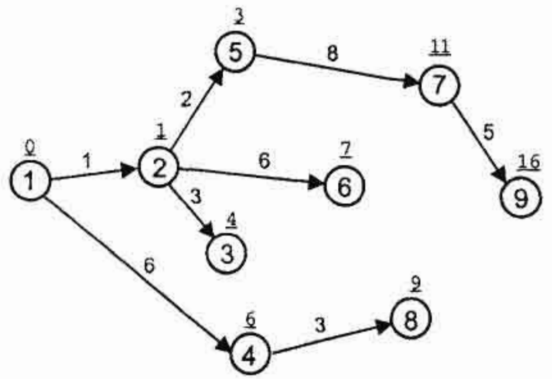


Fig. 12.3 The resulting network: only the surviving connections are shown; the underlined values represent the times of firings of the neurons

is possible that for some neurons more than one entry survived which is an indication of the existence of multiple solutions or alternative routes. During the learning process, whenever a PSP arrives at a synapse that synapse is activated and learning takes place. The first synapse activated is the one that determines the firing of the postsynaptic neuron. If a synapse is activated within a time interval t_{corr}, calculated backwards from the time of firing of the post-synaptic neuron, its weight increases, otherwise it decreases. For a small t_{corr} only the first activated synapse will be rewarded and the remaining ones will be penalized. If the learning rate is close to the value of one then the shortest path can be obtained in one cycle. In general, one starts with a large value of the correlation time, t_{corr}, which is decreased as the learning progresses. By doing so we obtain additional information regarding the evolution of the weights. Let us consider node 3, shown in Fig. 12.2; it has three input edges of lengths 5, 3 and 4. The evolution of the corresponding synapses in the network is shown in Fig. 12.4. We notice that the edge labeled 3 is the one corresponding to the shortest path to neuron 3, while the path through the edge labeled 5 is the next shortest. The path through the edge labeled 4 is the longest. Thus, more

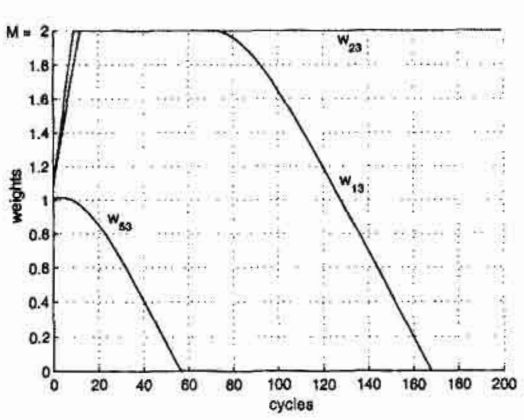


Fig. 12.4 Input weights evolution of neuron 3; t_{corr} starts at 3 ms and exponentially decays to zero

information is found by looking at the evolution of t_{corr}, which starts at 3 ms and decays exponentially towards zero. This type of analysis may be useful for large networks where the shortest path, next to the shortest path, etc., can provide additional useful information. In the above example we imposed the direction of the routes to simplify the explanation but in fact any route can be traveled in either direction. To account for this we just add arcs in the opposite directions with the same values. The algorithm would proceed in the same way using the new graph. It is worth mentioning that if the graph has cycles it does not influence the performance of the algorithm.

12.2.2 Minimal spanning tree

Another example of network optimization problem is the minimal spanning tree [12, 15]. For example, a phone company has to provide new service to five new customers. Due to costs of installation, the company is interested in finding the minimum cable length necessary to connect the new customers. We consider the same graph in Fig. 12.2 as the possible layout of the cable network, where neuron 1 represents the exchange office providing the service. The edges of the graph represent all possible connections between any two locations and their values the necessary cable length. The minimal spanning tree of the graph gives the solution to this problem. Again, we associate with the graph a network of spiking neurons [3]. The nodes are the neurons and the edges are the connections between the neurons with propagation times given by the edge values, with their strengths being initially set to a large value. The algorithm works in steps. At each step we stimulate a neuron, or a group of neurons, and determine the neuron that is closest by using the same method as in the shortest path algorithm. The additional requirement is that any neuron that fires, except the ones stimulated at the beginning, inhibits all other neurons. For instance initially we start by stimulating neuron 1, since it represents the exchange office, and find the next neuron to fire. In our example neuron 2 is the closest and thus it fires first, inhibiting all other neurons. Repeating this step for a number of cycles the strength of the connection from neuron 1 to neuron 2 increases gradually to the maximum value, and the reversed connection between the two neurons gradually decreases to 0, which means that there is no connection. The next step involves stimulating neurons 1 and 2 (found at the previous step) together and finding the neuron that fires first. In our example it is neuron 5. Repeating this step for a number of cycles the strength of the connection from neuron 2 to neuron 5 increases gradually to the maximum value. The connection from neuron 1 and those from neuron 5 to neurons 2 and 1 decrease to 0. At each step we start by stimulating the neurons found at the previous step and determine the neuron that is closest to any of them. The process stops after a number of steps equal to the number of neurons. Fig. 12.5 shows the strengthened connections at each step together with those from previous steps. It is easy to see that the surviving edges in the final network represent the minimal spanning tree.

12.3 Clustering

Suppose that we are interested in finding how the nodes of a given graph cluster together based on their edge distances. Let us consider the graph shown in Fig. 12.2, and associate with it a neural network in the same way

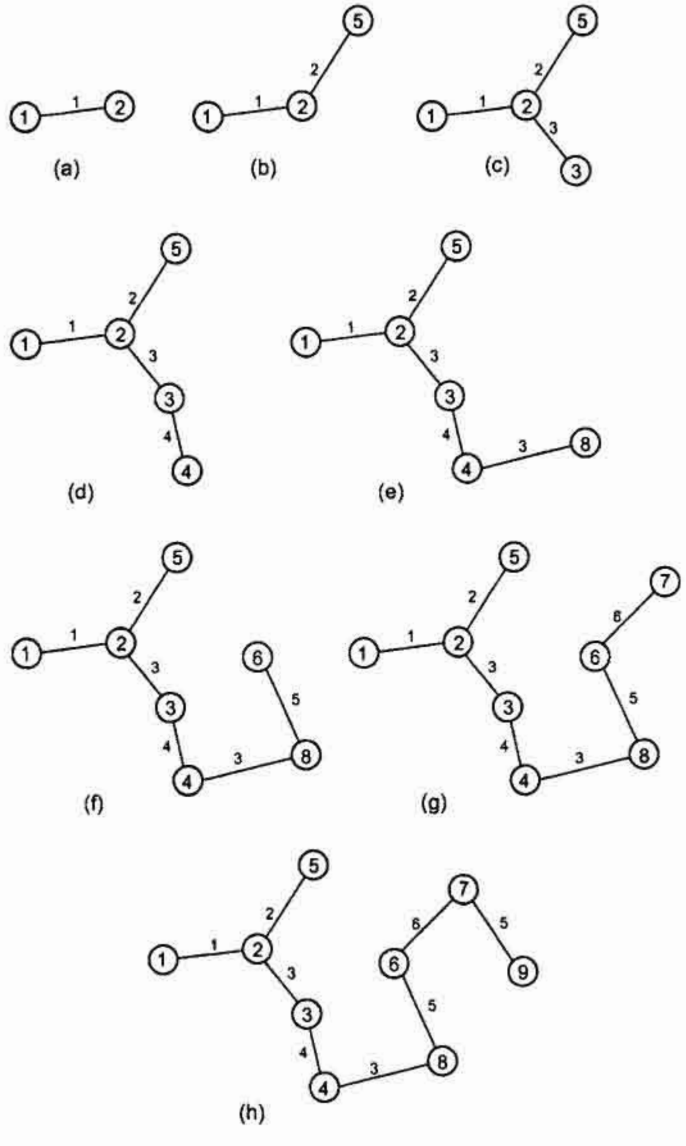


Fig. 12.5 Finding the closest neuron at each step. At step (h) the process stops and the result is the minimal spanning tree

we did for the minimal spanning tree. This time we change the stimulation procedure. Learning is cyclical. Each cycle consists in direct stimulation, in random order, of all neurons. At each step the closest neuron, *i.e.*, the one that fired first after the stimulated neuron, increases its connection strength with the stimulated neuron. Thus they start to cluster together. After a number of cycles the connection strengths between the closest neurons become saturated while the connection strengths between neurons farther apart decrease to zero. The shortest edges saturate the fastest and the process continues until each neuron is clustered with at least one other neuron. In our example it resulted in three clusters as shown in Fig. 12.6(a), where only the surviving edges are shown. The connections between neurons are bidirectional and the arrows on the surviving edges show the direction of the association. The algorithm emulates the single linkage clustering method or nearest neighbor algorithm [2, 6]. In general, in hierarchical clustering methods, like the single linkage, one starts with calculating minimal distance between all elements. The elements that correspond to the minimal distance are then clustered together. Next, the procedure is repeated with the elements clustered previously treated as one element. The distance from this new element to any other element is the minimum distance between the elements of the cluster and that element. In our network the clustering is done simultaneously for all the elements. It is easy

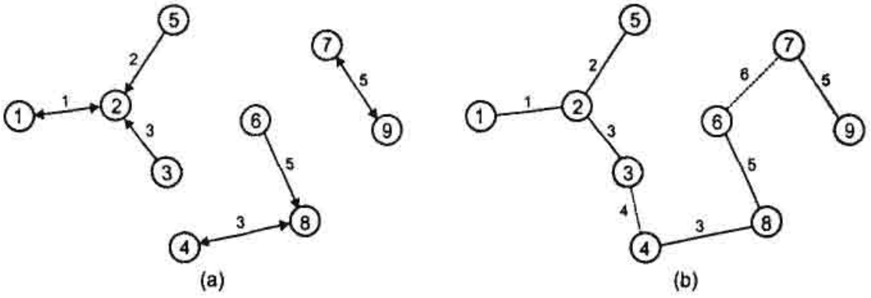


Fig. 12.6 (a) Final network showing clustering of neurons. The clustering process results in tree clusters; (b) The shortest edges between the clusters, shown as dotted lines, complete the minimal spanning tree

to notice that the surviving edges in the clustered network form a subset of the minimal spanning tree. To obtain the minimal spanning tree one needs to add one more phase, in which the obtained clusters play the role of single neurons. The temporal correlation learning rule still applies. Now, however, instead of choosing a single neuron we select an entire cluster. This means that all the neurons within that cluster are activated simultaneously at the beginning of a learning cycle. The rest of the process continues in exactly the same way as for the original clustering phase described above. Thus, as our network has clustered into three clusters, at each step we activate simultaneously the neurons of a cluster. The neurons from the remaining clusters will be activated, in the order of their connection delays, and the strength of connections will be modified according to the temporal correlation learning rule. Repeating the cycle for a number of times we obtain the minimal spanning tree shown in Fig. 12.6(b) where the connections between the clusters are drawn with dotted lines. We can check that the result is the same as the previous one.

Table 12.1 Project's activities and durations

Activity	Immediate Predecessors	Duration
A	-	10
B	-	1
C	-	5
D	A	9
E	A	8
F	A	10
G	D	3
H	D	4
I	B,G	5
J	C,E	7
K	B,G	4
L	C,E	3
M	F,H,I,J	8

12.4 Critical path method

Problems encountered in many areas require careful planning and scheduling. In order to make a problem, or a project, tractable usually it is divided into a small set of sub-projects called activities. The activities are inter-related so that some activities cannot start until some other activities are completed. An example of such a project is given in Table 12.1. It lists activities along with their inter-relationships and duration.

The project can be represented as a graph, or project network. The project network describes the sequence of activities necessary for completion of the project. A directed arc represents each activity, while nodes represent events. An event is completed only when all activities directed to it have been completed. The representation given in Fig. 12.7 is used, where each edge is labeled by the corresponding activity and its duration [5]. Nodes S and T represent the start and finish of the project, respectively. The goal is to determine the earliest time in which the project can be completed. The solution is given by the critical path, the longest path in the network from start to finish. In order to determine the critical path let us consider a neural network of spiking neurons associated with the graph shown in Fig. 12.7. The nodes represent neurons and the edges represent the connections. The edge values represent propagation delays of spikes from one neuron to another. We assume that the modification of the membrane potential of a neuron caused by arrival of

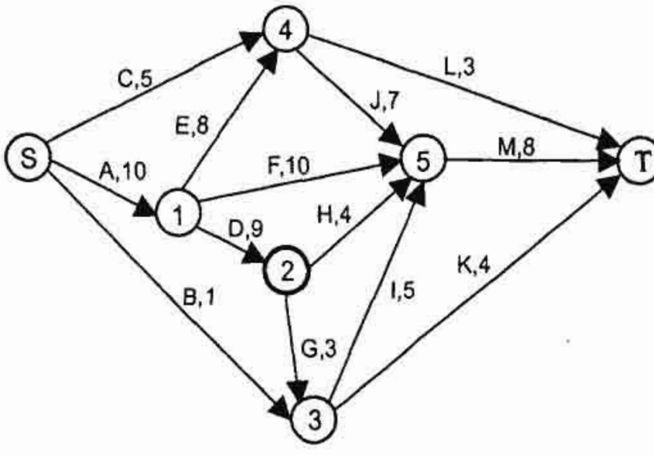


Fig. 12.7 Project network; S - starting of the project and T - end of the project

a spike is the same, and that the membrane time constant is sufficiently large so that during the time between two subsequent spikes the potential change is insignificant. We also assume that the number of spikes necessary to activate a neuron is equal to the number of input connections. Now let us suppose that in the network previously described we stimulate neuron S at time $t = 0$. After a delay of 10 time units neuron 1 fires. The next neuron to fire is neuron 4. At time $t = 5$ neuron 4 is stimulated by the spike emitted by S, but this stimulation is not sufficient to activate it, because it requires two spikes. Neuron 4 is activated and fires only when the spike from neuron 1 arrives at $t = 18$. Similarly, we can analyze events at other neurons. Neuron 5 needs five spikes and fires after $t = 27$, and the output neuron T fires after $t = 35$, which is also the earliest time in which the project can be completed. Thus, we can interpret firing of each neuron as representing the time when the corresponding event is completed. Although we have determined the completion time of the events, we also want to determine the critical path, the path for which a delay of any event included in the path, delays the entire project. For this we employ the learning rule described by (12.5). The arrival of an input spike within t_{corr} to the time of firing of the neuron increases its weight, otherwise it decreases

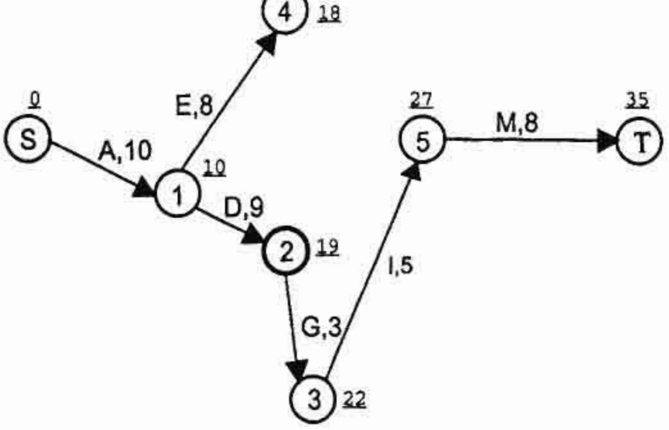


Fig. 12.8 The resulting network; only the surviving connections are shown; the underlined values represent the completion time

it. Learning is done in cycles with each cycle consisting of stimulating neuron S and waiting until neuron T fires. Narrowing t_{corr} as the learning proceeds, assures that only the neuron that sent the last spike before firing remains connected. The resulting network is shown in Fig. 12.8. The path connecting S to T represents the critical path: $S-1-2-3-5-T$, or it can be described in terms of activities: $A-D-G-I-M$. We have also obtained information about the completion time for all events, even if they are not critical events like event 4. Fig. 12.9 shows the evolution of the weights for neuron 5 during learning.

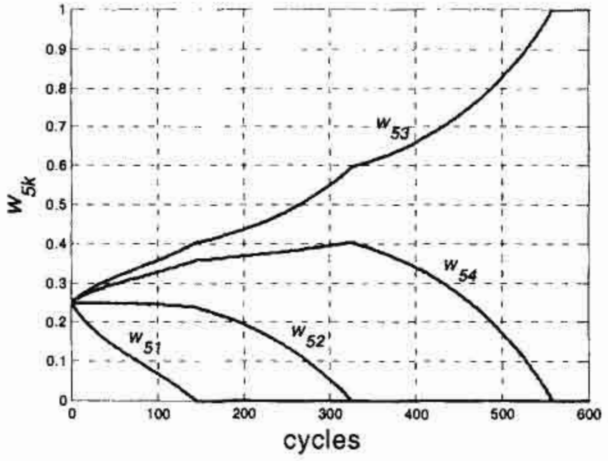


Fig. 12.9 Neuron number 5 evolution of weights during learning

12.5 The longest common subsequence

The last problem we consider is determination of the longest common subsequence. Given two sequences $X = \{x_1, x_2, \ldots x_m\}$ and $Y = \{y_1, y_2, \ldots y_n\}$ we want to find the common subsequence of maximal length. In this section we show how this problem can be solved using a neural network of spiking neurons and the results of the previous section. First we build an $m \times n$, two-dimensional network in such a way that every neuron is connected with all the neurons below and to the right of it. Fig. 12.10 shows the architecture of a 4×4 network with the connection showed only for the upper leftmost neuron. Each column is labeled in order with the elements of X, and each row is labeled in order with the elements of Y. The determination of the longest common

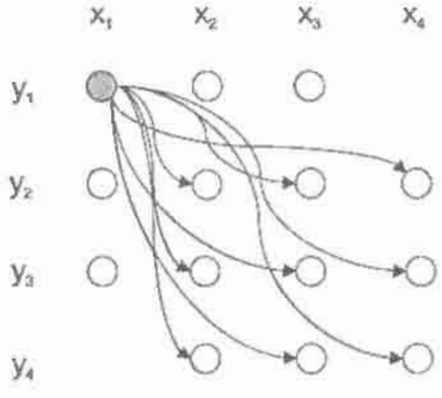


Fig. 12.10 Network layout

subsequence is done in two phases. In the first phase we determine all common subsequences of length greater than or equal to 2. This is done by firing, in order, the neurons corresponding to one of the sequences. Let us start with sequence Y. At each time step we fire the neurons in a row that have the same column label as the row. First, we fire the neurons corresponding to A in the first row of the network (see Fig. 12.11) and then the neurons corresponding to B in the second row; we continue until we finish all the rows. During this process the connection strength between two linked neurons that have both fired will increase while the connection between two linked neurons in which only one has fired will diminish, according to the temporal learning rule. The result is shown in Fig. 12.11, where only the surviving connections are shown. The gray-shaded neurons are the ones that have fired. Starting from any linked neuron that fired, and following its links we get a common subsequence of the initial sequences. The problem becomes then one of finding the maximal subsequence. In phase 2 we solve this problem by using the approach explained above for solving the critical path problem. We assume that all surviving connections have the same propagation delay (*i.e.*, in the equivalent graph all edges are equal). The resulting network from phase 1 will have a number of input neurons, *i.e.*, neurons that have no inputs, and a number of output neurons, *i.e.*, the neurons that have no outputs. In our example there are three input neurons and two output neurons. We start by stimulating all input neurons and proceed in the same way as explained in the critical path method. The last output neuron to fire belongs to the longest path in the net-

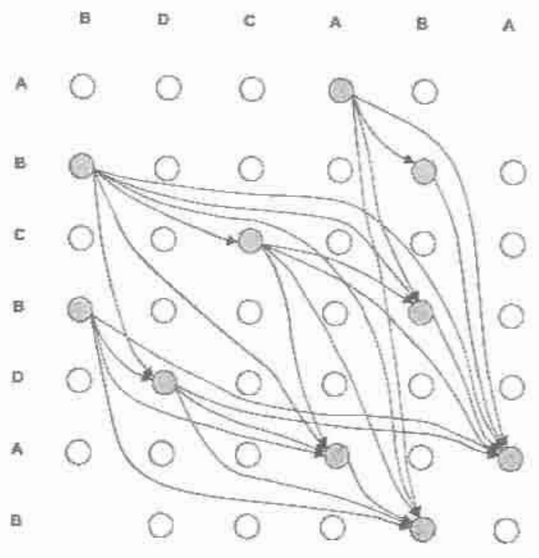


Fig. 12.11 Phase 1 surviving connections

work and following its surviving links gives us the longest subsequence, which is shown in Fig. 12.12. In our example the maximal length is 4 and the solution is not unique. In fact two output neurons (the black ones) fire at the same time meaning that both are part of a maximal length sequence. This is shown by the fact that there are more surviving paths in the network. The solution can be any of the following sequences: BCAB, BDAB and BCBA.

12.6 Conclusions

In the chapter we presented new findings in design and applications of artificial neural networks that use a biologically-inspired spiking neuron model. The used model is a point neuron with the interaction between neurons described by post-synaptic potentials. The synaptic plasticity is achieved by using a temporal correlation learning rule, specified as a function of time difference between the firings of pre- and post-synaptic neurons. By considering temporal coding, that represents distances on a given spatial configuration, we have shown how to associate graphs with networks of spiking neurons. By using the temporal correlation learning rule we solved the shortest path algorithm, clustering based

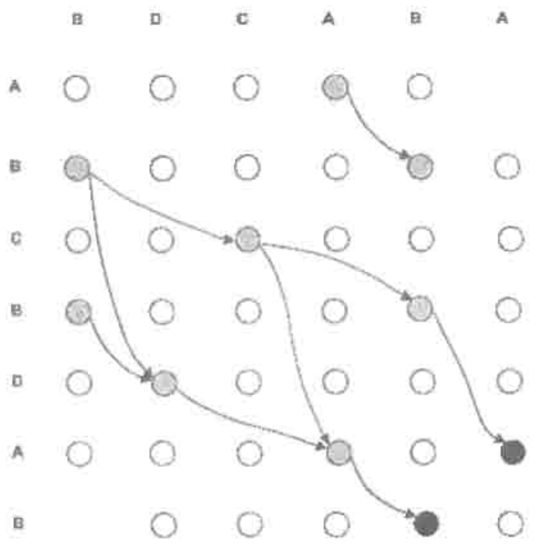


Fig. 12.12 Phase 2 surviving connections

on the nearest neighbor, the minimal spanning tree, and found the longest common subsequence of two sequences. The examples show that a network of spiking neurons emulates several graph algorithms. We showed how by using the temporal correlation rule certain associations between neurons in a network of spiking neurons can be implemented. Networks of spiking neurons, along with the temporal correlation rule, have a great potential for data mining, in particular in domains where very little is known about the data since the network can find the clustering structure on its own. As such it can serve as an unsupervised learning tool to be used before a supervised learning method is subsequently used on the data.

References

[1] Bose, N. K. and Liang, P. (1996) *Neural Network Fundamentals with Graphs, Algorithms, and Applications*, McGraw-Hill, New-York.

[2] Cios, K.J., Pedrycz, W., and Swiniarski, R.W. (1998) *Data mining. Methods for knowledge discovery*, Kluwer, Boston.

[3] Cios, K.J. and Sala, D.M. (2000) "Advances in applications of spiking neuron networks", SPIE AeroSense 14th Int. Symposium, Orlando,

April, in print

[4] Chartrand, G. and Oellerman, O.R. (1993) *Applied and Algorithmic Graph Theory*, McGraw-Hill, New-York.

[5] Evans, J.R. and Minieka, E. (1992) *Optimization Algorithms for Networks and Graphs*, 2nd edition, Marcel Dekker, New-York.

[6] Even, S. (1979) *Graph Algorithms*, Computer Science Press, Potomac, MD.

[7] Everitt, B.S. (1993) *Cluster Analysis*, 3rd edition, Edward Arnold, London.

[8] Hopfield, J.J. and Tank, D.W. (1985) "Neural computations of decisions in optimization problems", *Biological Cybernetics*, **52**, pp. 141-152.

[9] Kuhn, R. and Hemmen, J.L. (1995) "Temporal Association", *Models of Neural Networks I*, eds. E. Domany, J.L. van Hemmen, and K. Schulten, Springer Verlag, Berlin, pp. 201-218.

[10] Markram, H., Lubke, J., Forster, M., and Sakman, B. (1997) "Regulation of Synaptic Efficacy by Coincidence of postsynaptic APs and EPSPs", *Science*, **275**, pp 213-215.

[11] Markram, H. and Tsodyks, M. (1996) "Redistribution of synaptic efficacy between neocortical pyramidal neurons", *Nature*, **382**, pp. 807-810.

[12] Minieka, E. (1978) *Optimization Algorithms for Networks and Graphs*, Marcel Dekker, New-York..

[13] Sala, D.M., Cios, K.J. (1998) "Self-Organization in Networks of Spiking Neurons", *Australian Journal of Intelligent Information Processing Systems*, **5**, No 3, pp. 161-170.

[14] Sala, D.M., Cios, K.J. (1999) "Solving Graph Algorithms with Networks of Spiking Neurons", *IEEE Trans. on Neural Networks*, **10**, No 4, pp. 953-957.

[15] Taha, H.A. (1992) *Operations Research*, 5th edition, Macmillan, New-York.

Chapter 13

GENETIC ALGORITHMS, PATTERN CLASSIFICATION AND NEURAL NETWORKS DESIGN

S. Bandyopadhyay, C. A. Murthy and S. K. Pal

Machine Intelligence Unit
Indian Statistical Institute
203 B. T. Road, Calcutta - 700 035, INDIA
e-mail: {*sanghami,murthy,sankar*}*@www.isical.ac.in*

Abstract

Basic principles of genetic algorithms (GAs) along with the characteristic features and application domains are explained. The searching capability of GAs is exploited for modeling the class boundaries for pattern recognition problems. This is preceded by a discussion on the relevance of GAs to pattern recognition problems along with some attempts made in the related areas. Various classifiers using fixed string length GAs, variable string length GAs and GAs with chromosome differentiation are described. The superiority of these classifiers over the Bayes classifier (with assumption of multivariate normal distribution), k-NN rule and multilayer perceptron is demonstrated for both speech and remotely sensed image data. Based on an analogy between the classification principles of a multilayer perceptron (MLP) and the genetic classifier, a scheme for the automatic determination of the MLP architecture is presented and some experimental results are provided. Some of the results reported here are taken from the existing literature. An extensive bibliography is also provided.

13.1 Introduction

Genetic algorithms [14, 47] are randomized search and optimization techniques guided by the principles of evolution and natural genetics. The term was first mentioned by Bagley in 1967 [3], when he devised a genetic algorithm based game playing program using some commonly used operators. He found that the GA was insensitive to the game non-linearity, and performed well over a range of environments. It was then with the pioneering work of Holland in 1975 [32] that GAs were firmly established as an effective search and optimization strategy.

GAs mimic some of the processes observed in natural evolution, which include operations like selection, crossover and mutation. They perform multi-modal search in complex landscapes and provide near optimal solutions for objective or fitness function of an optimization problem. They are efficient, adaptive and robust search processes, with a large amount of implicit parallelism [23, 32]. Genetic algorithms are gradually finding widespread applications during the past decade in solving problems requiring efficient and effective search, in business, scientific and engineering circles [20, 21, 61, 64]. Some of the applications of GAs, so far being made, include pattern classification and feature selection [21, 28, 61, 66, 75], image processing and recognition [1, 2, 31, 54, 72, 76], rule generation and classifier systems [11, 34, 35, 36, 37, 39], case based reasoning [58], neural network design [12, 27, 38, 44, 48, 53, 70, 71, 80], scheduling problems [16, 17], control systems [73, 74], VLSI design [79], path planning [15, 63], the traveling salesman problem [25, 40, 52], graph coloring [18], and numerical optimization [41, 46]. Moreover, several researchers are actively engaged in developing enhanced and more effective genetic operators and models, and analyzing their performance for different applications. Some such attempts are described below.

The issue of convergence of GAs to the globally optimal solution has been pursued in [9], where GAs are modeled as Markov chains having a finite number of states. A state is represented by a population together with a potential string. Irrespective of the choice of initial population, GAs have been proved to converge to the optimal string for infinite number of iterations, provided the conventional mutation operation is incorporated. Murthy *et al.* [50] have provided a stopping criterion, called ϵ-optimal stopping time, for the elitist model of the GAs. Subsequently, they have derived the ϵ-optimal stopping time for GAs with elitism under a 'practically valid assumption'.

An attempt to incorporate the ancestors influence into the fitness of individual chromosomes has been made in [19]. This is based on the observations in nature where an individual is not an independent entity, but is highly influenced by the environment. Ghosh *et al.* [22] have incorporated the concept of aging of individuals for measuring their suitability for participation in genetic operations, by combining both the functional value and the age of an individual for computing its effective fitness. Results have shown that this scheme provides enhanced performance and maintains more diversity in the population.

Bhandari *et al.* [10] have proposed a new mutation operator known as *directed mutation* which follows from the concept of induced mutation in biological systems [49]. This operation uses the information acquired in the previous generations rather than probabilistic decision rules. In certain environments, directed mutation will deterministically introduce a new point in the population. The new point is directed (guided) by the solutions obtained earlier, and therefore the technique is called *directed mutation*.

In [53], Pal and Bhandari incorporated GAs to find out the optimal set of weights (biases) in a layered network. Weighted mean square error over the training examples has been used as the fitness measure. They introduced a new concept of selection, called *non-linear selection*, which enhances genetic homogeneity of the population and speeds up searching. Implementation results on both linearly separable and non-linearly separable pattern sets are also reported.

An attempt has also been made for evolving architectures of Hopfield type optimum neural networks for extracting object regions from gray images using GAs [56], where each binary chromosome represents a network architecture. The presence (or absence) of connectivity between neurons is represented by 1 (or 0). The proposed GA based technique has been able to evolve network architectures whose connectivity is about two-third of the requirement of the corresponding fixed fully connected ones in order to produce comparable segmented output. The optimized networks have been found to be more noise independent. Other attempts for evolving the architecture of neural networks using GAs can be found in [8, 67, 80].

Many tasks involved in the process of recognizing a pattern need appropriate parameter selection and efficient search in complex and large spaces in order to attain optimal solutions. This makes the process not only computationally intensive, but also leads to a possibility of losing the exact solution. Therefore, the application of GAs for solving certain problems of pattern recognition, that require optimization of computation requirements, and robust, fast and close approximate solution, seems appropriate and natural. Additionally, the existence of the proof of convergence of GAs to the global optimal solution as the number of iterations goes to infinity [9], further strengthens the theoretical basis of its use in search problems. Significance of GAs to pattern recognition and image processing problems is adequately demonstrated in [21, 57, 60, 61]. Some of the investigations are mentioned below.

A method for determining the optimal enhancement operator for both bimodal and multimodal images is described by Pal *et al.* in [54]. The algorithm does not need iterative visual interaction and prior knowledge of image statistics for this purpose. The fuzziness measures are used as fitness function.

Selection of a subset of principal components for classification using GAs is made in [62]. Since the search space depends on the product of the number of classes and the number of original features, this selection process by conventional means may be computationally very expensive. Results on two data sets with small and large cardinalates are presented.

Murthy and Chowdhury [51] have used GAs for finding optimal clusters, without the need for searching all possible clusters. The experimental results show that the GA based scheme may improve the final output of the K-means algorithm [78], where an improvement is possible. Another attempt for using GAs for clustering has been made in [45], where the cluster centers are encoded in the chromosome, and the distance metric is optimized.

Another hybridization of the k-means algorithm with GAs (called, GKA) for partitional clustering is reported in [42], where the superiority of GKA over other algorithms is demonstrated. The GKA is applied for codebook design that are used in image and speech coding. A class of representation schemes, called Voronoi Networks, has also been proposed in [42] and a new heuristic learning algorithm for them, called supervised K-means algorithm, is formulated.

One of the important and natural applications of GAs for supervised pattern classification is to search and appropriately place a number of surfaces in the feature space such that the decision boundary of a given data set is closely approximated. Attempts in this direction can be found in [4, 77]. In [77], Srikanth *et al.* described a genetic algorithmic approach to pattern classification, both crisp and fuzzy, where clusters in pattern space are approximated by ellipsoids. A variable number of ellipsoids is searched for, which collectively classify a set of objects by minimizing a criteria based on the number of misclassified points and the fuzzy distance of a pattern from the surface of the ellipsoid. In [4], the searching capability of GAs are exploited for approximating the class boundaries using a number of hyperplanes. Extensive experimental results and theoretical analyses of the resulting classifiers are provided.

The present chapter provides, in this direction, some key features of the results of investigation that has been carried out in Machine Intelligence Unit of Indian Statistical Institute, Calcutta, under the project Soft Computing in Pattern Recognition. This includes the *GA-classifier* (where the decision boundary is approximated by a fixed number of hyperplanes), the *VGA-classifier* (where variable string length GAs are used to approximate the decision boundary by a variable number of hyperplanes) and the *VGACD-classifier* (where the concept of chromosome differentiation is incorporated in designing the genetic classifiers), along with some of their real life applications. Some of the results are taken from the existing literature. A distinguishing feature of the above approach is that the boundaries need to be generated explicitly for making decisions. This is unlike the conventional methods or the multilayered perceptron (MLP) based approaches [30, 43, 68, 69], where the generation of boundaries is a consequence of the respective decision making processes. In a part of the article, we have shown how the principle of *VGA-classifier* can be used for describing a methodology for determining the architecture along with the weights of MLP, with each neuron executing the hard limiting function. Comparative performance with other related methods is also provided.

13.2 Overview of genetic algorithms

GAs are modeled on the principles of natural genetic systems, where the genetic information of each individual or potential solution is encoded in structures called *chromosomes*. They use some domain or problem dependent knowledge for directing the search in more promising areas; this is known as the *fitness function*. Each individual or chromosome has an associated fitness function, which indicates its degree of goodness with respect to the solution it represents. Various biologically inspired operators like *selection*, *crossover* and *mutation* are applied on the chromosomes to yield potentially better solutions.

Since a GA works simultaneously on a set of coded solutions it has very little chance of getting stuck at a local optimum when used as an optimization technique. Again, the search space need not be continuous, and no auxiliary information, like derivative of the optimizing function, is required. Moreover, the resolution of the possible search space is increased by operating on coded (possible) solutions and not on the solutions themselves.

To solve an optimization problem, GAs start with the chromosomal representation of a parameter set. The parameter set is to be coded as a finite length string over an alphabet of finite length. Usually, the chromosomes are strings of 0's and 1's. A set of such chromosomes in a generation is called a *population*, the size of which may be constant or may vary from one generation to another. A common practice is to choose the initial population randomly.

The fitness/objective function associated with a chromosome is chosen depending on the problem to be solved, in such a way that the strings (possible solutions) representing good points in the search space have high fitness values. This is the only information (also known as the payoff information) that GAs use while searching for possible solutions.

Subsequently, the selection/reproduction process copies individual strings (called parent chromosomes) into a tentative new population (known as mating pool) for genetic operations. The number of copies that an individual receives for the next generation is usually taken to be directly proportional to its fitness value; thereby mimicking the natural selection procedure to some extent. This scheme is commonly called the *proportional selection scheme*. A commonly used strategy known as the *elitist selection* [26] is adopted in GAs, thereby providing an *elitist GA* (EGA), where the best chromosome of the current generation in retained in the next generation.

The other two frequently used genetic operators applied on the population of chromosomes are crossover and mutation. The main purpose of crossover is to exchange information between randomly selected parent chromosomes by recombining parts of their corresponding strings. It recombines genetic material of two parent chromosomes to produce offspring for the next generation. *Single point crossover* is one of the most commonly used schemes.

The main aim of mutation is to introduce genetic diversity into the population. Sometimes, it helps to regain the information lost in earlier generations. In case of binary representation it negates the bit value and is known as bit mutation. Like natural genetic systems, mutation in GAs is usually performed occasionally. Here a random bit position of a randomly selected string is replaced by another character from the alphabet.

The cycle of selection, crossover and mutation is repeated a number of times till one of the following occurs:

(1) the average fitness value of a population becomes more or less constant over a specified number of generations,
(2) a desired objective function value is attained by at least one string in the population,
(3) the number of generations (or iterations) is greater than some threshold value.

13.3 Description of the genetic classifiers

13.3.1 GA-classifier: GA-based classifier using fixed number of hyperplanes

The *GA-classifier* [55] attempts to place H hyperplanes in the feature space appropriately such that the number of misclassified training points is minimized. From elementary geometry, the equation of a hyperplane in N-dimensional space $(X_1 - X_2 - \cdots - X_N)$ is given by

$$\beta_1 x_N + \beta_2 x_{N-1} + \ldots + \beta_N x_1 = d, \tag{13.1}$$

where $\beta_i = \cos\alpha_{N-i} \sin\alpha_{N-(i-1)} \ldots \sin\alpha_{N-1}$. Here α_j is the angle that the projection of the unit normal in the $(X_1 - X_2 - \cdots - X_{j+1})$ space makes with the X_{j+1} axis. Since $\alpha_0 = 0$, the N tuple $< \alpha_1, \alpha_2, \ldots, \alpha_{N-1}, d >$ specifies a hyperplane uniquely in N dimensional space. An appropriate binary encoding is adopted for these N parameters corresponding to a hyperplane. For details the reader may refer to [55].

Thus, if b_1 and b_2 bits are used to represent an angle and the perpendicular distance variable respectively, then each chromosome is of a fixed length of $l = H((N-1) * b_1 + b2)$, where H denotes the number of hyperplanes. These are initially generated randomly for a population of size Pop.

Using the parameters of the hyperplanes encoded in a chromosome, the region in which each training pattern point lies is determined based on equation (13.1). A region is said to provide the demarcation for class i, if among the points that lie in this region, majority belong to class i. Other points that lie in this region are considered to be misclassified. The misclassifications associated with all the regions (for these H hyperplanes) are summed up to provide the total misclassification, $miss$, for the string. Its fitness is defined as $(n - miss)$, where n is the size of the training data set.

After computing the fitness, the genetic operators of selection, crossover and mutation are applied [23] to generate a new population of chromosomes. Elitism is incorporated in the process for preserving the best candidate found so far. Fitness computation followed by genetic operations are executed for a fixed number of generations, at the end of which the best chromosome provides the set of hyperplanes constituting the final decision boundary. The flowchart for the *GA-classifier* is given in Fig. 13.1.

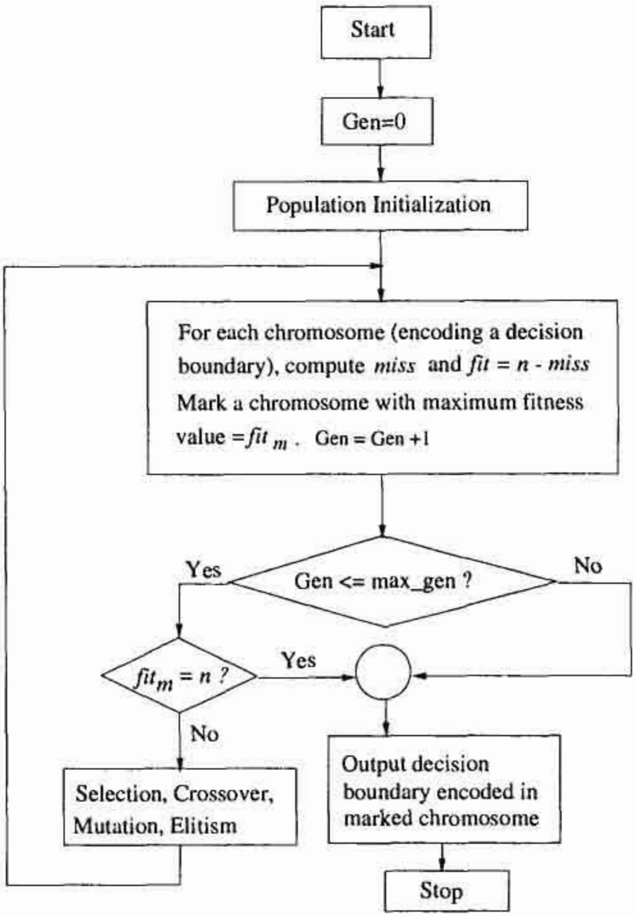

Start
Gen=0
Population Initialization
For each chromosome (encoding a decision boundary), compute miss and fit = n - miss
Mark a chromosome with maximum fitness value = fit m . Gen = Gen +1
Gen <= max_gen ?
Yes
No
fitm = n ?
Yes
No
Selection, Crossover, Mutation, Elitism
Output decision boundary encoded in marked chromosome
Stop

Fig. 13.1 Flowchart for the *GA-classifier*

13.3.2 Determination of optimal H: VGA-classifier

Since it is very difficult to estimate a proper value of H, the *GA-classifier* often suffers from the problem of overfitting of the data set, resulting from a conservative estimate of H. This also leads to the presence of redundant hyperplanes in the final decision boundary. In order to overcome this limitation, the concept of variable string lengths in GAs [24], encoding the parameters of a variable number of hyperplanes, is incorporated in the *GA-classifier*, thereby providing the *VGA-classifier* [5].

In the *VGA-classifier*, the chromosomes are represented by strings of 1, 0 and # (don't care), encoding the parameters of variable number of hyperplanes. Let H_{max} represent the maximum number of hyperplanes that may be required to model the decision boundary of a given data set. It is specified *a priori*.

Fitness Computation

For each string i encoding H_i hyperplanes, the number of misclassified points $miss_i$, is found as in the case for *GA-classifier*. If n is the size of the training data, then the fitness of the ith string, fit_i, is defined as

$$fit_i = (n - miss_i) - \alpha H_i,$$

where $\alpha = \frac{1}{H_{max}}$ and H_i is the number of hyperplanes encoded in the string. A string with zero hyperplane is defined to have zero fitness. Maximization of the fitness function ensures the minimization of, primarily, the number of misclassified points and then the number of hyperplanes.

Genetic Operators

Since the strings have variable length, the crossover and mutation operators are defined afresh as follows.

Crossover: Two strings, i and j, having lengths l_i and l_j respectively are selected from the mating pool. Let $l_i \leq l_j$. Then string i is padded with #s so as to make the two lengths equal. Conventional crossover like single point crossover, two point crossover [23] is now performed over these two strings with probability μ_c. The following two cases may now arise:

1) All the hyperplanes in the offspring are complete. (A hyperplane in a string is called *complete* if all the bits corresponding to it are either defined (*i.e.*, 0s and 1s) or #s. Otherwise it is incomplete.)

2) Some hyperplanes are incomplete.

In the second case let u = number of defined bits (either 0 or 1) and t = total number of bits per hyperplane. Then, for each incomplete hyperplane, all the #s are set to defined bits (either 0 or 1 randomly) with probability $\frac{u}{t}$. In case this is not permitted, all the defined bits are set to #. Thus each hyperplane in the string becomes complete. Subsequently, the string is rearranged so that all the #s are pushed to the end.

Mutation: In order to introduce greater flexibility in the method, the mutation operator is defined in such a way that it can both increase and decrease the string length. For this, the strings are padded with #s such that the resultant length becomes equal to l_{max}. Now for each defined bit position, it is determined whether conventional mutation [23] can be applied or not with probability μ_m. Otherwise, the position is set to # with probability μ_{m_1}. Each undefined position is set to a defined bit (randomly chosen) according to another mutation probability μ_{m_2}.

Note that mutation may result in some incomplete hyperplanes, and these are handled in a manner, as done for crossover operation. Also, mutation may yield strings having all #s indicating that no hyperplanes are encoded in it. Consequently, this string will have fitness $= 0$ and will be automatically eliminated during selection.

13.3.3 Incorporation of chromosome differentiation: VGACD-classifier

In this section, we investigate the effect of incorporating chromosome discrimination [7] on the performance of the said *GA-classifier*. In conventional genetic algorithms, since no restriction is placed upon the selection of mating pair for crossover operation, often chromosomes with similar characteristics are mated. Therefore, no significant new information is gained out of this process and the result is wastage of computational resources. In *VGACD* (variable string length GA with chromosome differentiation) *classifier*, we try to alleviate this problem by distinguishing the chromosomes into two categories, M and F (determined by two additional bits called *class bits*), and therefore two populations. These two populations are initially generated in such a way that they are maximally apart. Crossover is restricted only between individuals from these two populations. Since, as a result of this process, we allow crossover only between dissimilar individuals, a higher level of diversity is likely to be introduced and subsequently maintained in the system. This will, in turn, result in faster information interchange between the chromosomes, and therefore, faster convergence of the algorithm. Interestingly, an analogy of this concept of chromosome differentiation exists in natural genetic systems, in the widely witnessed phenomenon of male and female sexes.

As mentioned above, two additional bits called *class bits* are used to distinguish the chromosomes into 2 classes, M and F. If the *class bits* contain either 01 or 10, the corresponding chromosome is called an M chromosome, and if it contains 00, the corresponding chromosome is called an F chromosome. These bits are not allowed to assume the value 11. (This is in analogy with the X and Y chromosomes in natural genetic systems, where XY or YX indicates male while XX indicates female.) The remaining bits are called *data bits*, which may be either 1, 0 or # (don't care). The *data bits* encode the parameters of H_i hyperplanes, where $1 \leq H_i \leq H_{max}$. The structure of a chromosome in VGACD is shown in Fig. 13.2. Two separate populations, one

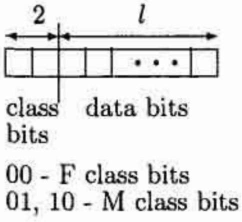


Fig. 13.2 Structure of a chromosome in GACD

containing the M chromosomes (M population) and the other containing the F chromosomes (F population), are maintained over the generations. The sizes of these two populations, p_m and p_f respectively, may vary. Let $p_m + p_f = p$, where p is fixed (equivalent to the population size of conventional GA). Initially we consider $p_m = p_f = \frac{p}{2}$. The M population is first generated in such a way that the first two chromosomes encode the parameters of 1 and H_{max} hyperplanes respectively. The remaining chromosomes encode the parameters of H_i hyperplanes where $1 \leq H_i \leq H_{max}$. For these chromosomes, one of the two *class bits*, chosen randomly, is initialized to 0 and the other to 1. The data bits of the F chromosomes are initially generated in such a way that the hamming distance between the M and F populations (in terms of the data bits) is maximum. The hamming distance between two chromosomes c_1 and c_2, denoted by $h(c_1, c_2)$, is defined as the number of bit positions in which the two chromosomes differ. Hamming distance between two populations, P_1 and P_2, denoted by $h(P_1, P_2)$, is defined as follows :

$$h(P_1, P_2) = \sum_i \sum_j h(c_i, c_j), \quad \forall c_i \in P_1, \forall c_j \in P_2.$$

The computation of fitness in the *VGACD-classifier* is done in a manner similar to that of the *VGA-classifier*. Restricted mating takes place during crossover, where one parent is selected from the M population and the other from the F one. Ignoring the class bits, crossover is performed between the M and F parents as in the *VGA-classifier*. Subsequently, each parent contributes one class bit to the offspring. Since the F parent can only contribute a 0 (its class bits being 00), the class of the child is primarily determined by the M parent which can contribute a 1 (yielding an M child) or a 0 (yielding an F child) depending upon the bit position (among the two class bits) of the M parent chosen. This process is performed for both the offspring whereby either two M or two F or one M and one F offspring will be generated. These are put in the respective populations. Mutation is performed as in *VGA-classifier*, with the class bits being kept outside the purview of this operator.

13.3.4 Theoretical studies of the genetic classifiers

Theoretical analyses of the above mentioned GA based classifiers show that for infinitely large number of iterations it will provide the minimum misclassification error during training; at the same time the number of hyperplanes required to model the decision boundary appropriately for providing the minimum number of misclassified points will also be the minimum.

It is known from the literature that Bayes classifier [78] is the best possible classifier if the class conditional densities and the *a priori* probabilities are known. No classifier can provide better performance than Bayes classifier under such conditions. In practice, it is difficult to use Bayes classifier because the class conditional densities and the *a priori* probabilities may not be known. Hence new classifiers are devised and their performances are compared to that of the Bayes classifier. The desirable property of any classifier is that its performance should approximate or approach that of the Bayes classifier under limiting conditions. There are many ways in which the performance of a classifier is compared to that of the Bayes classifier. One such way is to investigate the behavior of the error rate (defined as the ratio of the number of misclassified points to the size of the data set) as the size of the training data goes to infinity, and check whether the limiting error rate is equal to the Bayes error probability. Such an investigation [6] establishes that this is true for the aforesaid genetic classifiers when the number of iterations goes to infinity. In other words, the decision boundary provided by the genetic classifiers approaches the Bayes decision boundary as the number of training data points and the number of iterations approaches infinity.

13.3.5 Experimental results

This section has two parts. In the first part, some experimental results are presented for the genetic classifiers on a *Vowel* data set. This includes a description of the data set, variation of the recognition scores of the *GA-classifier* during testing for different values of H, performance of the *VGA-classifier* and *VGACD-classifier*, and their comparison to the Bayes maximum likelihood classifier and k-NN rule. In the second part, the above genetic classifiers are used for pixel classification of a *SPOT* satellite image of a part of the city of Calcutta for locating different landcover regions.

For the GA based classifiers the numbers of bits used to represent an angle and the perpendicular distance are 8 and 16 respectively. *Roulette wheel* selection is adopted to implement the *proportional selection strategy*. *Single point* crossover is applied with a fixed crossover probability (μ_c) value of 0.8. The mutation operation is performed on a bit by bit basis for a varying mutation probability value (μ_m) in the range [0.015, 0.333]. The form of the variation of μ_m with the number of generations is shown in Fig. 13.3. The range is divided

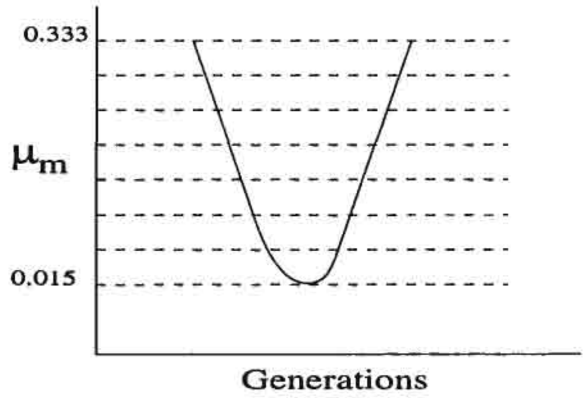


Fig. 13.3 Variation of mutation probability with the number of generations

into eight equispaced values. μ_m is slowly decreased in steps from 0.333 to 0.015, and then increased again. This ensures that initially, a random search is performed through the feature space. The randomness is gradually decreased with the passing of generations so that now the algorithm performs a detailed search in the vicinity of promising solutions obtained so far. In spite of this, the algorithm may still get stuck at a local optimum. This problem is overcome by increasing the mutation probability to a high value, thereby making the search more random once again. 100 and 200 iterations are executed with each value of μ_m for the *GA-classifier* and the variable string length GA-based classifiers respectively. (Note that since the search space is larger for the latter, they are allowed more time to execute.) The fixed length genetic classifier is terminated if the population contains at least one string with no misclassified points, while the variable length ones are terminated if the number of hyperplanes is reduced to one in addition to zero misclassified points for at least one string. Otherwise, the algorithms are executed for the prespecified number of generations.

13.3.5.1 *Results on* Vowel *data*

Vowel data consists of 871 Indian Telugu vowel sounds [59]. These were uttered in a consonant-vowel-consonant context by three male speakers in the age group of 30-35 years. The data set has three features F_1, F_2 and F_3, corresponding to the first, second and third vowel formant frequencies, and six classes $\{\delta, a, i, u, e, o\}$. Fig. 13.4 shows the distribution of the six classes in the $F_1 - F_2$ plane. (It is known [59] that these two features are more important in characterizing the classes than F_3.) Note that the boundaries of the classes are seen to be very ill-defined and overlapping. The scores provided here are the average values obtained over five different runs of the algorithms. Table 13.1 presents the test recognition scores of the *GA-classifier* for different values of H. The scores are found to improve with the value of H upto $H = 7$ which provides the maximum score. Low values of H viz., 2 and 3 are seen to provide quite low recognition scores indicating that they are insufficient for modeling the overlapping class boundaries appropriately. Even $H = 4$ is not a proper choice since in this case one class is not recognized at all (class a). Interestingly, although it was found that the recognition scores during training of the *GA-classifier* consistently improved with the increase of H from 2 to 8, a decrease in the test recognition score is observed from $H = 7$ to $H = 8$. The reason for this is that as H is increased upto a certain point (from 2 to 7), the *GA-classifier* is able to surround the data points more easily during training thereby providing improved scores. However increasing H beyond a certain point (in this case 7) results in overfitting of the training data points at the cost of reduced generalization capability. Therefore, although the training scores improve even further, the recognition scores during testing get degraded.

Table 13.2 shows the number of hyperplanes H_{VGA} and H_{VGACD} as determined automatically by the *VGA-classifier* and *VGACD-classifier* respectively,

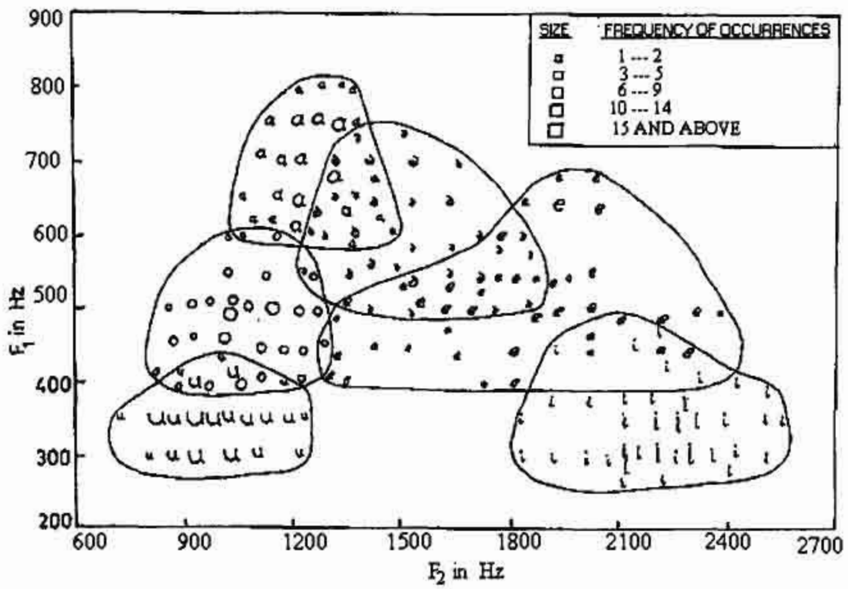


Fig. 13.4 Vowel data in the $F_1 - F_2$ plane

for modeling the class boundaries of *Vowel*. Two different values of H_{max}, namely, 6 and 10, are used for this purpose. The overall recognition scores obtained during testing of the variable string length GA classifiers along with their comparison with those of the corresponding fixed length version (*i.e.*, *GA-classifier* with $H = 6$ and 10) are also shown. The purpose of this exercise is to compare the performance of the asexual and sexual versions of the genetic classifiers, as well as the fixed and variable length ones, starting with the same number of hyperplanes, *i.e.*, H_{max} for *VGA-classifier* and *VGACD-classifier*, and H for *GA-classifier*.

It is found from Table 13.2 that the *VGACD-classifier* consistently outperforms the other two classifiers, indicating the significant advantage of incorporating the concept of chromosome differentiation. Moreover, it is also able to considerably reduce the number of hyperplanes. The *VGA-classifier* is unable to eliminate any hyperplane when it is initiated with $H_{max} = 6$. Interestingly, although its recognition score on the test data set is found to be higher than that of the *GA-classifier* for $H_{max} = 10$ (where finally six hyperplanes are utilized), this is not the case for $H_{max} = 6$. This may be due to the fact

Table 13.1 Variation of overall recognition scores (%) during testing with H for *Vowel* data with $perc = 10$

Number of hyperplanes	Overall recognition score
8	74.56
7	74.68
6	71.99
5	71.37
4	69.21
3	57.50
2	53.30

Table 13.2 H_{VGA} and the comparative overall recognition scores (%) during testing (when 10% of the data set is used for training and the remaining 90% for testing)

H_{max}	*VGA-classifier*		*VGACD-classifier*		Score for *GA-classifier*
	H_{VGA}	Score(%)	H_{VGACD}	Score (%)	when $H = H_{max}(\%)$
6	6	71.19	3	77.09	71.99
10	6	73.66	4	78.49	69.21

that with ten hyperplanes the *VGA-classifier* has more flexibility of placing a smaller number of hyperplanes appropriately than in the case when $H_{max} = 6$. Therefore on termination of the algorithm after a fixed number of generations, the former is able to better approximate the decision boundary than the latter. However it may be noted that the recognition scores of the *VGA-classifier* would have improved further if more iteration of the classifier are executed. The superior results for the *VGACD-classifier* demonstrates that incorporation of the concept of chromosome differentiation increases the rate of convergence of the algorithm, thereby providing betters results earlier.

Regarding the time taken for training of the classifiers, the variable length genetic classifiers are found to take longer than the *GA-classifier*. As is mentioned earlier, this is because the search space is larger for the former (where the number of hyperplanes varies in the range $[1, H_{max}]$) than for *GA-classifier* (where the number of hyperplanes is fixed).

For the purpose of comparing the performance of the genetic classifiers, we have used Bayes maximum likelihood classifier (which is well known for discriminating overlapping classes), and k-NN classifier and the multilayer perceptron (both of which are well known for discriminating non-overlapping, non-linear regions by generating piecewise linear boundaries). Recognition scores on the 90% test data when the remaining 10% of the data set was used for training were 77.73%, 70.35% and 68.48% for Bayes maximum likelihood classifier (with assumption of normal distribution, and estimation of the covariance matrices and *a priori* probabilities from the training data set), k-NN rule (for k=$\sqrt{n}$) and MLP (with two hidden layers and twenty hidden nodes per layer) respectively. As can be found from comparison with the results in Table 13.2, the performance of Bayes maximum likelihood classifier and the *VGACD-classifier* are comparable. Note that earlier findings [59] indicated that Bayes maximum likelihood classifier performs very well for this data. This is followed by the score of the *VGA-classifier* for $H_{max} = 10$. MLP is found to perform the poorest for this data. Detailed results considering other parameters are provided in [5, 55].

13.3.5.2 *Pixel classification of* SPOT *image*

In this section, the utility of the genetic classifiers for classification of pixels for partitioning different landcover regions in satellite images is investigated. Satellite images usually have a large number of classes with overlapping and non-linear class boundaries. Fig. 13.5 shows, as a typical example, the complexity in scatter plot of 932 points belonging to seven classes which are taken from the *SPOT* image of a part of the city of Calcutta. Therefore, for appropriate modeling of such non-linear and overlapping class boundaries, the utility of an efficient search technique like GAs is evident. Moreover, it is desirable that the search technique does not need to assume any particular distribution of the data set and/or class *a priori* probabilities.

The SPOT image considered in this experiment has three bands. These are:
Band 1 - green band of wavelength 0.50 - 0.59 μm,
Band 2 - red band of wavelength 0.61 - 0.68 μm, and
Band 3 - near infra red band of wavelength 0.79 - 0.89 μm.
The training set comprises 932 points belonging to seven classes, with three

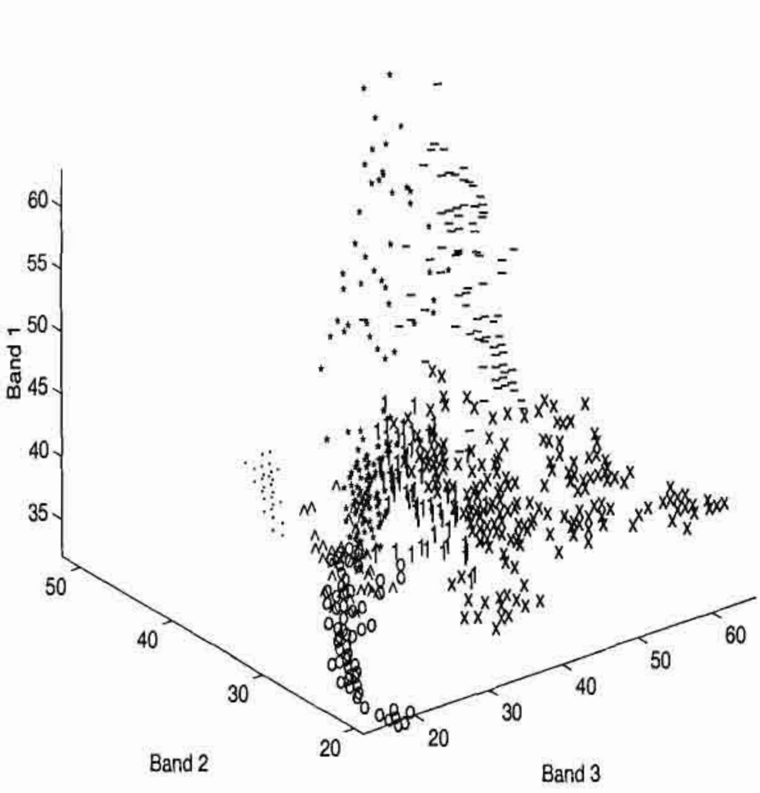


Fig. 13.5 Scatter plot for the training points in Calcutta image

features corresponding to the above mentioned bands. The seven classes are *turbid water* (TW), *pond water* (PW), *concrete* (Concr.), *vegetation* (Veg), *habitation* (Hab), *open space* (OS) and *roads* (including bridges) (B/R). Some important landcovers of Calcutta can be identified, from a knowledge about the area, more easily in Band 3 of the image (Fig. 13.6 shows the image with 75% stretching to make it more prominent). These are the following: The prominent black stretch across the figure is the river *Hooghly*. Portions of a bridge (referred to as the *second bridge*), which was under construction when the picture was taken, protrude into the *Hooghly* near its bend around the center of the image. There are two distinct black, elongated patches below the river, on the left side of the image. These are water bodies, the one to the left being *Garden Reach lake* and the one to the right being *Khidirpore dockyard*.

Just to the right of these water bodies, there is a very thin line, starting from the right bank of the river, and going to the bottom edge of the picture. This is a canal called the *Talis nala*. Above the *Talis nala*, on the right side of the picture, there is a triangular patch, the *race course*. On the top, right hand side of the image, there is a thin line, stretching from the top edge, and ending on the middle, left edge. This is the *Beleghata canal* with a road by its side. There are several roads on the right side of the image, near the middle and top portions. These are not very obvious from the images. A bridge cuts the river near the top of the image. This is called the *Rabindra Setu*.

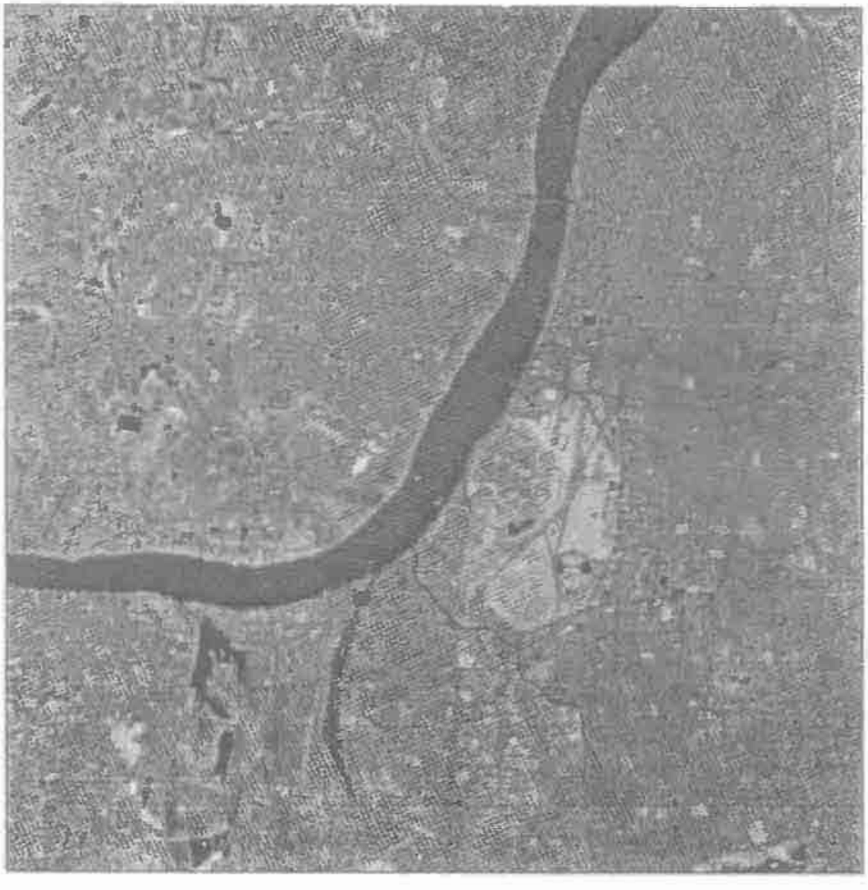


Fig. 13.6 Band 3 of the Calcutta image with 75% stretching

Issue of Large Value of H: In view of the complexity of the data sets, high values of H like 15 and 20 for the GA based classifiers were considered. Since the maximum number of regions provided by H hyperplanes is equal to 2^H, the aforesaid high values of H make the number of regions ($= 2^H$) also very large. This leads to a practical limitation of the method. However, an important point that needs to be taken into consideration is that the possible number of regions can never be larger than the number of points n in the training data set. Also, $n << 2^H$ for large H. Thus we need to consider at most n regions while tackling this problem. In fact, the number of regions for this problem was found to be considerably less than n as well.
Results: Figs. 13.7(a)-(f) provide, as an illustration, the results obtained by the *VGACD-classifier*, *VGA-classifier*, Bayes maximum likelihood classifier, and k-NN rule (for k=1, 3, and $\sqrt{n}$), for partitioning the 512×512 image, by zooming a characteristic portion of the image containing the *race course* (a triangular structure). Here 80% of the design set is used for training.

As seen from the figures, although all the classifiers (with the exception of k=1 for k-NN rule) are able to identify the *race course*, only the *VGACD-classifier* and the *VGA-classifier* are able to identify a triangular lighter outline (which is an open space, corresponding to the tracks) within the *race course* properly. The performance of k-NN rule is found to gradually improve with the value of k, being the best for k=$\sqrt{n}$. On inspection of the full classified images it was found that the Bayes maximum likelihood classifier tends to over estimate the roads in the image. On the other hand, the *VGA-classifier* tends to confuse between the classes bridges and roads (B/R) and pond water (PW). It was revealed on investigation that a large amount of overlap exists between the classes Concr and B/R on one hand (in fact, the latter class has been extracted from the former), and PW and B/R on the other hand. These problems were not evident for the case of the *VGACD-classifier*.

13.4 Determination of MLP architecture

This section describes a scheme for the determination of the architecture as well as the connection weights of an MLP based on the results of the *VGA-classifier*. An MLP with hard limiting neurons is described first followed by establishing the analogy between its classification principles and those of the genetic classifier. Subsequently, the architecture determination method is presented followed by presentation of the experimental results.

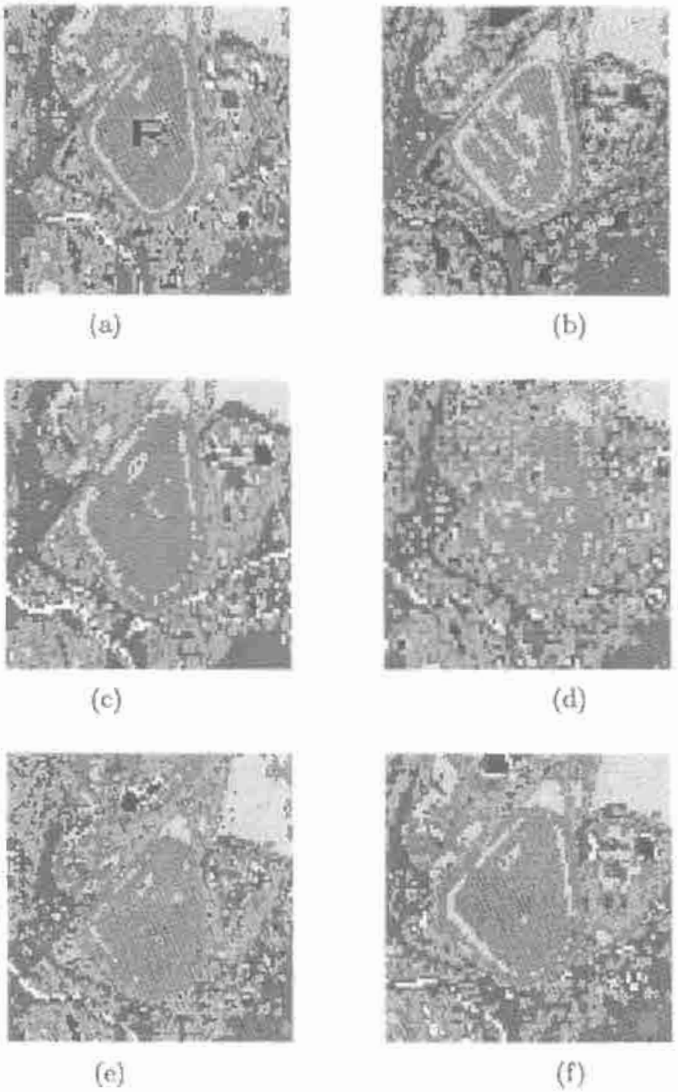


Fig. 13.7 Classified *SPOT* image of Calcutta (zooming the *race course*, represented by 'R' on the first figure, only) using (a) *VGACD-classifier*, H_{max} =15, final value of H=13, (b) *VGA-classifier*, H_{max} =15, final value of H=10, (c) Bayes maximum likelihood classifier. (d) k-NN rule, k = 1, (e) k-NN rule, k = 3, (f) k-NN rule, k = $\sqrt{n}$. Training set = 80% of design set

In this context it must be mentioned that there are several approaches for determining the MLP architecture and connection weights [13, 44, 53]. In [53], the connection weights for a given MLP architecture are determined using GAs, where the weights are encoded in the chromosomes. The weighted error is taken as the fitness of a string. This method, therefore, totally eliminates the necessity of using back propagation (BP) algorithm for training. In [44], parallel genetic algorithm is used for evolving the topology and weights of feedforward artificial neural networks. Here both the connectivity and the weights are encoded in the chromosomes. Additionally, the granularity *i.e.*, the number of bits used for encoding the weights is also encoded as a parameter of the problem. This method, thus, utilizes variable string lengths for topology and weight determination. Another method based on the construction of Voronoi diagrams over the set of training patterns is described in [13], where the number of layers, number of neurons in each layer and the connection weights are automatically determined. Pruning a network of a large size is another approach towards determination of proper network architecture. A detailed survey can be found in [65].

13.4.1 Multilayer perceptrons

A Multilayer Perceptron (MLP) consists of several layers of simple neurons with full connectivity existing between neurons of adjacent layers. The neurons in the input layer serve the purpose of fanning out the input values to the neurons of *layer 1*. Let $w_{ji}^{(l)}$ represent the connections weight on the link from the ith neuron in layer $l-1$ to the jth neuron in layer l. Let $\theta_j^{(l)}$ represent the threshold of the jth neuron in layer l. The total input, $x_j^{(l)}$, received by the jth neuron in layer l is given by

$$x_j^{(l)} = \sum_i y_i^{(l-1)} \; w_{ji}^{(l)} + \theta_j^{(l)}, \tag{13.2}$$

where $y_i^{(l-1)}$ is the output of the ith neuron in layer $l-1$. For the input layer

$$y_i^{(0)} = x_i, \tag{13.3}$$

where x_i is the ith component of the input vector. For the other layers

$$y_i^{(l)} = f(x_i^{(l)}). \tag{13.4}$$

Several functional forms like threshold logic, hard limiter and sigmoid, can be used for $f(\cdot)$.

There are several algorithms for training the network in order to learn the connection weights and the thresholds from a given training data set. Back-propagation (BP) is one such learning algorithm, where the least mean square error of the network output is computed, and this is propagated in a top down manner (*i.e.*, from the output side) in order to update the weights. The error is computed as the difference between the actual and the desired output when a known input pattern is presented to the network. A gradient descent method along the error surface is used in BP.

13.4.2 Analogy between multilayer perceptron and VGA-classifier

It is known in the literature [43] that Multilayered Perceptron (MLP) with hard limiting nonlinearities approximates the decision boundaries by piecewise linear surfaces. The parameters of these surfaces are encoded in the connection weights and threshold biases of the network. Similarly, the *VGA-classifier* also generates decision boundaries by appropriately fitting a number of hyperplanes in the feature space. The parameters are encoded in the chromosomes. Thus a clear analogy exists between these two models.

Both the methods start from an initial randomly generated state (the set of initial random weights in MLP). Both of them iterate over a number of generations while attempting to decrease the classification error in the process.

The obvious advantage of the GA based method over that of the MLP is that the former performs concurrent search for a number of sets of hyperplanes, each representing a different classification in the feature space. On the other hand, the MLP deals with only one such set. Thus it has a greater chance of getting stuck at a local optimum, which the genetic classifier can overcome. Moreover, the *VGA-classifier* does not assume any fixed value of the number of hyperplanes, while MLP assumes a fixed number of hidden nodes and layers. This results in the problem of over fitting with an associated loss of generalization capability for MLP. In this context one must note that since the *VGA-classifier* has to be terminated after finitely many iterations, and the size of the data set is also finite, it may not always end up with the optimal number of hyperplanes. Consequently, the problem of overfitting exists for *VGA-classifier* also, although it is comparatively reduced.

13.4.3 Deriving the MLP architecture

In this section we describe how the principle of fitting a number of hyperplanes using GA, for approximating the class boundaries, can be exploited in determining the appropriate architecture of MLP. Since our aim is to model the equation of hyperplanes, we use the hard limiting function in the neurons of the MLP, defined as

$$f(x) = \begin{cases} +1 \text{ if } x \geq 0 \\ -1 \text{ if } x < 0. \end{cases}$$

13.4.3.1 *Terminology*

Let us assume that the *VGA-classifier* provides H_{VGA} hyperplanes, designated by

$$\{Hyp_1, Hyp_2, \ldots, Hyp_{H_{VGA}}\},$$

r regions, designated by

$$\{R_1, R_2, \ldots, R_r\},$$

and k classes, designated by

$$\{C_1, C_2, \ldots, C_k\}.$$

Note that more than one region may be labeled with a particular class, indicating that $r \geq k$.

Let R^1 be the region representing class C_1, and let it be a union of r_1 regions given by

$$R^1 = R_{j_1^1} \cup R_{j_2^1} \cup \ldots R_{j_{r_1}^1}, \qquad 1 \leq j_1^1, j_2^1, \ldots, j_{r_1}^1 \leq r.$$

Generalizing the above, let R^i $(i = 1, 2, \ldots, k)$ be the region representing class C_i, and let it be a union of r_i regions given by

$$R^i = R_{j_1^i} \cup R_{j_2^i} \cup \ldots \cup R_{j_{r_i}^i}, \qquad 1 \leq j_1^i, j_2^i, \ldots, j_{r_i}^i \leq r.$$

Note that each R^i is disjoint, *i.e.*,

$$R^i \cap R^j = \phi \quad i \neq j, \;\; i, j = 1, 2, \ldots, k.$$

13.4.3.2 *Network construction algorithm*

The network construction algorithm (NCA) is a four step process where the number of neurons, their connection weights and the threshold values are determined. It guarantees that the total number of hidden layers (excluding the input and output layers) will be at most two. (In this context, Kolmogorov's Mapping Neural Network Existence Theorem must be mentioned. The theorem states that any continuous function can be implemented exactly by a three layer, including input and output layers, feedforward neural network. The proof can be found in [29]. However, nothing has been stated about the selection of connection weights and the neuronal functions.)

The output of the *VGA-classifier* is the parameters of the H_{VGA} hyperplanes. These are obtained as follows :

$$\begin{array}{ll}
\alpha_1^1, \; \alpha_2^1, \ldots, & \alpha_{N-1}^1, \;\; d^1 \\
\alpha_1^2, \; \alpha_2^2, \ldots, & \alpha_{N-1}^2, \;\; d^2 \\
\vdots & \\
\alpha_1^{H_{VGA}}, \; \alpha_2^{H_{VGA}}, \ldots, \alpha_{N-1}^{H_{VGA}}, \; d^{H_{VGA}} &
\end{array}$$

Step 1: Allocate N neurons in the input layer, *layer 0*, where N is the dimensionality of the input vector. The neurons in this layer simply transmit the value in the input links to all the output links.

Step 2: Allocate H_{VGA} neurons in *layer 1*. Each neuron is connected to the N neurons of *layer 0*. Let the equation of the ith hyperplane ($i = 1, 2, \ldots, H_{VGA}$) be

$$c_1^i x_1 + c_2^i x_2 + \ldots + c_N^i x_N - d = 0,$$

where from (13.1) we may write

$$\begin{aligned} c_N^i &= \cos\alpha_{N-1}^i \\ c_{N-1}^i &= \cos\alpha_{N-2}^i \sin\alpha_{N-1}^i \\ c_{N-2}^i &= \cos\alpha_{N-3}^i \sin\alpha_{N-2}^i \sin\alpha_{N-1}^i \\ &\vdots \\ c_1^i &= \cos\alpha_0^i \sin\alpha_1^i \ldots \sin\alpha_{N-1}^i \\ &= \sin\alpha_1^i \ldots \sin\alpha_{N-1}^i \end{aligned}$$

since $\alpha_0^i = 0$.

Then the corresponding weights on the links to the ith neuron in *layer 1* from those in *layer 0* are

$$w_{ij}^1 = c_j^i \qquad j = 1, 2, \ldots, N$$

and

$$\theta_i^1 = -d^i,$$

since the bias term is added to the weighted sum of the inputs to the neurons.
Step 3: Allocate r neurons in *layer 2* corresponding to the r regions. If the ith region R_i $(i = 1, 2, \ldots, r)$ lies on the positive side of the jth hyperplane Hyp_j $(j = 1, 2, \ldots, H_{VGA})$, then

$$w_{ij}^2 = +1.$$

Otherwise

$$w_{ij}^2 = -1,$$

and

$$\theta_i^2 = -(H_{VGA} - 0.5).$$

Note that the neurons in this layer effectively serve the AND function, such that the output is high $(+1)$ if and only if all the inputs are high $(+1)$. Otherwise, the output is low (-1).

Step 4: Allocate k neurons in *layer 3* (output layer), corresponding to the k classes. The task of these neurons is to combine all the distinct regions that actually correspond to a single class. Let the ith class ($i = 1, 2, \ldots, k$) be a combination of r_i regions. That is,

$$R^i = R_{j_1^i} \cup R_{j_2^i} \cup \ldots \cup R_{j_{r_i}^i}.$$

Then the ith neuron of *layer 3*, ($i = 1, 2, \ldots, k$), is connected to neurons $j_1^i, j_2^i \ldots j_{r_i}^i$ of *layer 2* and,

$$w_{ij}^3 = 1 \qquad j \in \{j_1^i, j_2^i \ldots j_{r_i}^i\}$$

whereas

$$w_{ij}^3 = 0 \qquad j \notin \{j_1^i, j_2^i \ldots j_{r_i}^i\}$$

and

$$\theta_i^3 = r_i - 0.5.$$

Note that the neurons in this layer effectively serve the OR function, such that the output is high (+1) if at least one of the inputs is high (+1). Otherwise, the output is low (-1). For any given point, at most one output neuron, corresponding to its class, will be high. Also, none of the output neurons will be high if an unknown pattern, lying in a region with unknown classification (*i.e.*, there were no training points in the region), becomes an input to the network.

13.4.3.3 *Post-processing step*

The network obtained from the application of NCA may be further optimized in terms of the links and neurons in the output layer. A neuron in *layer 3* that has an input connection from only one neuron in *layer 2* may be eliminated completely. Mathematically, let for some i, $1 \le i \le k$,

$$\begin{aligned} w_{ij}^3 &= 1 \ \text{ if } j = j' \\ &= 0 \ \text{ otherwise,} \end{aligned}$$

then neuron i of *layer 3* is eliminated and is replaced by neuron j' of *layer 2*. Its output then becomes the output of the network. Note that this step produces a network where a neuron in layer i is connected to a neuron in layer $i+2$.

In the extreme case, when all the neurons in the output layer (*layer 3*) get their inputs from exactly one neuron in *layer 2*, the output layer can be totally eliminated, and *layer 2* becomes the output layer. This reduces the number of layers from three to two. This will be the case when $r = k$, *i.e.*, a class is associated with exactly one region formed by the H_{VGA} hyperplanes.

13.4.4 Implementation

The effectiveness of the network construction algorithm (NCA) has been demonstrated on a number of real life and artificial data sets [8]. Here we provide the results corresponding to the *Vowel* data only. The parameters for the *VGA-classifier* are kept the same as mentioned earlier. The recognition scores provided here are the average values obtained over five different runs of the algorithm. H_{max} is set to 10, so $\alpha = 0.1$.

The MLP is executed using both hard limiters and the sigmoid function in the neurons. The sigmoid function is defined as

$$f(x) = \frac{1}{1+e^{-x}}.$$

The learning rate and momentum factor are fixed at 0.8 and 0.2 respectively. Online weight updating, *i.e.*, updating after each training data input, is performed for a maximum of 3000 iterations.

The performance of *VGA-classifier* and consequently that of the MLP derived using NCA (*i.e.*, where the architecture and the connection weights have been determined using NCA) is compared with that of a conventional MLP having the same architecture as provided by NCA, but trained using the back propagation (BP) algorithm with the neurons executing the sigmoid function. For the purpose of comparison, we have also considered here three more typical architectures for the conventional MLP having two hidden layers with 5, 10 and 20 nodes in each layer respectively. Table 13.3 summarizes the results obtained. The MLP architecture is denoted by Arch. in the tables.

The number of hyperplanes (H_{VGA}) and regions (r) obtained by the *VGA-classifier* starting from $H_{max} = 10$ are 6 and 7 respectively. These are used to select the MLP architectures as shown in columns 8-9 and 10-11. Note that the recognition scores mentioned in columns 10 and 11 for MLP (derived using

NCA) are the same as those obtained for the corresponding *VGA-classifier*. From the table it is found that the MLP derived using NCA provides a superior performance than the MLPs trained with BP. The overall recognition score during testing for *Vowel* is found to increase with the increase in the number of nodes in the hidden layers (columns 5, 7 and 9) since the classes are highly overlapping.

Note that the Arch. values of the MLPs mentioned in columns 8-9 and 10-11 of the tables are the ones obtained without the application of the post-processing step. These values are put in order to clearly represent the mapping from *VGA-classifier* to MLP, in terms of the number of hyperplanes and regions, although the post-processing task could have reduced the size of the network while keeping the performance unchanged. In this case the architecture became 3:6:2:6 after post-processing.

13.5 Discussion and conclusions

This article dealt with the development of several pattern classifiers, along with their theoretical and practical aspects, using both conventional genetic algorithms and some of their modifications/enhancements. The search and optimization capability of genetic algorithms has been exploited for the placement of an appropriate number of surfaces in the feature space, such that the associated number of misclassified points is minimized. The classifiers store the parameters of the surfaces constituting the final decision boundary, and the region-class associations. These are later used for determining the region and hence the class of an unknown pattern. Various versions of the genetic classifier *e.g.*, *GA-classifier* using fixed number of surfaces, *VGA-classifier* using variable number of surfaces, *GACD-classifier* incorporating the concept of chromosome differentiation, have been formulated. Some theoretical results have been provided.

The effectiveness of these genetic classifiers and their comparison with Bayes maximum likelihood classifier (which is well known for discriminating overlapping classes), k-NN rule and MLP (both of which are well known for discriminating non-overlapping, non-convex regions by generating piecewise linear boundaries) are demonstrated on a speech data *Vowel*. The way of incorporating the concept of variable string length in *VGA-classifier* is also compared with that of Srikanth *et al.* [77] in a part of the experiment. Besides these, the problem of pixel classification from satellite images for partitioning various

Table 13.3 Classwise and overall recognition scores (%) for *Vowel*

Class	MLP									
	SIGMOID								Hard Limiter	
	Arch.=3:5:5:6		Arch.=3:10:10:6		Arch.=3:20:20:6		Arch.=3:6:7:6		Arch.=3:6:7:6	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
δ	85.71	12.30	87.51	47.69	100.00	47.23	85.71	6.15	10.30	8.21
a	75.00	43.20	100.00	16.04	100.00	27.62	87.50	13.58	100.00	91.35
i	100.00	69.67	100.00	79.35	100.00	80.58	100.00	78.70	94.11	84.51
u	86.67	80.14	100.00	77.94	100.00	83.29	100.00	91.91	73.33	66.91
e	80.00	68.44	94.98	90.37	100.00	87.30	80.00	59.35	89.99	85.56
o	88.89	77.16	94.44	54.93	94.44	51.70	77.78	43.21	83.33	75.92
Overall	87.05	65.26	95.82	67.55	98.82	68.48	88.23	56.36	80.00	73.66

landcover types with ill-defined boundaries is considered as another real life application.

As an interesting real-life application of the *VGA-classifier*, as attempt to determine the architecture of MLP (with hard limiting neurons) automatically has been described. The superiority of the MLP derived by this method *vis-a-vis* a few randomly selected ones is demonstrated experimentally. An important area of further research in this direction would be to utilize the genetically derived MLP as the initial architecture, and then attempt to refine it further using methods similar to cascade correlation algorithm. Moreover, the said analogy will augment the application domain of the *VGA-classifier* to those areas where MLP has widespread use.

In this article, binary representation of chromosomes in linear form has been used, primarily because it is well studied in the literature, and it maximizes the number of schemata sampled by each member of the population; thereby making the implicit parallelism of GAs to be used to the fullest. However, in many practical situations, this may not be a natural choice. Thus, other kinds of representation, like floating point, tree, matrix representation [33, 47] may be studied.

A modification of the fitness function by incorporating the information on relative position of the boundary from the training data may constitute another part of further investigation. The classification methodology presented here considers only the total number of misclassified points as the optimizing criterion. It does not take the classwise recognition scores into account. This may sometimes lead to an undesirable situation where the overall recognition score is high, but some classes are totally ignored. In order to tackle this, an investigation may be undertaken where the classwise weighted scores constitute the fitness criterion of the chromosomes. The weighting factor may take some *a priori* class information, like the class probabilities, into account.

In GACD, the chromosomes have been differentiated into two classes and a form of restricted mating has been applied. As a part of future work, the effect of differentiating the chromosomes into more than two categories may be investigated. An investigation also needs to be performed for the selection of appropriate kinds of surfaces (*i.e.*, linear, higher order) for modeling the class boundaries.

References

[1] A. S. Abutaleb and M. Kamel, "A genetic algorithm for the estimation of ridges in fingerprints," *IEEE Trans. on Image Processing*, vol. 8, pp. 1134–1139, 1999.

[2] C. A. Ankenbrandt, B. P. Buckles, and F. E. Petry, "Scene recognition using genetic algorithms with semantic nets," *Pattern Recog. Lett.*, vol. 11, pp. 285–293, 1990.

[3] J. D. Bagley, *The Behaviour of Adaptive Systems which Employ Genetic and Correlation Algorithms.* PhD thesis, University of Michigan, Ann Arbor, 1967.

[4] S. Bandyopadhyay, *Pattern Classification Using Genetic Algorithms.* PhD thesis, Machine Intelligence Unit, Indian Statistical Institute, Calcutta, India, 1998.

[5] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Pattern classification using genetic algorithms: Determination of H," *Pattern Recog. Lett.*, vol. 19, no. 13, pp. 1171–1181, 1998.

[6] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Theoretical performance of genetic pattern classifier," *J. Franklin Institute*, vol. 336, pp. 387–422, 1999.

[7] S. Bandyopadhyay, S. K. Pal, and U. Maulik, "Incorporating chromosome differentiation in genetic algorithms," *Inform. Sci.*, vol. 104, no. 3/4, pp. 293–319, 1998.

[8] S. Bandyopadhyay and S. K. Pal, "Relation between VGA-classifier and MLP: Determination of network architecture," *Fundamenta Informaticae*, vol. 37, pp. 177–196, 1999.

[9] D. Bhandari, C. A. Murthy, and S. K. Pal, "Genetic algorithm with elitist model and its convergence," *Int. J. Pattern Recog. Art. Intell.*, vol. 10, pp. 731–747, 1996.

[10] D. Bhandari, N. R. Pal, and S. K. Pal, "Directed mutation in genetic algorithms," *Inform. Sci.*, vol. 79, pp. 251–270, 1994.

[11] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Art. Intell.*, vol. 40, pp. 235–282, 1989.

[12] S. Bornholdt and D. Graudenz, "General asymmetric neural networks and structure design by genetic algorithms," *Neural Networks*, vol. 5, pp. 327–334, 1992.

[13] N. K. Bose and A. K. Garga, "Neural network design using voronoi diagrams," *IEEE Trans. Neural Networks*, vol. 4, pp. 778–787, 1993.

[14] B. P. Buckles and F. E. Petry, eds., *Genetic Algorithms*. Los Alamitos: IEEE Computer Society Press, 1994.

[15] T. Cleghorn, P. Baffes, and L. Wang, "Robot path planning using a genetic algorithm," in *Proc. SOAR*, (Houston), pp. 81–87, 1988.

[16] F. A. Cleveland and S. F. Smith, "Using genetic algorithms to schedule flow shop releases," in *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. Schaffer, ed.), pp. 160–169, San Mateo: Morgan Kaufmann, 1989.

[17] L. Davis, "Job shop scheduling with genetic algorithms," in *Proc. 1st Int. Conf. Genetic Algorithms* (J. J. Grefenstette, ed.), pp. 136–140, Hillsdale: Lawrence Erlbaum Associates, 1985.

[18] L. Davis, ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[19] S. De, A. Ghosh, and S. K. Pal, "Fitness evaluation in genetic algorithms with ancestors' influence," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 1–23, Boca Raton: CRC Press, 1996.

[20] L. J. Eshelman, ed., *Proc. 6th Int. Conf. Genetic Algorithms*. San Mateo: Morgan Kaufmann, 1995.

[21] E. S. Gelsema, ed., *Special Issue on Genetic Algorithms, Pattern Recognition Letters*, vol. 16, no. 8. Elsevier Sciences, Inc., 1995.

[22] A. Ghosh, S. Tsutsui, and H. Tanaka, "Genetic search with aging of individuals," *Int. J. Knowledge-based Intell. Eng. Syst.*, vol. 1, pp. 86–103, 1997.

[23] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. New York: Addison-Wesley, 1989.

[24] D. E. Goldberg, K. Deb, and B. Korb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex Systs.*, vol. 3, pp. 493–530, 1989.

[25] J. J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proc. 1st Int. Conf. Genetic Algorithms* (J. J. Grefenstette, ed.), pp. 160–168, Hillsdale: Lawrence Erlbaum Associates, 1985.

[26] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 16, pp. 122–128, 1986.

[27] S. A. Harp and T. Samad, "Genetic synthesis of neural network architecture," in *Handbook of Genetic Algorithms* (L. Davis, ed.), pp. 202

– 221, New York: Van Nostrand Reinhold, 1991.

[28] A. Haydar, M. Demirekler, and M. K. Yurtseven, "Feature selection using genetic algorithm and its application to speaker verification," *Electronics Letters*, vol. 34, pp. 1457–1459, 1998.

[29] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *Proc. 1st IEEE Int. Conf. Neural Networks*, vol. 3, (San Diego), pp. 11–14, 1987.

[30] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. New York: Addison-Wesley, 1991.

[31] A. Hill and C. J. Taylor, "Model-based image interpretation using genetic algorithms," *Image and Vision Comput.*, vol. 10, pp. 295–300, 1992.

[32] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.

[33] H. V. Hove and A. Verschoren, "Genetic algorithms and recognition problems," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 145–166, Boca Raton: CRC Press, 1996.

[34] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Acquisition of fuzzy classification knowledge using genetic algorithms," in *Proc. 3rd IEEE Int. Conf. Fuzzy Systs.*, (Orlando), pp. 1963–1968, 1994.

[35] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Trans. Fuzzy Systs.*, vol. 3, pp. 260–270, 1995.

[36] H. Ishibuchi, T. Murata, and H. Tanaka, "Construction of fuzzy classification systems with linguistic if-then rules using genetic algorithms," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 227–251, Boca Raton: CRC Press, 1996.

[37] H. Ishibuchi, M. Nii, and T. Murata, "Linguistic rule extraction from neural networks and genetic algorithm based rule selection," in *Proc. IEEE Int. Conf. Neural Networks*, (Houston), pp. 2390–2395, 1997.

[38] K. Izumi, K. Watanabe, T. Shimokawa, and K. Kiguchi, "Facial image recognition system using an RBF neural network optimally constructed by GA," in *Proc. Sixth International Conference on Soft Computing, IIZUKA,*, pp. 119–124, 2000.

[39] C. J. Janikow, "A genetic algorithm method for optimizing the fuzzy component of a fuzzy decision tree," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 253–281, Boca Raton: CRC Press, 1996.

[40] P. Jog, J. Y. Suh, and D. V. Gucht, "The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem," in *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. Schaffer, ed.), pp. 110–115, San Mateo: Morgan Kaufmann, 1989.

[41] N. Koga, D. Tominaga, and M. Okamoto, "Fast numerical optimization technique based on parallel genetic algorithm," in *Proc. Sixth International Conference on Soft Computing, IIZUKA,*, pp. 87–92, 2000.

[42] K. Krishna, *Hybrid Evolutionary Algorithms for Supervised and Unsupervised Learning*. PhD thesis, Department of Electrical Engineering, Indian Institute of Science, Bangalore, India, 1998.

[43] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, no. 2, pp. 4–22, 1987.

[44] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 39–53, 1994.

[45] U. Maulik and S. Bandyopadhyay, "Genetic algorithm based clustering technique," *Pattern Recog.*, vol. 33, pp. 1455–1465, 2000.

[46] Z. Michalewicz and C. Z. Janikow, "Genetic algorithms for numerical optimization," *Statistics and Computing*, vol. 1, pp. 75–91, 1991.

[47] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1992.

[48] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Int. Joint Conf. Art. Intell.* (N. S. Sridharan, ed.), pp. 762–767, San Mateo: Morgan Kaufmann, 1989.

[49] H. J. Muller, ed., *Studies in Genetics - Selected Papers*. Bloomington: Indiana University Press, 1962.

[50] C. A. Murthy, D. Bhandari, and S. K. Pal, "Optimal stopping time for genetic algorithms with elitist model," *Fundamenta Informaticae*, vol. 35, pp. 4–22, 1998.

[51] C. A. Murthy and N. Chowdhury, "In search of optimal clusters using genetic algorithms," *Pattern Recog. Lett.*, vol. 17, pp. 825–832, 1996.

[52] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," in *Proc. 2nd Int. Conf. Genetic Algorithms*, pp. 224–230, Hillsdale: Lawrence Erlbaum Associates, 1987.

[53] S. K. Pal and D. Bhandari, "Selection of optimal set of weights in a layered network using genetic algorithms," *Inform. Sci.*, vol. 80, pp. 213–234, 1994.

[54] S. K. Pal, D. Bhandari, and M. K. Kundu, "Genetic algorithms for optimal image enhancement," *Pattern Recog. Lett.*, vol. 15, pp. 261–271, 1994.

[55] S. K. Pal, S. Bandyopadhyay, and C. A. Murthy, "Genetic algorithms for generation of class boundaries," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, no. 6, pp. 816–828, 1998.

[56] S. K. Pal, S. De, and A. Ghosh, "Designing Hopfield type networks using genetic algorithms and its comparison with simulated annealing," *Int. J. Pattern Recog. Art. Intell.*, vol. 11, pp. 447–461, 1997.

[57] S. K. Pal, A. Ghosh, and M. K. Kundu, eds., *Soft Computing for Image Processing*. Heidelberg: Physica Verlag, 1999.

[58] S. K. Pal, T. S. Dillon, and D. S. Yeung, eds., *Soft Computing in Case Based Reasoning*. London: Springer, 2001.

[59] S. K. Pal and D. Dutta Majumder, "Fuzzy sets and decision making approaches in vowel and speaker recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-7, pp. 625–629, 1977.

[60] S. K. Pal and S. Mitra, *Neuro-fuzzy Pattern Recognition: Methods in Soft Computing*. New York: John Wiley, 1999.

[61] S. K. Pal and P. P. Wang, eds., *Genetic Algorithms for Pattern Recognition*. Boca Raton: CRC Press, 1996.

[62] M. Prakash and M. N. Murty, "A genetic approach for selection of (near-) optimal subsets of principal components for discrimination," *Pattern Recog. Lett.*, vol. 16, pp. 781–787, 1995.

[63] D. K. Pratihar, "Optimal/near optimal gait generation of a six-legged robot – A GA-fuzzy approach," in *Proc. Sixth International Conference on Soft Computing, IIZUKA,*, pp. 93–98, 2000.

[64] *Proc. Sixth International Conference on Soft Computing, IIZUKA 2000*, (Fukuoka, Japan), October, 2000.

[65] R. Reed, "Pruning algorithms – a survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, 1993.

[66] F. C. Rhee and Y. J. Lee, "Unsupervised feature selection using a fuzzy-genetic algorithm," in *Proc. IEEE Intl. Conf. Fuzzy Systems, III*, 1999.

[67] S. G. Romaniuk, "Learning to learn with evolutionary growth percep-

trons," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 179–211, Boca Raton: CRC Press, 1996.

[68] D. E. Rumelhart, J. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge: MIT Press, 1986.

[69] D. E. Rumelhart, J. McClelland, and the PDP Research Group, *Parallel Distributed Processing*, vol. 2. Cambridge: MIT Press, 1986.

[70] S. Saha and J. P. Christensen, "Genetic design of sparse feedforward neural networks," *Inform. Sci.*, vol. 79, pp. 191–200, 1994.

[71] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," *Physica D*, vol. 42, pp. 244–248, 1990.

[72] G. Seetharaman, A. Narasimahan, and L. Stor, "Image segmentation with genetic algorithms: A formulation and implementation," in *Proc. SPIE Conf. Stochastics and Neural Methods in Signal Processing and Computer Vision*, vol. 1569, (San Diego), 1991.

[73] T. L. Seng, M. B. Khalid, and R. Yusof, "Tuning of a neuro-fuzzy controller by a genetic algorithm," *IEEE Trans. on Systs., Man and Cyberns., Part B*, vol. 29, pp. 226–236, 1999.

[74] S. C. Shin and S. B. Park, "GA-based predictive control for nonlinear processes," *Electronics Letters*, vol. 34, pp. 1980–1981, 1998.

[75] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," *Pattern Recog. Lett.*, vol. 10, pp. 335–347, 1989.

[76] J. Silva, P. Simoni, and K. Bharadwaj, "A hierarchical approach to multiple-point correspondences in stereo vision using a genetic algorithm search," in *Proc. Sixth International Conference on Soft Computing, IIZUKA,*, pp. 125–132, 2000.

[77] R. Srikanth, R. George, N. Warsi, D. Prabhu, F. E. Petry, and B. P. Buckles, "A variable-length genetic algorithm for clustering and classification," *Pattern Recog. Lett.*, vol. 16, pp. 789–800, 1995.

[78] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading: Addison-Wesley, 1974.

[79] B. W. Wah, A. Ieumwananonthachai, and Y. Li, "Generalization of heuristics learned in genetics-based learning," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 87–126, Boca Raton: CRC Press, 1996.

[80] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing*, vol. 14, pp. 347–361, 1990.

Chapter 14
ROUGH SETS IN PATTERN RECOGNITION

A. Skowron* and R. Swiniarski+

**Institute of Mathematics, Warsaw University*
Banacha 2, 02-095 Warsaw, POLAND
e-mail: *skowron@mimuw.edu.pl*

+*San Diego State University*
Department of Mathematical and Computer Sciences
5500 Campanile Drive, San Diego, California 92182-7720, U.S.A.
e-mail: *rswiniar@saturn.sdsu.edu*

Abstract

We present applications of rough set methods for feature selection, feature extraction, discovery of patterns and their applications for decomposition of large data tables as well as the relationship of rough sets with association rules. Boolean reasoning is crucial for all the discussed methods. We also discuss briefly potential applications of some extensions of the classical rough set approach.

14.1 Basic rough set approach

In recent years we have witnessed a rapid growth of interest in rough set theory and its applications, worldwide. The theory has been followed by the development of several software systems that implement rough set operations, in particular for solving knowledge discovery and data mining tasks. Rough sets are applied in domains such as, medicine, finance, telecommunication, vibration analysis, conflict resolution, intelligent agents, pattern recognition, control theory, signal analysis, process industry, and marketing.

We start by presenting the basic notions of classical rough set approach [41] introduced to deal with imprecise or vague concepts.

14.1.1 Information systems

A data set can be represented by a table where each row represents, for instance, an object, a case, or an event. Every column represents an attribute, or an observation, or a property that can be measured for each object; it can also be supplied by a human expert or user. This table is called an information system. More formally, it is a pair $\mathcal{A} = (U, A)$ where U is a non-empty finite set of *objects* called the *universe* and A is a non-empty finite set of *attributes* such that $a : U \to V_a$ for every $a \in A$. The set V_a is called the value set of a. By $Inf_B(x) = \{(a, a(x)) : a \in B\}$ we denote the *information signature of x with respect to B*, where $B \subseteq A$ and $x \in U$.

14.1.2 Decision systems

In many cases the target of the classification, that is, the family of concepts to be approximated, is represented by an additional attribute called decision. Information systems of this kind are called decision systems. A decision system is any system of the form $\mathcal{A} = (U, A, d)$, where $d \notin A$ is the *decision attribute* and A is a set of conditional attributes or simply *conditions*.

Let $\mathcal{A} = (U, A, d)$ be given and let $V_d = \{v_1, \ldots, v_{r(d)}\}$. Decision d determines a partition $\{X_1, \ldots, X_{r(d)}\}$ of the universe U, where $X_k = \{x \in U : d(x) = v_k\}$ for $1 \le k \le r(d)$. The set X_i is called the *i-th decision class of* $\mathcal{A}$. By $X_d(u)$ we denote the decision class $\{x \in U : d(x) = d(u)\}$, for any $u \in U$.

One can generalize the above definition to a case of decision systems of the form $\mathcal{A} = (U, A, D)$ where the sets $D = \{d_1, ...d_k\}$ of decision attributes and A are assumed to be disjoint. Formally this system can be treated as the decision system $\mathcal{A} = (U, A, d_D)$ where $d_D(x) = (d_1(x), ..., d_k(x))$ for $x \in U$.

The decision tables can be identified with training samples known in machine learning and used to induce concept approximations in the process known as supervised learning [29].

Rough set approach allows a precise definition of the notion of concept approximation. It is based [43] on the indiscernibility relation between objects defining a partition (or covering) of the universe U of objects. The indiscernibility of objects follows from the fact that they are perceived by means of values of available attributes. Hence some objects having the same (or similar) values of attributes are indiscernible.

14.1.3 Indiscernibility relation

Let $\mathcal{A} = (U, A)$ be an information system. Then, with any $B \subseteq A$, an equivalence relation $IND_{\mathcal{A}}(B)$ is associated as follows:

$$IND_{\mathcal{A}}(B) = \{(x, x') \in U^2 \; : \; \forall a \in B \; a(x) = a(x')\}.$$

$IND_{\mathcal{A}}(B)$ (or, $IND(B)$, for short) is called the B-indiscernibility relation, and its classes are denoted by $[x]_B$. By X/B we denote the partition of U defined by the indiscernibility relation $IND(B)$.

Now we will discuss what sets of objects can be expressed (defined) by formulae constructed by means of attributes and their values. The simplest formulae, called descriptors, are of the form $a = v$ where $a \in A$ and $v \in V_a$. One can consider generalized descriptors of the form $a \in S$ where $S \subseteq V_a$. The descriptors can be combined into more complex formulae using propositional connectives. The meaning $\|\varphi\|_{\mathcal{A}}$ in $\mathcal{A}$ of formula φ is defined inductively by

(1) if φ is of the form $a = v$ then $\|\varphi\|_{\mathcal{A}} = \{x \in U \; : \; a(x) = v\}$;
(2) $\|\varphi \wedge \varphi'\|_{\mathcal{A}} = \|\varphi\|_{\mathcal{A}} \cap \|\varphi'\|_{\mathcal{A}}$; $\|\varphi \vee \varphi'\|_{\mathcal{A}} = \|\varphi\|_{\mathcal{A}} \cup \|\varphi'\|_{\mathcal{A}}$; $\|\neg\varphi\|_{\mathcal{A}} = U - \|\varphi\|_{\mathcal{A}}$.

The above definition can be easily extended to generalized descriptors.

Any set of objects $X \subseteq U$ definable in $\mathcal{A}$ by some formula φ (*i.e.*, X=$\|\varphi\|_{\mathcal{A}}$) is referred to as a crisp (exact) set. Otherwise, the set is *rough* (*inexact, vague*). Vague concepts may be approximated only by crisp concepts; these approximations are defined now [43].

14.1.4 Lower and upper approximation of sets, boundary regions

Let $\mathcal{A} = (U, A)$ be an information system and let $B \subseteq A$ and $X \subseteq U$. We can approximate X using only the information contained in B by constructing the *B-lower* and *B-upper approximations of X*, denoted $\underline{B}X$ and $\overline{B}X$ respectively, where $\underline{B}X = \{x : [x]_B \subseteq X\}$ and $\overline{B}X = \{x : [x]_B \cap X \neq \emptyset\}$.

The lower approximation corresponds to certain rules while the upper approximation to possible rules (rules with confidence greater than 0) for X. The B-lower approximation of X is the set of all objects which can be with certainty classified to X using attributes from B. The set $U - \overline{B}X$ is called the *B-outside region of X* and consists of those objects which can be classified with certainty as not belonging to X, using attributes from B. The set $BN_B(X) = \overline{B}X - \underline{B}X$, called the *$B$-boundary region of X*, thus consists of those objects that cannot be unambiguously classified into X on the basis of the attributes from B. A set is said to be *rough* (respectively *crisp*) if the boundary region is non-empty (respectively empty). Consequently each rough set has boundary-line cases, *i.e.*, objects which cannot be classified with certainty either as members of the set or of its complement. Obviously crisp sets have no boundary-line elements at all. This means that boundary-line cases cannot be properly classified by employing the available knowledge. The size of the boundary region can be used as a measure of the quality of set approximation (in U).

It can be easily seen that the lower and upper approximations of a set are, respectively, the interior and the closure of the set in the topology generated by the indiscernibility relation.

One can consider weaker indiscernibility relations defined by tolerance relations defining coverings of the universe of objects by tolerance (similarity) classes. An extension of rough set approach based on tolerance relations has been used for pattern extraction and concept approximation (see, *e.g.*, [32, 37, 65, 71]).

14.1.5 Quality measures of concept approximation and measures of inclusion and closeness of concepts

We now present some examples of measures of quality approximation as well as of inclusion and closeness (approximate equivalence). These notions are instrumental in evaluating the strength of rules and closeness of concepts as well as being applicable in determining plausible reasoning schemes [47, 54]. An important role is also played by entropy measures (see, *e.g.*, [13]).

Let us consider first an example of a measure of the quality of approximation.

Accuracy of approximation. A rough set X can be characterized numerically by the following coefficient

$$\alpha_B(X) = \frac{|\underline{B}(X)|}{|\overline{B}(X)|},$$

called the accuracy of approximation, where $|X|$ denotes the cardinality of $X \neq \emptyset$ and B is a set of attributes. Obviously, $0 \leq \alpha_B(X) \leq 1$. If $\alpha_B(X) = 1$, X is *crisp* with respect to B (X is *exact* with respect to B), and otherwise, if $\alpha_B(X) < 1$, X is *rough* with respect to B (X is *vague* with respect to B).

Rough membership function. In classical set theory either an element belongs to a set or it does not. The corresponding membership function is the characteristic function of the set, *i.e.*, the function takes values 1 and 0, respectively. In the case of rough sets the notion of membership is different. The rough membership function quantifies the degree of relative overlap between the set X and the equivalence class to which x belongs. It is defined as follows:

$$\mu_X^B(x) : U \to [0,1] \text{ and } \mu_X^B(x) = \frac{|[x]_B \cap X|}{|[x]_B|}.$$

The rough membership function can be interpreted as a frequency-based estimate of $\Pr(y \in X \mid u)$, the conditional probability that object y belongs to set X, given the information signature $u = Inf_B(x)$ of object x with respect to attributes B. The value $\mu_X^B(x)$ measures the degree of inclusion of $\{y \in U : Inf_B(x) = Inf_B(y)\}$ in X.

Positive region and its measure. If $X_1, \ldots, X_{r(d)}$ are decision classes of $\mathcal{A}$, then the set $\underline{B}X_1 \cup \ldots \cup \underline{B}X_{r(d)}$ is called the *B-positive region of* $\mathcal{A}$ and is denoted by $POS_B(d)$. The number $|POS_B(d)|/|U|$ measures a degree of inclusion of the partition defined by attributes from B into the partition defined by the decision.

Dependencies in a degree. Another important issue in data analysis is discovering dependencies among attributes. Intuitively, a set of attributes D depends totally on a set of attributes C, denoted $C \Rightarrow D$, if all values of attributes from D are uniquely determined by values of attributes from C. In other words, D depends totally on C, if there exists a functional dependency between values of D and C. Dependency can be formally defined as follows.

Let D and C be subsets of A. We will say that D *depends on* C in a *degree*

k $(0 \leq k \leq 1)$, denoted $C \Rightarrow_k D$, if

$$k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|},$$

where $POS_C(D) = POS_C(d_D)$.
Obviously,

$$\gamma(C, D) = \sum_{X \in U/D} \frac{|\underline{C}(X)|}{|U|}.$$

If $k = 1$ we say that D *depends totally* on C, and if $k < 1$, we say that D *depends partially* (to a *degree* k) on C. $\gamma(C, D)$ describes the closeness of the partition U/D and its approximation with respect to conditions from C.

The coefficient k expresses the ratio of all elements of the universe which can be properly classified to blocks of the partition U/D by employing attributes C. It will be called the *degree of the dependency*.

Inclusion and closeness in a degree. Instead of classical exact set inclusion, inclusion in a degree is often used in the process of deriving knowledge from data. A well-known measure of inclusion of two non-empty sets $X, Y \subseteq U$ is $|X \cap Y|/|X|$ [2, 47]; their closeness can be defined by

$$\min\left(|X \cap Y|/|X|, |X \cap Y|/|Y|\right).$$

14.2 Searching for knowledge

In this section, we discuss the problem of concept approximations. We point out that it is also the main goal of strategies searching for knowledge. Next we present selected methods based on rough sets and Boolean reasoning for concept approximation.

14.2.1 Concept approximation

Searching for concept approximations is a basic task in pattern recognition or machine learning. It is also crucial to knowledge discovery [14] and, in particular, to scientific discovery [24, 77]. For example, scientific discovery [77] uses, as a main source of power, relatively general knowledge, including knowledge to search combinatorial spaces. Hence, it is important to discover efficient searching strategies. This includes the processes of inducing the relevant features and functions over which these strategies are constructed as well as the structure of searching strategy induced from such constructs. The goal of knowledge discovery [24, 62] is to find knowledge that is novel, plausible and understandable. Certainly, these soft concepts should be induced up to a sufficient degree, *i.e.*, their approximations should be induced to specify the main constraints in searching for knowledge. In this sense, concept approximation is the basic step not only for machine learning or for pattern extraction but also for knowledge discovery and scientific discovery. Certainly, in the latter cases the inducing processes of concept approximations are much more complex and searching for such approximations creates a challenge for researchers.

Qualitative process representation, qualitative reasoning, spatial reasoning, perception and measurement instruments, collaboration and communication, embodied agents are only some of the potential research directions mentioned in [62] as important for scientific reasoning and discovery. The topics mentioned above are very much in the scope of computing with words [78, 79] and granular computing (see *e.g.*, [54, 67]). A rough set extension called rough mereology (see, *e.g.*, [46, 50, 51]) has been proposed as a tool for approximate reasoning to deal with such problems [56, 67]. Schemes of reasoning in rough mereology approximating soft patterns seem to be crucial for making progress in knowledge discovery. In particular, this approach has been used to build a calculus on information granules [54, 67] as a foundation for computing with words. Among the issues related to knowledge discovery using the approach discussed there, are generalized soft association rules, synthesis of interfaces between sources exchanging concepts and using different languages, and problems in spatial reasoning.

Let us now return to the discussion on inducing concept approximation by using rough set approach. First we recall the definition of a generalized approximation space, introduced in [65]. This definition helps to explain a general approach offered by rough sets for concept approximations.

Definition 14.1 A parameterized approximation space is a system

$$AS_{\#,\$} = (U, I_{\#}, \nu_{\$})$$

where

- U is a non-empty set of objects,
- $I_{\#} : U \rightarrow P(U)$ is an *uncertainty function* and $P(U)$ denotes the power set of U,

- $\nu_\$: P(U) \times P(U) \rightarrow [0,1]$ is a *rough inclusion function*,
- # and \$ are sets of parameters.

■

The uncertainty function defines for every object x, a set of objects, called the neighborhood of x, consisting of objects indistinguishable with x or similar to x. The parameters (from #) of the uncertainty function are used to search for relevant neighborhoods with respect to the task to be solved, *e.g.*, concept description.

A constructive definition of the uncertainty function can be based on the assumption that some metrics (distances) are given on attribute values. For example, if for some attribute $a \in A$ a metric $\delta_a : V_a \times V_a \longrightarrow [0,\infty)$ is given, where V_a is the set of all values of an attribute a then one can define the following uncertainty function:

$$y \in I_a^{f_a}(x) \text{ if and only if } \delta_a(a(x), a(y)) \leq f_a(a(x), a(y)),$$

where $f_a : V_a \times V_a \rightarrow [0,\infty)$ is a given threshold function.

A set $X \subseteq U$ is *definable in* $AS_{\#,\$}$ if it is a union of some values of the uncertainty function.

The rough inclusion function defines a degree of inclusion between two subsets of U [50, 65]. The parameters (from \$) of the rough inclusion function are used to search for relevant inclusion degrees with respect to the task to be solved, *e.g.*, concept description.

For a parameterized approximation space $AS_{\#,\$} = (U, I_\#, \nu_\$)$ and any subset $X \subseteq U$ the lower and the upper approximations are defined by

$LOW(AS_{\#,\$}, X) = \{x \in U : \nu_\$(I_\#(x), X) = 1\}$,

$UPP(AS_{\#,\$}, X) = \{x \in U : \nu_\$(I_\#(x), X) > 0\}$, respectively.

Any generalized approximation space consists of a family of approximation spaces creating the search space for data models. Any approximation space in this family is distinguished by some parameters. Searching strategies for optimal (sub-optimal) parameters are basic rough set tools in searching for data models and knowledge.

There are two main types of parameters. The first type is used to define object sets called neighborhoods, while the second type measures the inclusion or closeness of neighborhoods.

The basic assumption of the classical rough set approach, shared with other approaches like machine learning, pattern recognition or statistics, is that objects are perceived by means of some features, (*e.g.*, formulae being the results of measurement of the form $attribute = value$ called descriptors). Hence, some objects can be indiscernible (indistinguishable) or similar to each other. The sets of indiscernible or similar objects expressible by some formulae are called neighborhoods. In the simplest case, the family of all neighborhoods creates a partition of the universe. In more general cases, it defines a covering. Formulae defining the neighborhoods are basic building blocks from which the approximate descriptions of other sets (decision classes or concepts) are induced. Usually, like in machine learning, the specification of concepts is incomplete, *e.g.*, given by examples and counterexamples. Having incomplete specification of concepts, one can induce only approximate description of concepts by means of formulae defining the neighborhoods. Hence it follows that it will be useful to have parameterized formulae (*e.g.*, in the simplest case $a > p \wedge b < q$ where a, b are attributes and p, q are parameters) so that by tuning their parameters one can select formulae being relevant for inducing concept approximation. A formula is relevant for concept description if it defines a large neighborhood still included to a sufficient degree in approximated concept. In the simplest case the formulae defining neighborhoods are conjunctions of descriptors. Parameters to be tuned can be of different types, like the number of conjunction connectives in the formula, or the interval boundaries in case of discretization of real-valued attributes. In more general cases, these formulae can express the results of measurement or perception of observed objects and represent complex information granules. Among such granules can be decision algorithms labeled by feature value vectors (describing an actual situation in which algorithm should be performed), clusters of such granules defined by their similarity, or hierarchical structures of such granules (see, *e.g.*, [67]). These complex granules become more and more important for qualitative reasoning, in particular for spatial reasoning [55].

Assuming that a partition (covering) of objects has been fixed, the set approximations are induced by tuning of parameters specifying the degree of set inclusion.

In this way concept approximations are induced from data using rough set approach.

14.2.2 Discernibility and Boolean reasoning

We have pointed out that rough set approach has been introduced by Pawlak [43] to deal with vague or imprecise concepts. More generally, it is an approach for deriving knowledge from data and for reasoning about knowledge derived from data. Searching for knowledge is usually guided by some constraints [24]. A wide class of such constraints can be expressed using rough set setting or its generalizations (like rough mereology [46], or granular computing [54]). Knowledge derived from data by rough set approach consists of different constructs. Among them are basic for rough set approach constructs, called reducts, different kinds of rules (like decision rules or association rules) dependencies, patterns (templates) or classifiers. The reducts are of special importance because all other constructs can be derived from different kinds of reducts using rough set approach. Searching strategies for reducts are based on Boolean (propositional) reasoning [5] because constraints (*e.g.*, related to discernibility of objects) are expressible by propositional formulae. Moreover, using Boolean reasoning data models with the minimum description length [29, 58] can be induced because they correspond to some constructs of Boolean functions called prime implicants (or their approximations). Searching for knowledge can be performed in the language close to data or in a language with more abstract concepts what is closely related to problems of feature selection and feature extraction in Machine Learning or Pattern Recognition [29]. Let us also mention that data models derived from data by using rough set approach are controlled using statistical test procedures (for more details see, *e.g.*, [12, 13]).

In this chapter, we present examples showing how the general scheme outlined above is used for deriving knowledge.

At this point, a brief discussion on Boolean reasoning is presented, as most methods discussed later are based on generation of reducts using Boolean reasoning.

14.2.3 Boolean reasoning

The combination of rough set approach with Boolean Reasoning [5] has created a powerful methodology allowing to formulate and efficiently solve searching problems for different kinds of reducts and their approximations.

The idea of Boolean reasoning is based on the construction, for a given problem P, of a corresponding Boolean function f_P with the following property: the solutions for the problem P can be recovered from prime implicants of f_P. An implicant of a Boolean function f is any conjunction of literals (variables or their negations) such that if the values of these literals are true under an arbitrary valuation v of variables, then the value of the function f under v is also true. A prime implicant is a minimal implicant.

Searching strategies for data models under a given partition of objects are based, using rough set approach, on discernibility and Boolean reasoning (see, *e.g.*, [32, 37, 50, 51, 66, 71, 72]). This process also covers the tuning of parameters like thresholds that are used to extract relevant partition (or covering), to measure the degree of inclusion (or closeness) of sets, or the parameters measuring the quality of approximation.

It is necessary to deal with Boolean functions of large size to solve real-life problems. However, a successful methodology based on the discernibility of objects and Boolean reasoning has been developed for the computation of many constructs important for applications, like reducts and their approximations, decision rules, association rules, discretization of real-valued attributes, symbolic value grouping, searching for new features defined by oblique hyperplanes or higher-order surfaces, pattern extraction from data as well as conflict resolution or negotiation. Reducts are also basic tools for the extraction, from data, of functional dependencies or functional dependencies in a degree (for references see the papers and bibliography in [39, 50, 51, 66]).

Most of the problems related to generation of the constructs mentioned above are of high computational complexity (*i.e.*, they are NP-complete or NP-hard). This also shows that most of the problems related to, *e.g.*, feature selection, pattern extraction from data have intrinsically high computational complexity. However, using methodology developed on the basis on discernibility and Boolean reasoning, it was possible to discover efficient heuristics that return suboptimal solutions to the problems.

The reported results of experiments on many data sets are very promising. They show a very high quality of solutions (expressed in terms of the classification quality of unseen objects and the time needed for solution construction) generated by the heuristics, in comparison with other methods reported in literature. Moreover, for large data sets the decomposition methods based on patterns called templates have been developed (see, *e.g.*, [36, 37]) as well as a method to deal with large relational databases (see, *e.g.*, [33]). The first one is based on the decomposition of large data into regular subdomains which are of size feasible for the methods developed. We will discuss this method later. The second (see, *e.g.*, [33]) has shown that Boolean reasoning methodology can be extended to large relational data bases. The main idea is based on the observation that relevant Boolean variables, for a very large formula (corresponding to analyzed relational data base), can be discovered by analyzing some statistical information. This statistical information can be efficiently extracted from large data bases.

Another interesting statistical approach is based on different sampling strategies. Samples are analyzed using the developed strategies, and stable constructs for a sufficiently large number of samples are considered as relevant for the whole table. This approach has been successfully used for generating different kinds of so-called dynamic reducts (see, *e.g.*, [4]). It has been used, for example, for the generation of dynamic decision rules. Experiments on different data sets have proved that these methods are promising for large data sets.

Our approach is strongly related to propositional reasoning [61] and further advancement in propositional reasoning will result in more progress in the development of the methods discussed. It is important to note that the methodology allows construction of heuristics having a very important *approximation property*, which can be formulated as follows:
Expressions, (*i.e.*, implicants) generated by heuristics *close* to prime implicants, define approximate solutions for the problem [61].

In the sequel, we will discuss in greater detail different kinds of reducts and their applications for deriving different forms of knowledge from data.

14.2.4 Reducts in information systems and decision systems

We start from reducts of information systems. Given an information system $\mathcal{A} = (U, A)$, a *reduct* is a minimal set of attributes $B \subseteq A$ such that $IND_{\mathcal{A}}(B) = IND_{\mathcal{A}}(A)$. In other words, a reduct is a minimal set of attributes from A that preserves the original classification defined by the set A of attributes. Finding a minimal reduct is NP-hard [63]; one can also show that for any m there exists an information system with m attributes having an exponential number of reducts. There exist fortunately good heuristics that compute sufficiently many reducts in an acceptable time.

Let $\mathcal{A}$ be an information system with n objects. The *discernibility matrix* of $\mathcal{A}$ is a symmetric $n \times n$ matrix with entries c_{ij} as given below. Each entry consists of the set of attributes upon which objects x_i and x_j differ.

$$c_{ij} = \{a \in A \mid a(x_i) \neq a(x_j)\} \text{ for } i, j = 1, ..., n.$$

A *discernibility function* $f_{\mathcal{A}}$ for an information system $\mathcal{A}$ is a Boolean function of m Boolean variables $a_1^*, ..., a_m^*$ (corresponding to the attributes $a_1, ..., a_m$) defined by

$$f_{\mathcal{A}}(a_1^*,...,a_m^*) = \bigwedge\left\{\bigvee c_{ij}^* \mid 1 \leq j \leq i \leq n,\ c_{ij} \neq \emptyset\right\},$$

where $c_{ij}^* = \{a^* \mid a \in c_{ij}\}$. In the sequel we will write a_i instead of a_i^*.

The discernibility function $f_{\mathcal{A}}$ describes constraints which should be satisfied if one wishes to preserve discernibility between all pairs of discernible objects from $\mathcal{A}$. It requires retention of at least one attribute from each non-empty entry of the discernibility matrix, *i.e.*, corresponding to any pair of discernible objects. One can show [63] that the sets of all minimal sets of attributes preserving discernibility between objects, *i.e.*, reducts correspond to prime implicants of the discernibility function $f_{\mathcal{A}}$. The intersection of all reducts is the so-called *core*.

In general, the decision is not constant on the indiscernibility classes. Let $\mathcal{A} = (U, A, d)$ be a decision system. The *generalized decision in* $\mathcal{A}$ is the function $\partial_A : U \longrightarrow \mathcal{P}(V_d)$ defined by

$$\partial_A(x) = \{i \mid \exists x' \in U\ x'\, \mathit{IND}(A)\, x \text{ and } d(x') = i\}.$$

A decision system $\mathcal{A}$ is called *consistent (deterministic)*, if $|\partial_A(x)| = 1$ for any $x \in U$, otherwise $\mathcal{A}$ is *inconsistent (non-deterministic)*. Any set consisting of all objects with the same generalized decision value is called the *generalized decision class*.

It is easy to see that a decision system $\mathcal{A}$ is consistent if and only if $\mathit{POS}_A(d) = U$. Moreover, if $\partial_B = \partial_{B'}$, then $\mathit{POS}_B(d) = \mathit{POS}_{B'}(d)$ for any pair of non-empty sets $B, B' \subseteq A$. Hence the definition of a decision-relative reduct:

Definition 14.2 A subset $B \subseteq A$ is a *relative reduct* if it is a minimal set such that $POS_A(d) = POS_B(d)$. ■

Decision-relative reducts may be found from a discernibility matrix: $M^d(\mathcal{A}) = (c_{ij}^d)$ assuming $c_{ij}^d = c_{ij} - \{d\}$ if $(|\partial_A(x_i)| = 1$ or $|\partial_A(x_j)| = 1)$ and $\partial_A(x_i) \neq \partial_A(x_j)$, $c_{ij}^d = \emptyset$, otherwise. The matrix $M^d(\mathcal{A})$ is called the *decision-relative discernibility matrix of* $\mathcal{A}$. Construction of *the decision-relative discernibility function* from this matrix follows the construction of the discernibility function from the discernibility matrix. One can show [63] that the set of *prime implicants* of $f_M^d(\mathcal{A})$ defines the set of all *decision–relative reducts* of $\mathcal{A}$.

In some applications, instead of reducts we prefer to use their approximations called α-reducts, where $\alpha \in [0, 1]$ is a real parameter. For a given information system $\mathcal{A} = (U, A)$, the set of attributes $B \subseteq A$ is called α-reduct if B has nonempty intersection with at least $100\alpha\%$ of nonempty sets $c_{i,j}$ of the discernibility matrix of $\mathcal{A}$.

14.2.5 Reducts and Boolean reasoning: examples of applications

We will present examples showing how rough set methods, in combination with Boolean reasoning, can be used for solving several KDD problems. A crucial component of our approach are rough set constructs called reducts. They are (prime) implicants of suitably chosen Boolean functions expressing discernibility conditions which should be preserved during reduction.

14.2.6 Feature selection

Selection of relevant features is an important problem and has been extensively studied in machine learning and pattern recognition (see, *e.g.*, [29]). It is also a very active research area in the rough set community.

One of the first ideas [43] was to consider the *core* of the reduct set of the information system $\mathcal{A}$ as the source of relevant features. One can observe that relevant feature sets (in a sense used by the machine learning community) can be interpreted in most cases as the decision–relative reducts of decision systems obtained by adding appropriately constructed decisions to a given information system.

Another approach is related to dynamic reducts (for references see, *e.g.*, [49]). The attributes are considered relevant if they belong to dynamic reducts with a sufficiently high stability coefficient, *i.e.*, they appear with sufficiently high frequency in random samples of a given information system. Several experiments [49] show that the set of decision rules based on such attributes is much smaller than the set of all decision rules. At the same time, the quality of classification of new objects increases or does not change if one only considers rules constructed over such relevant features.

The idea of attribute reduction can be generalized by introducing a concept of *significance of attributes* which enables the evaluation of attributes not only in the two-valued scale *dispensable–indispensable* but also in the multi-valued case by assigning to an attribute a real number from the interval [0,1] that expresses the importance of an attribute in the information table. Significance of an attribute can be evaluated by measuring the effect of removal of the attribute from an information table.

Let C and D be sets of condition and decision attributes, respectively, and let $a \in C$ be a condition attribute. It was shown previously that the number $\gamma(C, D)$ expresses the degree of dependency between attributes C and D, or the accuracy of the approximation of U/D by C. It may be now checked how the coefficient $\gamma(C, D)$ changes when attribute a is removed. In other words, what is the difference between $\gamma(C, D)$ and $\gamma((C - \{a\}, D)$? The difference is normalized and the significance of attribute a is defined by

$$\sigma_{(C,D)}(a) = \frac{(\gamma(C, D) - \gamma(C - \{a\}, D))}{\gamma(C, D)}$$

$$= 1 - \frac{\gamma(C - \{a\}, D)}{\gamma(C, D)}.$$

Coefficient $\sigma_{C,D}(a)$ can be understood as a classification error which occurs when attribute a is dropped. The significance coefficient can be extended to sets of attributes as follows:

$$\sigma_{(C,D)}(B) = \frac{(\gamma(C, D) - \gamma(C - B, D))}{\gamma(C, D)}$$

$$= 1 - \frac{\gamma(C - B, D)}{\gamma(C, D)}.$$

Another possibility is to consider as relevant the features that come from approximate reducts of sufficiently high quality.

Any subset B of C can be treated as an *approximate reduct* of C and the number

$$\varepsilon_{(C,D)}(B) = \frac{(\gamma(C, D) - \gamma(B, D))}{\gamma(C, D)} = 1 - \frac{\gamma(B, D)}{\gamma(C, D)},$$

is called an *error of reduct approximation.* It expresses how exactly the set of attributes B approximates the set of condition attributes C with respect to determining D.

Several other methods of reduct approximation based on measures different from positive region have been developed. All experiments confirm the hypothesis that by tuning the level of approximation the classification quality of new objects may be increased in most cases. It is important to note that it is once again possible to use Boolean reasoning to compute the different types of reducts and to extract from them relevant approximations.

14.2.7 Feature extraction

Non-categorical attributes must be discretized in a preprocessing step. The discretization step determines how coarsely we want to view the world. Discretization is a step that is not specific to the rough set approach. A majority of rule or tree induction algorithms require it in order to perform well. The search for appropriate cut-off points can be reduced to finding some minimal Boolean expressions called prime implicants.

Discretization can be treated as a search for coarser partitions of the universe still relevant for inducing concept description of high quality. We will also show that this basic problem can be reduced to the computation of basic constructs of rough sets, namely, reducts, of some systems. Hence it follows that we can estimate the computational complexity of the discretization problems. Moreover, heuristics for computing reducts and prime implicants can be used here. The general heuristics can be modified to more optimal ones using knowledge about the problem, *e.g.*, natural order of the set of reals. Discretization is only an illustrative example of many other problems with the same property.

The rough set community has been committed to the construction of efficient algorithms for (new) feature extraction. Rough set methods combined with Boolean reasoning [5] lead to several successful approaches to feature extraction. The most successful methods are:

- discretization techniques,
- methods of partitioning of nominal attribute value sets, and
- combinations of the above methods.

Search for new features expressed by multi-modal formulae can be mentioned in this context. Structural objects can be interpreted as models (so-called Kripke models) of such formulae and the problem of searching for relevant features reduces to construction of multi-modal formulae expressing properties of the structural objects, discerning objects or sets of objects [38]. For more details, the reader is referred to the bibliography in [50].

The reported results show that discretization problems and symbolic value partition problems are of high computational complexity (*i.e.*, NP-complete or NP-hard), which clearly justifies the importance of designing efficient heuristics. The idea of discretization is illustrated with a simple example.

Example 14.2.1 Let us consider a (consistent) decision system (see Table 14.1(a)) with two conditional attributes a and b and seven objects

Table 14.1 The discretization process: (a) The original decision system $\mathcal{A}$. (b) The **P**-discretization of $\mathcal{A}$, where $\mathbf{P} = \{(a, 0.9), (a, 1.5), (b, 0.75), (b, 1.5)\}$

$\mathbf{A}$	a	b	d
u_1	0.8	2	1
u_2	1	0.5	0
u_3	1.3	3	0
u_4	1.4	1	1
u_5	1.4	2	0
u_6	1.6	3	1
u_7	1.3	1	1

(a)

$\Rightarrow$

$\mathbf{A}^{\mathbf{P}}$	a^P	b^P	d
u_1	0	2	1
u_2	1	0	0
u_3	1	2	0
u_4	1	1	1
u_5	1	2	0
u_6	2	2	1
u_7	1	1	1

(b)

$u_1, ..., u_7$. The values of the attributes of these objects and the values of the decision d are presented in Table 14.1.

The sets of possible values of a and b are defined by:

$$V_a = [0, 2) ; V_b = [0, 4) .$$

The sets of values of a and b for objects from U are respectively given by:

$$a(U) = \{0.8, 1, 1.3, 1.4, 1.6\} \text{ and}$$
$$b(U) = \{0.5, 1, 2, 3\}$$

□

A discretization process produces a partition of the value sets of the conditional attributes into intervals. The partition is done so that a consistent decision system is obtained from a given consistent decision system by a substitution of the original value of any object in $\mathcal{A}$ by the (unique) name of the interval(s) in which it is contained. In this way, the size of the value sets of the attributes may be reduced. If a given decision system is not consistent one can transform it to the consistent decision system by taking the generalized decision instead of the original one. Next, it is possible to apply the above method. It will return cuts with the following property: regions bounded by them consist of objects with the same generalized decision. Certainly, one can consider also *soft (impure)* cuts and induce the relevant cuts on their basis (see the bibliography in [49]).

Example 14.2.2 The following intervals are obtained in our example system:

$[0.8, 1)$; $[1, 1.3)$; $[1.3, 1.4)$; $[1.4, 1.6)$ for a);

$[0.5, 1)$; $[1, 2)$; $[2, 3)$ for b).

The idea of cuts can be introduced now. Cuts are pairs (a, c) where $c \in V_a$. Our considerations are restricted to cuts defined by the middle points of the above intervals. In our example the following cuts are obtained:

$(a, 0.9)$; $(a, 1.15)$; $(a, 1.35)$; $(a, 1.5)$;

$(b, 0.75)$; $(b, 1.5)$; $(b, 2.5)$.

Any cut defines a new conditional attribute with binary values. For example, the attribute corresponding to the cut $(a, 1.2)$ is equal to 0 if $a(x) < 1.2$; otherwise it is equal to 1. □

Any set P of cuts defines a new conditional attribute a_P for any a. Given a partition of the value set of a by cuts from P put the unique names for the elements of this partition.

Example 14.2.3 Let

$$P = \{(a, 0.9), (a, 1.5), (b, 0.75), (b, 1.5)\}$$

be the set of cuts. These cuts glue together the values of a smaller then 0.9, all the values in interval $[0.9, 1.5)$ and all the values in interval $[1.5, 4)$. A similar construction can be repeated for b. The values of the new attributes a_P and b_P are shown in Table 14.1 (b). □

The next natural step is to construct a set of cuts with a minimal number of elements. This may be done using Boolean reasoning.

Let $\mathcal{A} = (U, A, d)$ be a decision system where $U = \{x_1, x_2, \ldots, x_n\}, A = \{a_1, \ldots, a_k\}$ and $d : U \longrightarrow \{1, \ldots, r\}$. We assume $V_a = [l_a, r_a) \subset \Re$ to be a real interval for any $a \in A$ and $\mathcal{A}$ to be a consistent decision system. Any pair (a, c) where $a \in A$ and $c \in \Re$ will be called a *cut on* V_a. Let $\mathbf{P}_a = \{[c_0^a, c_1^a), [c_1^a, c_2^a), \ldots, [c_{k_a}^a, c_{k_a+1}^a)\}$ be a partition of V_a (for $a \in A$) into subintervals for some integer k_a, where $l_a = c_0^a < c_1^a < c_2^a < \ldots < c_{k_a}^a < c_{k_a+1}^a = r_a$ and $V_a = [c_0^a, c_1^a) \cup [c_1^a, c_2^a) \cup \ldots \cup [c_{k_a}^a, c_{k_a+1}^a)$. It follows that any partition $\mathbf{P}_a$ is uniquely defined and is often identified with the set of cuts

$$\{(a, c_1^a), (a, c_2^a), \ldots, (a, c_{k_a}^a)\} \subset A \times \Re.$$

Given $\mathcal{A} = (U, A, d)$ any set of cuts $\mathbf{P} = \bigcup_{a \in A} \mathbf{P}_a$ defines a new decision system $\mathcal{A}^{\mathbf{P}} = (U, A^{\mathbf{P}}, d)$ called **P**-*discretization of* $\mathcal{A}$, where $A^{\mathbf{P}} = \{a^{\mathbf{P}} : a \in A\}$ and $a^{\mathbf{P}}(x) = i \Leftrightarrow a(x) \in [c_i^a, c_{i+1}^a)$ for $x \in U$ and $i \in \{0, .., k_a\}$.

Two sets of cuts $\mathbf{P}'$ and $\mathbf{P}$ are equivalent, written $\mathbf{P}' \equiv_{\mathbf{A}} \mathbf{P}$, if and only if $\mathcal{A}^{\mathbf{P}} = \mathcal{A}^{\mathbf{P}'}$. The equivalence relation $\equiv_{\mathcal{A}}$ has a finite number of equivalence classes. Equivalent families of partitions will not be discerned in the sequel.

The set of cuts $\mathbf{P}$ is called $\mathcal{A}$-*consistent* if $\partial_{\mathcal{A}} = \partial_{\mathcal{A}^{\mathbf{P}}}$, where $\partial_{\mathcal{A}}$ and $\partial_{\mathcal{A}^{\mathbf{P}}}$ are generalized decisions of $\mathcal{A}$ and $\mathcal{A}^{\mathbf{P}}$, respectively. An $\mathcal{A}$-consistent set of cuts $\mathbf{P}^{irr}$ is $\mathcal{A}$-*irreducible* if $\mathbf{P}$ is not $\mathcal{A}$-consistent for any $\mathbf{P} \subset \mathbf{P}^{irr}$. The $\mathcal{A}$-consistent set of cuts $\mathbf{P}^{opt}$ is $\mathcal{A}$-*optimal* if $card(\mathbf{P}^{opt}) \leq card(\mathbf{P})$ for any $\mathcal{A}$-consistent set of cuts $\mathbf{P}$.

It can be shown that the decision problem of checking whether, for a given decision system $\mathcal{A}$ and an integer k, there exists an irreducible set of cuts $\mathbf{P}$ in $\mathcal{A}$ such that $card(\mathbf{P}) < k$ is NP-complete. The problem of searching for an optimal set of cuts $\mathbf{P}$ in a given decision system $\mathcal{A}$ is NP-hard.

Despite these complexity bounds it is possible to devise efficient heuristics that return semi-minimal sets of cuts. The simplest heuristic is based on Johnson's strategy. The strategy is first to look for a cut discerning a maximal number of object pairs and then to eliminate all already discerned object pairs. This procedure is repeated until all object pairs to be discerned are discerned. It is interesting to note that this heuristics can be realized by computing the minimal relative reduct of the corresponding decision system. The *"MD heuristic"* is analogous to Johnson's approximation algorithm. It may be formulated as follows:

ALGORITHM: MD-heuristics(A semi-optimal family of partitions)

Step 1. *Construct table* $\mathcal{A}^* = (U^*, A^*)$ *from* $\mathcal{A} = (U, A, d)$ *where* U^* *is the set of pairs* (x, y) *of objects to be discerned by* d *and* A^* *consists of attribute* c^* *for any cut* c *and* c^* *is defined by* $c^*(x, y) = 1$ *if and only if* c *discerns* x *and* y *(i.e.,* x, y *are in different half-spaces defined by* c*); set* $\mathcal{B} = \mathcal{A}^*$;

Step 2. *Choose a column from* $\mathcal{B}$ *with the maximal number of occurrences of 1's;*

Step 3. *Delete from* $\mathcal{B}$ *the column chosen in Step 2 and all rows marked with 1 in this column;*

Step 4. *If* $\mathcal{B}$ *is non-empty then go to Step 2, else Stop.*

This algorithm searches for a cut which discerns the largest number of pairs of objects (MD-heuristic). Then the cut c is moved from A^* to the resulting set of cuts $\mathbf{P}$; and all pairs of objects discerned by c are removed from U^*. The algorithm continues until U^* becomes empty.

Let n be the number of objects and let k be the number of attributes of decision system $\mathcal{A}$. The following inequalities hold:

$$card\,(A^*) \leq (n-1)\,k$$

and

$$card\,(U^*) \leq \frac{n\,(n-1)}{2}.$$

It is easy to observe that for any cut $c \in A^*$, $O(n^2)$ steps are required in order to find the number of all pairs of objects discerned by c. A straightforward realization of this algorithm therefore requires $O(kn^2)$ of memory space and $O(kn^3)$ steps in order to determine one *cut*. This approach is clearly impractical. However, it is possible to observe that in the process of searching for the set of pairs of objects discerned by currently analyzed cut from an increasing sequence of cuts, one can use information about such set of pairs of objects computed for the previously considered cut. The MD-heuristic using this observation [31] determines the best cut (for a given attribute) in $O(kn)$ steps using $O(kn)$ space only. This heuristic is reported to be very efficient with respect to the time necessary for decision rules generation as well as with respect to the quality of unseen object classification.

Let us observe that in the considered case of discretization the new features are of the form $a \in V$, where $V \subseteq V_a$ and V_a is the set of the values of attribute a.

We report some results of experiments on data sets using this heuristic. We would like to comment, for example, on the result of classification received by application of this heuristic to Shuttle data (Table 14.3). The result concerning classification quality is the same as the best result reported in [28] but the time is of an order better than that of the best result from [28]. In the table, we also present the results of experiments with heuristic searching for features defined by oblique hyperplanes. This heuristic has been developed using genetic algorithms to tune the position of a hyperplane to the optimal one [31]. In this way one can implement propositional reasoning using some background knowledge about the problem.

In our experiments we have chosen several data tables with real value attributes from the U.C. Irvine repository. For some tables, taking into account

Table 14.2 Data tables stored in the UC Irvine Repository

Names	No. of class	Train. table	Test. table	Best results
Australian	2	690×14	CV5	85.65%
Glass	7	214×9	CV5	69.62%
Heart	2	270×13	CV5	82.59%
Iris	3	150×4	CV5	96.00%
Vehicle	4	846×19	CV5	69.86%
Diabetes	2	768×8	CV5	76.04%
SatImage	6	4436×36	2000	90.06%
Shuttle	6	43500×7	14500	99.99%

Table 14.3 Results of experiments on Machine Learning data

Data tables	Diagonal cuts		Hyperplanes	
	#cuts	quality	#cuts	quality
Australian	18	79.71%	16	82.46%
Glass	14±1	67.89%	12	70.06%
Heart	11±1	79.25%	11±1	80.37%
Iris	7±2	92.70%	6±2	96.7%
Vehicle	25	59.70%	20±2	64.42%
Diabetes	20	74.24%	19	76.08%
SatImage	47	81.73%	43	82.90%
Shuttle	15	99.99%	15	99.99%

the small number of objects, we have adopted the approach based on five-fold cross-validation ($CV-5$). The results obtained (Table 14.3) can be compared with those reported in [11, 28] (Table 14.2). For predicting decisions on new cases we apply only decision rules generated either by the decision tree (using hyperplanes) or by rules generated in parallel with discretization.

For some tables the classification quality of our algorithm is better than that of the C4.5 or naive-Bayes induction algorithms [57] even when used with different discretization methods [7, 11, 28].

Comparing this method with the other methods reported in [28], we can conclude that the algorithms discussed have the shortest run-time and a good overall classification quality. (In many cases our results were the best in comparison to many other methods reported in literature).

We would like to stress that the induction of the minimal number of relevant cuts is equivalent to the computation of the minimal reduct of decision system constructed from the system $\mathcal{A}^*$ discussed above [31]. This, in turn, as we have shown, is equivalent to the problem of computation of minimal prime implicants of Boolean functions. This is only an illustration of a wide class of basic problems of machine learning, pattern recognition and KDD which can be reduced to problems of relevant reduct computation.

Our next illustrative example concerns symbolic (nominal, qualitative) attribute value grouping. We also present some experimental results of heuristics based on the developed methods in case of mixed nominal and numeric attributes.

In case of symbolic value attribute (*i.e.*, without pre-assumed order on values of given attributes) the problem of searching for new features of the form $a \in V$ is, in a sense, from a practical point of view, more complicated than that for real-valued attributes. However, it is possible to develop efficient heuristics for this case using Boolean reasoning.

Let $\mathcal{A} = (U, A, d)$ be a decision table. Any function $P_a : V_a \to \{1, \ldots, m_a\}$ (where $m_a \leq |V_a|$) is called *a partition of* V_a. The *rank of* P_{a_i} is the value $rank(P_{a_i}) = |P_{a_i}(V_{a_i})|$. The family of partitions $\{P_a\}_{a \in B}$ is *consistent with* B (B-consistent) if and only if the condition $[(u, u') \notin IND(B)$ and $d(u) \neq d(u')$ implies $\exists_{a \in B}$ such that $[P_a(a(u)) \neq P_a(a(u'))]]$ holds for any $(u, u') \in U$. It means that if two objects u, u' are discerned by B and d, then they must be discerned by partition attributes defined by $\{P_a\}_{a \in B}$. We consider the following optimization problem:

Symbolic value partition problem. Given a decision table $\mathcal{A} = (U, A, d)$ and a set of attributes $B \subseteq A$, search for the minimal B-consistent family of partitions, (*i.e.*, such a B-consistent family $\{P_a\}_{a \in B}$ that

$$\sum_{a \in B} rank(P_a)$$

is minimal).

To discern between pairs of objects we will use new binary features $a_v^{v'}$ (for $v \neq v'$) defined by $a_v^{v'}(x, y) = 1$ if and only if $a(x) = v \neq v' = a(y)$. One can apply the Johnson heuristic for the new matrix with these attributes to search for minimal set of new attributes that discerns all pairs of objects from different decision classes. After extraction of these sets, we construct for each attribute a_i a graph $\Gamma_a = \langle V_a, E_a \rangle$, where E_a is defined as the set of all new attributes (propositional variables) found for the attribute a. Any vertex coloring of Γ_a defines a partition of V_a. The colorability problem is solvable in polynomial time for $k = 2$, but remains NP-complete for all $k \geq 3$. As for discretization, one can apply some efficient heuristic searching for optimal partition.

Let us consider an example of decision table presented in Fig. 14.1 and (a reduced form) of its discernibility matrix (Fig. 14.1). From the Boolean function $f_{\mathcal{A}}$ with Boolean variables of the form $\mathbf{a}_{v_1}^{v_2}$, one can find the shortest prime implicant:

$$\mathbf{a}_{a_2}^{a_1} \wedge \mathbf{a}_{a_3}^{a_2} \wedge \mathbf{a}_{a_4}^{a_1} \wedge \mathbf{a}_{a_4}^{a_3} \wedge \mathbf{b}_{b_4}^{b_1} \wedge \mathbf{b}_{b_4}^{b_2} \wedge \mathbf{b}_{b_3}^{b_2} \wedge \mathbf{b}_{b_3}^{b_1} \wedge \mathbf{b}_{b_5}^{b_3}$$

which can be represented by graphs (see Fig. 14.2).

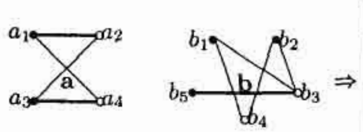
$\mathcal{A}$	$\mathbf{a}$	$\mathbf{b}$	$\mathbf{d}$
u_1	a_1	b_1	0
u_2	a_1	b_2	0
u_3	a_2	b_3	0
u_4	a_3	b_1	0
u_5	a_1	b_4	1
u_6	a_2	b_2	1
u_7	a_2	b_1	1
u_8	a_4	b_2	1
u_9	a_3	b_4	1
u_{10}	a_2	b_5	1

$\Rightarrow$

$\mathcal{M}(\mathcal{A})$	u_1	u_2	u_3	u_4
u_5	$\mathbf{b}_{b_4}^{b_1}$	$\mathbf{b}_{b_4}^{b_2}$	$\mathbf{a}_{a_2}^{a_1}, \mathbf{b}_{b_4}^{b_3}$	$\mathbf{a}_{a_3}^{a_1}, \mathbf{b}_{b_4}^{b_1}$
u_6	$\mathbf{a}_{a_2}^{a_1}, \mathbf{b}_{b_2}^{b_1}$	$\mathbf{a}_{a_2}^{a_1}$	$\mathbf{b}_{b_3}^{b_2}$	$\mathbf{a}_{a_3}^{a_2}, \mathbf{b}_{b_2}^{b_1}$
u_7	$\mathbf{a}_{a_2}^{a_1}$	$\mathbf{a}_{a_2}^{a_1}, \mathbf{b}_{b_2}^{b_1}$	$\mathbf{b}_{b_3}^{b_1}$	$\mathbf{a}_{a_3}^{a_2}$
u_8	$\mathbf{a}_{a_4}^{a_1}, \mathbf{b}_{b_2}^{b_1}$	$\mathbf{a}_{a_4}^{a_1}$	$\mathbf{a}_{a_4}^{a_2}, \mathbf{b}_{b_3}^{b_2}$	$\mathbf{a}_{a_4}^{a_3}, \mathbf{b}_{b_2}^{b_1}$
u_9	$\mathbf{a}_{a_3}^{a_1}, \mathbf{b}_{b_4}^{b_1}$	$\mathbf{a}_{a_3}^{a_1}, \mathbf{b}_{b_4}^{b_2}$	$\mathbf{a}_{a_3}^{a_2}, \mathbf{b}_{b_4}^{b_3}$	$\mathbf{b}_{b_4}^{b_1}$
u_{10}	$\mathbf{a}_{a_2}^{a_1}, \mathbf{b}_{b_5}^{b_1}$	$\mathbf{a}_{a_2}^{a_1}, \mathbf{b}_{b_5}^{b_2}$	$\mathbf{b}_{b_5}^{b_3}$	$\mathbf{a}_{a_3}^{a_2}, \mathbf{b}_{b_5}^{b_1}$

Fig. 14.1 The decision table and the discernibility matrix

We can color vertices of these graphs as shown in Fig. 14.2. The colors



$\mathbf{a}^{P_a}$	$\mathbf{b}^{P_b}$	d
1	1	0
2	2	0
1	2	1
2	1	1

Fig. 14.2 Coloring of attribute value graphs and the reduced table

correspond to the partitions

$$\begin{aligned}
&P_{\mathbf{a}}(a_1) = P_{\mathbf{a}}(a_3) = 1;\\
&P_{\mathbf{a}}(a_2) = P_{\mathbf{a}}(a_4) = 2;\\
&P_{\mathbf{b}}(b_1) = P_{\mathbf{b}}(b_2) = P_{\mathbf{b}}(b_5) = 1;\\
&P_{\mathbf{b}}(b_3) = P_{\mathbf{b}}(b_4) = 2.
\end{aligned}$$

At the same time one can construct the new decision table (Fig. 14.2).

One can extend the presented approach (see *e.g.*, [32]) to the case when in a given decision system nominal as well as numeric attributes appear. The received heuristics are of very good quality. Experiments for classification methods (see [32]) have been carried over decision systems using two techniques called *"train-and-test"* and *"n-fold-cross-validation"*. In Table 14.4, we present some results of experiments obtained by testing the proposed methods – MD (using only discretization based on MD-heuristic with Johnson approximation strategy [31, 66]) and MD-G (using discretization and symbolic value grouping [37, 66]) – for classification quality on some data tables from the "UC Irvine repository". The results reported in [15] are summarized in columns labeled by S-ID3 and C4.5 in Table 14.4. Let us note that the heuristics MD and MD-G are also very efficient with respect to time-complexity.

In the case of real-valued attributes, one can search for features in the feature set that contains the characteristic functions of half-spaces determined by hyperplanes or parts of spaces defined by more complex surfaces in the multi-dimensional spaces. Genetic algorithms have been applied in searching for semi-optimal hyperplanes [31]. The results reported show substantial increase in the quality of classification of unseen objects but at the price of increased time needed for searching for the semi-optimal hyperplane.

Table 14.4 Quality comparison of various decision tree methods. Abbreviations: MD: MD-heuristic; MD-G: MD-heuristic with symbolic value partition

Names of Tables	Classification accuracies			
	S-ID3	C4.5	MD	MD-G
Australian	78.26	85.36	83.69	84.49
Breast (L)	62.07	71.00	69.95	69.95
Diabetes	66.23	70.84	71.09	76.17
Glass	62.79	65.89	66.41	69.79
Heart	77.78	77.04	77.04	81.11
Iris	96.67	94.67	95.33	96.67
Lympho	73.33	77.01	71.93	82.02
Monk-1	81.25	75.70	100	93.05
Monk-2	69.91	65.00	99.07	99.07
Monk-3	90.28	97.20	93.51	94.00
Soybean	100	95.56	100	100
TicTacToe	84.38	84.02	97.7	97.70
Average	78.58	79.94	85.48	87.00

14.2.8 Decision rules

Reducts serve the purpose of inducing *minimal* decision rules. Any such rule contains the minimal number of descriptors in the conditional part so that their conjunction defines the largest subset of a generalized decision class (decision class, if the decision table is deterministic). Hence, information included in the conditional part of any minimal rule is sufficient for prediction of the generalized decision value for all objects satisfying this part. The conditional parts of minimal rules define largest object sets relevant for generalized decision class approximation. It turns out that the conditional parts of minimal rules can be computed (by using Boolean reasoning) as reducts relative to objects or local reducts (see *e.g.*, [4, 64]). Once the reducts have been computed, the conditional parts of rules are easily constructed by laying the reducts over the original decision system and reading off the values. In the discussed case the generalized decision value is preserved during the reduction. One can consider stronger constraints which should be preserved. For example, in [69] the constraints are described by probability distributions corresponding to information signatures of objects. Again the same methodology can be used to compute the reducts corresponding to these constraints.

The main challenge in inducing rules from decision systems lies in determining which attributes should be included in the conditional part of the rule. Using the strategy outlined above, first the minimal rules are computed. Their conditional parts describe largest object sets (definable by conjunctions of descriptors) with the same generalized decision value in a given decision system. Hence, they create the largest sets still relevant for defining the decision classes (or sets of decision classes when the decision system is inconsistent). Although such minimal decision rules can be computed, this approach can result in a set of rules of unsatisfactory classification quality. Such detailed rules will be overfit and they will classify unseen cases poorly. Shorter rules should rather be synthesized. Although they will not be perfect on the known cases there is a good chance that they will be of high quality when classifying new cases. They can be constructed by computing approximations of the reducts mentioned above. Approximations of reducts received by dropping some descriptors from the conditional parts of minimal rules define larger sets, not purely included in decision classes but included to a satisfactory degree. It means that these shorter descriptions can be more relevant for decision class (concept) approximation than the exact reducts. Hence, *e.g.*, one can expect that if, by dropping the descriptor from the conditional part we receive the description of the object set almost included in the approximated decision class, then this descriptor is a good candidate for dropping.

Several other strategies have been implemented. Methods of boundary region thinning [83] are based, *e.g.*, on the idea that sets of objects, included in decision classes to a satisfactory degree, can be treated as parts of the lower approximations of decision classes. Hence the lower approximations of decision classes are enlarged and decision rules generated for them are usually stronger, (*e.g.*, they are supported by more examples). The degree of inclusion is tuned experimentally to achieve, *e.g.*, high classification quality of new cases. One can also adopt an idea of dynamic reducts for decision rule generation.

For estimation of the quality of approximation of decision classes, global measures based on the positive region [64] or entropy [13] are used. When a set of rules has been induced from a decision system containing a set of training examples, they can be used to classify new objects. However, to resolve conflict between different decision rules recognizing new objects one should develop strategies for resolving conflicts between them when they are voting for different decisions (see the bibliography in [50] and [51]). Recently, it has been shown that rough set methods can be used to learn from data the strategy for conflict-resolution between decision rules when they are classifying new objects, contrary to existing methods that use some fixed strategies [74].

14.2.9 α-reducts and association rules

In this section we discuss a relationship between association rules [2] and approximations of reducts being basic constructs of rough sets [34, 64, 66].

We consider formulae called *templates* as being conjunction of descriptors. The templates will be denoted by $\mathbf{T}$, $\mathbf{P}$, $\mathbf{Q}$ and descriptors by D with or without subscripts. By $support_{\mathcal{A}}(\mathbf{T})$ is denoted the cardinality of $\|\mathbf{T}\|_{\mathcal{A}}$ and by $confidence_{\mathcal{A}}(\mathbf{P} \to \mathbf{Q})$ is denoted the number

$$support_{\mathcal{A}}(\mathbf{P} \wedge \mathbf{Q}) / support_{\mathcal{A}}(\mathbf{P}).$$

The reduct approximations mentioned above are descriptions of the object sets matched by templates. They describe these sets in an approximate sense expressed by coefficients called support and confidence.

There are two main steps in many association rule generation methods developed for a given information system $\mathcal{A}$ and parameters of support s and confidence c:

(1) Extraction from data of as many as possible templates $\mathbf{T} = D_1 \wedge D_2 ... \wedge D_k$ such that $support_{\mathcal{A}}(\mathbf{T}) \geq s$ and $support_{\mathcal{A}}(\mathbf{T} \wedge D) < s$ for any descriptor D different from descriptors of $\mathbf{T}$ (*i.e.*, generation of maximal templates among those supported by more than s objects);
(2) Searching for a partition $\mathbf{T} = \mathbf{P} \wedge \mathbf{Q}$ for any generated template $\mathbf{T}$ satisfying the following conditions:
    (a) $support_{\mathcal{A}}(\mathbf{P}) < \frac{support_{\mathcal{A}}(\mathbf{T})}{c}$
    (b) $\mathbf{P}$ has the shortest length among templates satisfying the previous condition.

The second step can be solved using rough set methods and Boolean reasoning approach. Let $\mathbf{T} = D_1 \wedge D_2 \wedge \ldots \wedge D_m$ be a template with $support_{\mathcal{A}}(\mathbf{T}) \geq s$. For a given confidence threshold $c \in (0; 1)$ the decomposition $\mathbf{T} = \mathbf{P} \wedge \mathbf{Q}$ is called c-irreducible if $confidence_{\mathcal{A}}(\mathbf{P} \to \mathbf{Q}) \geq c$ and for any decomposition $\mathbf{T} = \mathbf{P}' \wedge \mathbf{Q}'$ such that $\mathbf{P}'$ is a sub-template of $\mathbf{P}$, we have

$$confidence_{\mathcal{A}}(\mathbf{P}' \rightarrow \mathbf{Q}') < c.$$

Now we are going to explain why and how the problem of searching for c-irreducible association rules from the given template is equivalent to the problem of searching for local α-reducts (for some α) from a decision table. The last problem is a well-known one in rough set theory.

Let us define a new decision table $\mathcal{A}|_{\mathbf{T}} = (U, A|_{\mathbf{T}}, d)$ from the original information system $\mathcal{A}$ and the template $\mathbf{T}$ by

(1) $A|_{\mathbf{T}} = \{a_{D_1}, a_{D_2}, ..., a_{D_m}\}$ is a set of attributes corresponding to the descriptors of $\mathbf{T}$ such that

$$a_{D_i}(u) = \begin{cases} 1 \text{ if the object } u \text{ satisfies } D_i, \\ 0 \text{ otherwise.} \end{cases}$$

(2) the decision attribute d determines if the object satisfies template $\mathbf{T}$, *i.e.*,

$$d(u) = \begin{cases} 1 \text{ if the object } u \text{ satisfies } \mathbf{T}, \\ 0 \text{ otherwise.} \end{cases}$$

The following facts [34, 66] describe the relationship between association rules and approximations of reducts:

For the given information table $\mathcal{A} = (U, A)$, the template $\mathbf{T}$, the set of descriptors $\mathbf{P}$, the implication $\left(\bigwedge_{D_i \in \mathbf{P}} D_i \longrightarrow \bigwedge_{D_j \notin \mathbf{P}} D_j\right)$ is

(1) a 100%-irreducible association rule from $\mathbf{T}$ if and only if $\mathbf{P}$ is a reduct in $\mathcal{A}|_{\mathbf{T}}$.

(2) a c-irreducible association rule from $\mathbf{T}$ if and only if $\mathbf{P}$ is an α-reduct of $\mathcal{A}|_{\mathbf{T}}$, where $\alpha = 1 - (\frac{1}{c} - 1)/(\frac{n}{s} - 1)$, n is the total number of objects from U and $s = support_{\mathcal{A}}(\mathbf{T})$.

One can show that the problem of searching for the shortest α-reducts is NP-hard [34]. From the above facts it follows that extracting association rules from data is strongly related to extraction from the data reduct approximations [34], being basic constructs of rough sets.

The following example illustrates the main idea of our method. Let us consider the information system $\mathcal{A}$ presented in Table 14.5 with 18 objects and 9 attributes.

Table 14.5 The example of information table $\mathcal{A}$ and template $\mathbf{T}$ supported by 10 objects and the new decision table $\mathcal{A}|_{\mathbf{T}}$ constructed from $\mathcal{A}$ and template $\mathbf{T}$

$\mathcal{A}$	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
u_1	0	1	1	1	80	2	2	2	3
u_2	0	1	2	1	81	0	aa	1	aa
u_3	0	2	2	1	82	0	aa	1	aa
u_4	0	1	2	1	80	0	aa	1	aa
u_5	1	1	2	2	81	1	aa	1	aa
u_6	0	2	1	2	81	1	aa	1	aa
u_7	1	2	1	2	83	1	aa	1	aa
u_8	0	2	2	1	81	0	aa	1	aa
u_9	0	1	2	1	82	0	aa	1	aa
u_{10}	0	3	2	1	84	0	aa	1	aa
u_{11}	0	1	3	1	80	0	aa	2	aa
u_{12}	0	2	2	2	82	0	aa	2	aa
u_{13}	0	2	2	1	81	0	aa	1	aa
u_{14}	0	3	2	2	81	2	aa	2	aa
u_{15}	0	4	2	1	82	0	aa	1	aa
u_{16}	0	3	2	1	83	0	aa	1	aa
u_{17}	0	1	2	1	84	0	aa	1	aa
u_{18}	1	2	2	1	82	0	aa	2	aa

$\mathcal{A}\|_{\mathbf{T}}$	D_1	D_2	D_3	D_4	D_5	d
	$a_1 = 0$	$a_3 = 2$	$a_4 = 1$	$a_6 = 0$	$a_8 = 1$	
u_1	1	0	1	0	0	
u_2	1	1	1	1	1	1
u_3	1	1	1	1	1	1
u_4	1	1	1	1	1	1
u_5	0	1	0	0	1	
u_6	1	0	0	0	1	
u_7	0	0	0	0	1	
u_8	1	1	1	1	1	1
u_9	1	1	1	1	1	1
u_{10}	1	1	1	1	1	1
u_{11}	1	0	1	1	0	
u_{12}	1	0	0	1	0	
u_{13}	1	1	1	1	1	1
u_{14}	1	1	0	0	0	
u_{15}	1	1	1	1	1	1
u_{16}	1	1	1	1	1	1
u_{17}	1	1	1	1	1	1
u_{18}	0	1	1	1	0	

Assume the template

$$\mathbf{T} = (a_1 = 0) \wedge (a_3 = 2) \wedge (a_4 = 1) \wedge (a_6 = 0) \wedge (a_8 = 1)$$

has been extracted from the information table $\mathcal{A}$. We have

$$support(\mathbf{T}) = 10 \text{ and } length(\mathbf{T}) = 5.$$

The new decision table so constructed, $\mathcal{A}|_{\mathbf{T}}$, is presented in Table 14.5. The discernibility function for $\mathcal{A}|_{\mathbf{T}}$ is of the following form:

$$\begin{aligned} & f(D_1, D_2, D_3, D_4, D_5) \\ & = (D_2 \vee D_4 \vee D_5) \wedge (D_1 \vee D_3 \vee D_4) \wedge (D_2 \vee D_3 \vee D_4) \\ & \quad \wedge (D_1 \vee D_2 \vee D_3 \vee D_4) \wedge (D_1 \vee D_3 \vee D_5) \\ & \quad \wedge (D_2 \vee D_3 \vee D_5) \wedge (D_3 \vee D_4 \vee D_5) \wedge (D_1 \vee D_5) \end{aligned}$$

After simplification we obtain six reducts corresponding to the prime implicants: $f(D_1, D_2, D_3, D_4, D_5) = (D_3 \wedge D_5) \vee (D_4 \wedge D_5) \vee (D_1 \wedge D_2 \wedge D_3) \vee (D_1 \wedge D_2 \wedge D_4) \vee (D_1 \wedge D_2 \wedge D_5) \vee (D_1 \wedge D_3 \wedge D_4)$ for the decision table $\mathcal{A}|_{\mathbf{T}}$. Thus, we have found from $\mathbf{T}$ six association rules with (100%)-confidence.

$\mathcal{M}(\mathcal{A}\vert_{\mathbf{T}})$	u_2, u_3, u_4, u_8, u_9 $u_{10}, u_{13}, u_{15}, u_{16}, u_{17}$
u_1	$D_2 \vee D_4 \vee D_5$
u_5	$D_1 \vee D_3 \vee D_4$
u_6	$D_2 \vee D_3 \vee D_4$
u_7	$D_1 \vee D_2 \vee D_3 \vee D_4$
u_{11}	$D_1 \vee D_3 \vee D_5$
u_{12}	$D_2 \vee D_3 \vee D_5$
u_{14}	$D_3 \vee D_4 \vee D_5$
u_{18}	$D_1 \vee D_5$

$= 100\% \rightarrow$

$D_3 \wedge D_5 \longrightarrow D_1 \wedge D_2 \wedge D_4$
$D_4 \wedge D_5 \longrightarrow D_1 \wedge D_2 \wedge D_3$
$D_1 \wedge D_2 \wedge D_3 \longrightarrow D_4 \wedge D_5$
$D_1 \wedge D_2 \wedge D_4 \longrightarrow D_3 \wedge D_5$
$D_1 \wedge D_2 \wedge D_5 \longrightarrow D_3 \wedge D_4$
$D_1 \wedge D_3 \wedge D_4 \longrightarrow D_2 \wedge D_5$

$= 90\% \rightarrow$

$D_1 \wedge D_2 \longrightarrow D_3 \wedge D_4 \wedge D_5$
$D_1 \wedge D_3 \longrightarrow D_3 \wedge D_4 \wedge D_5$
$D_1 \wedge D_4 \longrightarrow D_2 \wedge D_3 \wedge D_5$
$D_1 \wedge D_5 \longrightarrow D_2 \wedge D_3 \wedge D_4$
$D_2 \wedge D_3 \longrightarrow D_1 \wedge D_4 \wedge D_5$
$D_2 \wedge D_5 \longrightarrow D_1 \wedge D_3 \wedge D_4$
$D_3 \wedge D_4 \longrightarrow D_1 \wedge D_2 \wedge D_5$

Table 14.6 The simplified version of discernibility matrix $\mathcal{M}(\mathcal{A}|_{\mathbf{T}})$ and association rules

If $c = 90\%$ it means that we would like to find α-reducts for the decision table $\mathcal{A}|_{\mathbf{T}}$, where $\alpha = 1 - \frac{\frac{1}{c}-1}{\frac{n}{s}-1} = 0.86$. Hence we would like to search for a set of descriptors that covers at least $\lceil (n-s)(\alpha) \rceil = \lceil 8 \cdot 0.86 \rceil = 7$ elements of the discernibility matrix $\mathcal{M}(\mathcal{A}|_{\mathbf{T}})$. One can see that the following sets of descriptors: $\{D_1, D_2\}$, $\{D_1, D_3\}$, $\{D_1, D_4\}$, $\{D_1, D_5\}$, $\{D_2, D_3\}$, $\{D_2, D_5\}$,

$\{D_3, D_4\}$ have nonempty intersection with exactly 7 members of the discernibility matrix $\mathcal{M}(\mathcal{A}|_{\mathrm{T}})$.

In Table 14.6 we present all association rules corresponding to those sets. Heuristics searching for α-reducts are discussed, *e.g.*, in [34].

14.2.10 Decomposition of large data tables

Several methods based on rough sets have been developed to deal with large data tables, *e.g.*, to generate strong decision rules for them. We will discuss one of the methods based on decomposition of tables by using patterns, called templates, describing regular sub-domains of the universe (*e.g.*, they describe large number of customers having large number of common features).

Long templates with large support are preferred in many data mining tasks. Several quality functions can be used to compare templates. For example, they can be defined by

$$quality^1_{\mathcal{A}}(\mathrm{T}) = support_{\mathcal{A}}(\mathrm{T}) + length(\mathrm{T})$$

and

$$quality^2_{\mathcal{A}}(\mathrm{T}) = support_{\mathcal{A}}(\mathrm{T}) \times length(\mathrm{T}).$$

Problems of high quality templates generation (by using different optimization criteria) are of high computational complexity. However, efficient heuristics have been developed for solving them (see, *e.g.*, [2, 37, 81]).

Templates extracted from data are used to decompose large data tables. In consequence, the decision tree is built with internal nodes labeled by the templates extracted from data, and outgoing edges are labeled by 0 (false) and 1 (true). Any leaf is labeled by a subtable (subdomain) consisting of all objects from the original table matching all templates or their complements appearing on the path from the root of the tree to the leaf. The process of decomposition is continued until the size of subtables attached to leaves is feasible for existing algorithms (*e.g.*, decision rules for them can be generated efficiently) based on rough set methods. The reported experiments are showing that such decomposition returns interesting patterns of regular subdomains of large data tables (for references see [36, 37, 50, 51]).

It is also possible to search for patterns that are almost included in the decision classes, *i.e.*, default rules [30]. For a presentation of generating default rules see the bibliography in [50, 51].

14.3 Hybrid methods

Several methods based on hybridization of rough set methods with other soft computing methods for pattern recognition have been reported (see, *e.g.*, [3, 39, 73, 74, 75]). In the sequel, we discuss two of them briefly.

14.3.1 Rough sets as a front-end of neural network-based texture classifiers

The article [73] describes an application of the rough sets approach to feature selection and reduction as a front-end of neural network-based texture image recognition. The methods applied include singular value decomposition for feature extraction, principal components analysis for feature projection and reduction, and rough set-based methods for feature selection and reduction. For texture classification the feedforward backpropagation neural networks were applied. Numerical experiments show the ability of rough sets to select reduced set of pattern features (minimizing the pattern size), while providing better generalization of the neural network-based texture classifiers.

14.3.2 Neuro-wavelet classifiers for EEG signals based on rough set methods

EEG is one of the most important sources of information in therapy of epilepsy. Several researchers tried to address the issue of decision support for such a data. In [74, 75], a tool for noise-resistant classification of EEG signals has been presented. The experiments reported are related to data connected to dissemination of different kinds of epilepsy. By identifying relevant features in the signal, an automatic system that gives diagnostic support to a physician is provided. A novel and reliable classifier architecture has been proposed through the application of wavelets, frequential analysis, rough sets and dynamic scaling in connection with simple neural networks. Experiments prove that the proposed method provides extended robustness and generalization abilities as well as the possibility to direct interpretation of the results obtained.

14.4 Conclusions

Substantial progress has been made in the development of rough set methods (like methods for extraction from data rules, partial or total dependencies, methods for elimination of redundant data, methods dealing with missing data, dynamic data and others) and reported, *e.g.*, in [8, 9, 10, 17, 19, 25, 30, 31, 39, 50, 51, 53, 85]. New methods for extracting patterns from data (see *e.g.*, [21, 22, 30, 36, 45]), decomposition of decision systems (see *e.g.*, [36]) as well as a new methodology for data mining in distributed and multi-agent systems (see, *e.g.*, [50]) have been reported. Recently, rough set based methods have been proposed for data mining in very large relational data bases.

There are numerous areas of successful applications of rough set software systems (see [51] and www page `http://www.idi.ntnu.no/~aleks/rosetta/` for the ROSETTA system). Many interesting case studies are reported (for references see *e.g.*, [39, 50, 51] and also the bibliographies in the books [9, 17, 21, 76, 85], in particular).

We have mentioned some generalizations of the rough set approach, like the rough mereological approach (see, *e.g.*, [46, 54]). Rough mereology has been developed as a tool for synthesis of objects satisfying a given specification to a satisfactory degree. Applications of rough mereology in areas like granular computing, spatial reasoning and data mining in distributed environment have recently been reported.

Several other generalizations of rough sets have been investigated and some of them have been used for real life data analysis (see, *e.g.*, [6, 16, 23, 26, 40, 41, 49, 60, 83]).

Finally, we would like to point out that the algebraic and logical aspects of rough sets have been intensively studied since the beginning of rough set theory. The reader interested in that topic is referred to the bibliography in [50].

Acknowledgment

This work has been supported by the Wallenberg Foundation, by the ESPRIT–CRIT 2 project #20288, and by grant 8 T11C 025 19 from the State Committee for Scientific Research (KBN) of the Republic of Poland.

References

[1] Ågotnes, T., Komorowski, J., Loken, T. (1999) "Taming large rule models in rough set approaches" *Proceedings of the 3rd European Conference of Principles and Practice of Knowledge Discovery in Databases*, September 15–18, 1999, Prague, Czech Republic, Lecture Notes in Artificial Intelligence **1704**, Springer–Verlag, Berlin, 193.

[2] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkano, A. (1996) "Fast discovery of association rules", In: Fayyad, U. M., Piatetsky–Shapiro, G., Smyth P., Uthurusamy R. (Eds.), *Advances in Knowledge Discovery and Data Mining*, The AAAI Press/The MIT Press, Menlo Park, CA, 307.

[3] Banerjee, M., Mitra, S., and Pal, S.K. (1998) "Rough-fuzzy MLP: knowledge encoding and classification," *IEEE Transactions on Neural Networks*, **9**, 1203.

[4] Bazan, J.G. (1998) "A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision system" In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery* **1***: Methodology and Applications*, Physica–Verlag, Heidelberg, 321.

[5] Brown, F.M. (1990) *Boolean Reasoning*, Kluwer Academic Publishers, Dordrecht.

[6] Cattaneo, G. (1998)" Abstract approximation spaces for rough theories", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery* **1***: Methodology and Applications*, Physica–Verlag, Heidelberg, 59.

[7] Chmielewski, M.R., Grzymala–Busse, J.W. (1994) "Global discretization of attributes as preprocessing for machine learning", *Proceedings of the Third International Workshop on Rough Sets and Soft Computing* (RSSC'94), San Jose State University, San Jose, California, USA, November 10–12, 294.

[8] Cios, J., Pedrycz, W., Swiniarski, R.W. (1998) *Data Mining in Knowledge Discovery*, Kluwer Academic Publishers, Dordrecht.

[9] Czyżewski, A. (1998) "Soft processing of audio signals", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery* **2***: Applications, Case Studies and Software Systems*, Physica–Verlag, Heidelberg, 147.

[10] Deogun, J., Raghavan, V., Sarkar, A., Sever, H. (1997) "Data mining: trends in research and development", In: Lin, T.Y., Cercone, N. (Eds.)

Rough Sets and Data Mining. Analysis of Imprecise Data, Kluwer Academic Publishers, Boston, 9.

[11] Dougherty, J., Kohavi, R., Sahami, M. (1995) "Supervised and unsupervised discretization of continuous features" *Proceedings of the Twelfth International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA.

[12] Düntsch, I., Gediga, G. (1997) "Statistical evaluation of rough set dependency analysis" *International Journal of Human–Computer Studies* **46**, 589.

[13] Düntsch, I., Gediga, G. (2000) "Rough set data analysis", *Encyclopedia of Computer Science and Technology*, Marcel Dekker, New York (to appear).

[14] Fayyad, U., Piatetsky–Shapiro, G. (Eds.) (1996) *Advances in Knowledge Discovery and Data Mining*, MIT/AAAI Press, Menlo Park.

[15] Friedman, J., Kohavi, R., Yun, Y. (1996) "Lazy decision trees", *Proceedings of AAAI–96*, 717.

[16] Greco, S., Matarazzo, B., Słowiński, R. (1998) "Rough approximation of a preference relation in a pairwise comparison table", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems*, Physica–Verlag, Heidelberg, 13.

[17] Grzymala–Busse, J.W. (1998) "Applications of the rule induction system LERS", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery* **1**: *Methodology and Applications*, Physica–Verlag, Heidelberg, 366.

[18] Huber, P.J. (1981) *Robust Statistics*, Wiley, New York.

[19] Komorowski, J., Żytkow, J. (Eds.) (1997) *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery* (PKDD'97). June 25–27, Trondheim, Norway, Lecture Notes in Artificial Intelligence **1263**, Springer–Verlag, Berlin.

[20] Komorowski, J., Pawlak, Z., Polkowski, L., Skowron, A. (1999) "Rough sets: A tutorial", In: S. K. Pal and A. Skowron (Eds.), *Rough-Fuzzy Hybridization: A New Trend in Decision-Making*, Springer–Verlag, Singapore, 3.

[21] Kowalczyk, W. (1998) "Rough data modeling, A new technique for analyzing data", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, Physica–Verlag, Heidelberg, 400.

[22] Krawiec, K., Slowiński, R., Vanderpooten, D. (1998) "Learning decision rules from similarity based rough approximations", In: In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery* **2**: *Applications, Case Studies and Software Systems*, Physica–Verlag, Heidelberg, 37.

[23] Kryszkiewicz, M. (1997) "Generation of rules from incomplete information systems", In: Komorowski, J., Żytkow, J. (Eds.) (1997) *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery* (PKDD'97). June 25–27, Trondheim, Norway, Lecture Notes in Artificial Intelligence **1263**, Springer–Verlag, Berlin, 156.

[24] Langley, P., Simon, H.A., Bradshaw, G.L., Żytkow, J.M. (1987) *Scientific Discovery, Computational Explorations of the Creative Processes*, The MIT Press, Cambridge, Massachusetts.

[25] Lin, T.Y., Cercone, N. (Eds.) (1997) *Rough Sets and Data Mining. Analysis of Imprecise Data*, Kluwer Academic Publishers, Boston.

[26] Lin, T.Y. (1998) "Granular computing on binary relations I, II", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, Physica–Verlag, Heidelberg, 107.

[27] Marek, V.M., Truszczyński, M. (1999) "Contributions to the theory of rough sets", *Fundamenta Informaticae* **39**, 389.

[28] Michie, D., Spiegelhalter, D.J., Taylor, C.C. (Eds.) (1994) *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, New York.

[29] Mitchell, T. M. (1997) *Machine Learning*, McGraw–Hill, Portland.

[30] Mollestad, T., Komorowski, J. (1998) "A rough set framework for propositional default rules data mining", In: S. K. Pal and A. Skowron (Eds.), *Rough–Fuzzy Hybridization: A New Trend in Decision Making*, Springer–Verlag, Singapore.

[31] Nguyen, H. S. (1997) *Discretization of real value attributes: Boolean reasoning approach*, Ph.D. Dissertation, Warsaw University.

[32] Nguyen, H. S., Nguyen, S. H. (1998) "Pattern extraction from data", Fundamenta Informaticae **34**, 129.

[33] Nguyen, H. S. (1999) "Efficient SQL–learning method for data mining in large data bases", *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (IJCAI'99), 806.

[34] Nguyen, H. S., Nguyen, S. H. (1999) "Rough sets and association rule generation", *Fundamenta Informaticae* **40**, (4), 383.

[35] Nguyen H. S., Nguyen, S. H., Skowron A. (1999), "Decomposition of Task Specification", *Proceedings of the 11th International Symposium on Foundations of Intelligent Systems*, June 1999, Warsaw, Poland, Lecture Notes in Computer Science **1609**, Springer–Verlag, Berlin, 310.

[36] Nguyen, S. H., Skowron, A., Synak, P. (1998) "Discovery of data patterns with applications to decomposition and classification problems", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems*, Physica–Verlag, Heidelberg, 55.

[37] Nguyen, S. H. (2000) *Data regularity analysis and applications in data mining*, Ph.D. Dissertation, Warsaw University.

[38] Orlowska, E. (Ed.) (1998) *Incomplete Information, Rough Set Analysis*, Physica–Verlag, Heidelberg.

[39] Pal, S.K., Skowron, A. (Eds.) (1999) *Rough–Fuzzy Hybridization: A New Trend in Decision Making*, Springer–Verlag, Singapore.

[40] Paun, G., Polkowski, L., Skowron, A. (1996) "Parallel communicating grammar systems with negotiations", *Fundamenta Informaticae* **28**, (3-4), 315.

[41] Pawlak, Z. (1981) "Information systems–theoretical foundations", *Information Systems* **6**, 205.

[42] Pawlak, Z. (1982) "Rough sets", *International Journal of Computer and Information Sciences* **11**, 341.

[43] Pawlak, Z. (1991) *Rough Sets–Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht.

[44] Pawlak, Z., Skowron, A. (1999) "Rough set rudiments", *Bulletin of the International Rough Set Society*, **3**, 181.

[45] Piasta, Z., Lenarcik, A. (1998) "Rule induction with probabilistic rough classifiers", *Machine Learning* (to appear).

[46] Polkowski, L., Skowron, A. (1996) "Rough mereology: A new paradigm for approximate reasoning", *International Journal of Approximate Reasoning* **15**, 333.

[47] Polkowski, L., Skowron, A. (1996) "Adaptive decision–making by systems of cooperative intelligent agents organized on rough mereological principles", *Journal of the Intelligent Automaton and Soft Computing* **2**, 121.

[48] Polkowski, L., Skowron, A. (1998) "Towards adaptive calculus of granules", *Proceedings of the FUZZ-IEEE'98 International Conference*, Anchorage, Alaska, USA, May 5–9, 111.

[49] Polkowski, L., Skowron, A. (1998) "Rough sets: A perspective", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery* **1**: *Methodology and Applications*, Physica–Verlag, Heidelberg, 31.

[50] Polkowski, L., Skowron, A. (Eds.) (1998) *Rough Sets in Knowledge Discovery* **1**: *Methodology and Applications*, Physica–Verlag, Heidelberg.

[51] Polkowski, L., Skowron, A. (Eds.) (1998) *Rough Sets in Knowledge Discovery* **2**: *Applications, Case Studies and Software Systems*, Physica–Verlag, Heidelberg.

[52] Polkowski, L., Skowron, A. (1998) *Rough mereological foundations for design, analysis, synthesis, and control in distributive systems, Information Sciences* **104**, 129.

[53] Polkowski, L., Skowron, A. (Eds.) (1998) *Proceedings of the First International Conference on Rough Sets and Soft Computing* (RSCTC'98). Warszawa, Poland, June 22–27, Lecture Notes in Artificial Intelligence **1424** Springer–Verlag, Berlin.

[54] Polkowski, L., Skowron, A. (1999) "Towards adaptive calculus of granules", In: Zadeh, L.A., Kacprzyk, J. (Eds.) (1999) *Computing with Words in Information/Intelligent Systems* **1–2**, Physica–Verlag, Heidelberg, **1**, 201.

[55] Polkowski, L., Skowron, A. (2000) Rough mereology in information systems. A case study: Qualitative spatial reasoning. In: Polkowski, L., Tsumoto, S., Lin, T.Y. (Eds.) (2000) *Rough Sets: New Developments*, Physica–Verlag, Heidelberg (in print).

[56] Polkowski, L., Tsumoto, S., Lin, T.Y. (Eds.) (2000) *Rough Sets: New Developments*, Physica–Verlag, Heidelberg (in print).

[57] Quinlan, J.R. (1993) *C4.5. Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA.

[58] Rissanen, J.J. (1978) "Modeling by shortest data description" *Automatica* **14**, 465.

[59] Roddick J.F., Spiliopoulou, M. (1999) "A bibliography of temporal, spatial, and temporal data mining research", *Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining* **1**, 34.

[60] Ras, Z.W. (1996) "Cooperative knowledge–based systems", *Journal of the Intelligent Automaton and Soft Computing* **2**, 193.

[61] Selman, B., Kautz, H., McAllester, D. (1997) "Ten challenges in propositional reasoning and search", *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (IJCAI'97) 1, Nagoya, Aichi, Japan, 50.

[62] Shrager, J., Langley, P. (1990) "Computational Approaches to Scientific Discovery", In: Shrager, J., Langley, P. (Eds.), *Computational Models of Scientific Discovery and Theory Formation*, Morgan Kaufmann, San Mateo, 1.

[63] Skowron, A., Rauszer, C. (1992) "The discernibility matrices and functions in information systems", In: Slowiński, R. (Ed.) *Intelligent Decision Support – Handbook of Applications and Advances of the Rough Sets Theory*, Kluwer Academic Publishers, Dordrecht, 331.

[64] Skowron, A. (1995) "Synthesis of adaptive decision systems from experimental data", In: A. Aamodt, J. Komorowski (eds), *Proceedings of the Fifth Scandinavian Conference on Artificial Intelligence* (SCAI'95), May 1995, Trondheim, Norway, IOS Press, Amsterdam, 220.

[65] Skowron, A., Stepaniuk, J. (1996) "Tolerance approximation spaces", *Fundamenta Informaticae* **27**, 245.

[66] Skowron, A., Nguyen, H.S. (1999)" Boolean reasoning scheme with some applications in data mining", *Proceedings of the 3-rd European Conference on Principles and Practice of Knowledge Discovery in Databases*, September 1999, Prague Czech Republic, Lecture Notes in Computer Science **1704**, 107.

[67] Skowron, A., Stepaniuk, J., Tsumoto, S. (1999) "Information Granules for spatial reasoning", *Bulletin of the International Rough Set Society* **3**, 147.

[68] Stepaniuk, J. (1998) "Approximation spaces, reducts and representatives", In: Polkowski, L., Skowron, A. (Eds.) *Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems*, Physica–Verlag, Heidelberg, 109.

[69] Ślęzak, D. (1998) "Approximate reducts in decision tables", In: *Proceedings of the Sixth International Conference, Information Processing and Management of Uncertainty in Knowledge–Based Systems* (IPMU'96) **3**, July 1–5, Granada, Spain, 1159.

[70] Slowiński, R. (Ed.) (1992) *Intelligent Decision Support–Handbook of Applications and Advances of the Rough Sets Theory*, Kluwer Academic Publishers, Dordrecht.

[71] Slowiński, R., Vanderpooten, D. (1997) "Similarity relation as a basis for rough approximations", In: P. Wang (Ed.): *Advances in Machine Intelligence & Soft Computing*, Bookwrights, Raleigh NC, 17.

[72] Slowiński, R., Vanderpooten, D. (1999) "A generalized definition of rough approximations based on similarity", *IEEE Transactions on Data and Knowledge Engineering* (to appear).

[73] Swiniarski, R., Hargis, L. (2001) Rough sets as a front-end of neural networks texture classifiers, *Neurocomputing* **36**, 85.

[74] Szczuka, M. (2000) *Symbolic and neural network methods for classifiers construction*, Ph.D. Dissertation, Warsaw University.

[75] Szczuka, M., Wojdyllo, P. (2001) "Neuro-wavelet classifiers for EEG signals based on rough set methods", *Neurocomputing* **36**, 103.

[76] Tsumoto, S. (1998) "Modeling diagnostic rules based on rough sets", In: Polkowski, L., Skowron, A. (Eds.) (1998) *Proceedings of the First International Conference on Rough Sets and Soft Computing* (RSCTC'98). Warszawa, Poland, June 22–27, Lecture Notes in Artificial Intelligence **1424**, Springer–Verlag, Berlin, 475.

[77] Valdz-Prez, R.E. (1999) "Discovery tools for science apps.", *Communications of the ACM* **42**, 37.

[78] Zadeh, L.A. (1996) "Fuzzy logic = computing with words", *IEEE Transactions on Fuzzy Systems* **4**, 103.

[79] Zadeh, L.A. (1997) "Toward a theory of fuzzy information granulation and its certainty in human reasoning and fuzzy logic", *Fuzzy Sets and Systems* **90**, 111.

[80] Zadeh, L.A., Kacprzyk, J. (Eds.) (1999) *Computing with Words in Information/Intelligent Systems* **1–2**. Physica–Verlag, Heidelberg.

[81] Zaki, M.J., Parthasarathy, S. , Ogihara, M., Li, W. (1997) "New parallel algorithms for fast discovery of association rules", *Data Mining and Knowledge Discovery : An International Journal*, special issue on Scalable High–Performance Computing for KDD **1**, 343.

[82] Zhong, N., Skowron, A., Ohsuga, S. (Eds.) (1999) *Proceedings of the 7-th International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular–Soft Computing* (RSFDGrC'99) Yamaguchi, November 9-11, 1999, Lecture Notes in Artificial Intelligence **1711**,

Springer–Verlag, Berlin.

[83] Ziarko, W. (1993) "Variable precision rough set model", *Journal of Computer and System Sciences* **46**, 39.

[84] Ziarko, W. (Ed.) (1994) *Rough Sets, Fuzzy Sets and Knowledge Discovery (RSKD'93)*, Workshops in Computing, Springer–Verlag & British Computer Society, London, Berlin.

[85] Ziarko, W. (1998) "Rough sets as a methodology for data mining", In: L. Polkowski A, Skowron (Eds.), *Rough Sets in Knowledge Discovery 1: Methods and Applications*, Physica–Verlag, Heidelberg, 554.

Chapter 15

COMBINING CLASSIFIERS: SOFT COMPUTING SOLUTIONS

L. I. Kuncheva

School of Informatics
University of Wales, Bangor
Bangor, Gwynedd, LL57 1UT, UK
e-mail: *l.i.kuncheva@bangor.ac.uk*

Abstract

Classifier combination is now an established pattern recognition subdiscipline. Despite the strong aspiration for theoretical studies, classifier combination relies mainly on heuristic and empirical solutions. Assuming that "soft computing" encompasses neural networks, evolutionary computation, and fuzzy sets, we explain how each of the three components can be used in classifier combination.

15.1 Introduction

Let $\mathcal{D} = \{D_1, D_2, \ldots, D_L\}$ be a set of classifiers (we shall also call $\mathcal{D}$ a team or ensemble), and let $\Omega = \{\omega_1, \ldots, \omega_c\}$ be a set of class labels. Each classifier gets as its input a feature vector $\mathbf{x} = [x_1, \ldots, x_n]^T$, $\mathbf{x} \in \Re^n$ and assigns it to a class label from Ω, *i.e.*, $D_i : \Re^n \rightarrow \Omega$. Alternatively, we may define the classifier output to be a c-dimensional vector with supports to the classes, *i.e.*,

$$D_i(\mathbf{x}) = [d_{i,1}(\mathbf{x}), \ldots, d_{i,c}(\mathbf{x})]^T. \tag{15.1}$$

Without loss of generality we can restrict $d_{i,j}(\mathbf{x})$ within the interval $[0,1]$, $i = 1,\ldots,L$, $j = 1,\ldots,c$, and call the classifier outputs "soft labels" (see [7]). Thus, $d_{i,j}(\mathbf{x})$ is the degree of "support" given by classifier D_i to the hypothesis that $\mathbf{x}$ comes from class ω_j (most often $d_{i,j}(\mathbf{x})$ is an estimate of the posterior probability $P(\omega_i|\mathbf{x})$). Combining classifiers means to find a class label for $\mathbf{x}$ based on the L classifier outputs $D_1(\mathbf{x}),\ldots,D_L(\mathbf{x})$. Again, instead of a single label, we can find a vector with c final degrees of support for the classes as a soft class label for x, denoted

$$D(\mathbf{x}) = [\mu_1(\mathbf{x}),\ldots,\mu_c(\mathbf{x})]^T. \tag{15.2}$$

If a crisp class label of $\mathbf{x}$ is needed, we can use the maximum membership rule: Assign $\mathbf{x}$ to class ω_s if and only if,

$$\mu_s(\mathbf{x}) \geq \mu_t(\mathbf{x}), \forall t = 1,\ldots,c. \tag{15.3}$$

Ties are resolved arbitrarily.

We assume that a labeled data set $\mathbf{Z}$ is available, $\mathbf{Z} = \{\mathbf{z}_1,\ldots,\mathbf{z}_N\}$, $\mathbf{z}_j \in \Re^n$, which is used to design the classifier combination system: both the individual classifiers and the combiner.

Classifier combination aims at a higher accuracy than that of a single member of the team $\mathcal{D}$. In the past few years, a lot of work has been done towards developing a rigorous theoretical background of classifier combination. Yet, the useful heuristics are a step ahead the theory, and collective effort is being devoted to understanding and explaining why these heuristics work so well.

Classifier combination is called different names in the literature as shown in Table 15.1. It is therefore important to recognize the pressing need for tidying up the shelf by grouping and arranging the existing solutions in a taxonomy.

This chapter surveys soft computing methods in classifier combination. We shall assume that *soft computing* covers neural and evolutionary computation, and fuzzy sets. Section 15.2 explains classifier combination. Section 15.3 identifies the place of the three components of soft computing within classifier combination tools and techniques, and Section 15.4 offers a concluding remark.

15.2 Classifier combination

15.2.1 Four approaches

Table 15.2 shows four approaches to designing a classifier combination system.

Table 15.1 Classifier combination "aliases" in the literature

1	combination of multiple classifiers [42, 54, 61, 72, 73];
2	classifier fusion [9, 16, 25, 28, 41];
3	mixture of experts [39, 38, 40, 58];
4	committees of neural networks [8, 21];
5	consensus aggregation [6, 5, 57];
6	voting pool of classifiers [3];
7	dynamic classifier selection [72];
8	composite classifier systems [18];
9	classifier ensembles [21, 22, 62];
10	bagging, boosting, arcing, wagging [62];
11	modular systems [62];
12	collective recognition [2, 60]
13	stacked generalization [71];
14	divide-and-conquer classifiers [13];
15	pandemonium system of reflective agents [64];
16	change-glasses approach to classifier selection [45], *etc.*

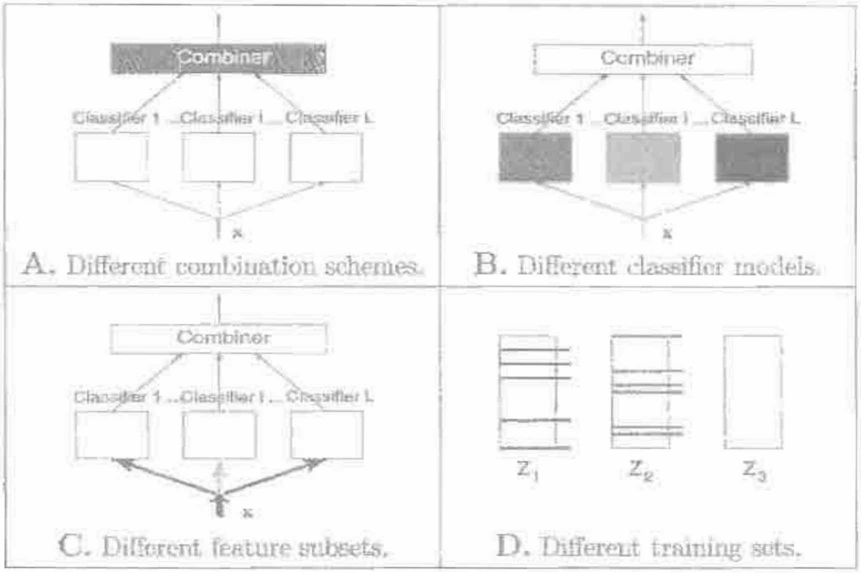
Approach A. We assume that $D_1, \ldots, D_L$ are given (trained in advance), and the problem is to pick a combination scheme and train it if necessary.

Approach B. Any pattern classifier can be used as a team member. Thus, $\mathcal{D}$ can be homogeneous, *i.e.*, formed using identical classifier models (*e.g.*, multilayer perceptron (MLP) neural networks) with different structure, parameters, initialization protocols, *etc.* Alternatively, a heterogeneous $\mathcal{D}$ can be designed, as for example in [72].

Approach C. Sometimes it is suitable to build each D_i on an individual subset of features (subspace of $\Re^n$). This is useful when n is large (*e.g.*, a few hundred), and groups of features come from different sources or different data pre-processing. Examples can be found in image and speaker recognition, etc. [12, 43].

Approach D. Many authors are of the opinion that training set alteration is the most powerful of the four approaches as it can lead to a team of *diverse* classifiers [19, 66] whereas none of the other three approaches is suited for that. Diversity among the classifiers in $\mathcal{D}$ means that the individual classifiers (D_i's) misclassify *different* objects, having at the same time high individual accuracy. This property alone can guarantee a good potential of the team even with the simplest combination schemes. Exploiting this idea, several methods have been proposed to select training subsets of the data set $\mathbf{Z}$.

Table 15.2 Four approaches to designing a classifier combination system


Combiner
Classifier 1 ...Classifier l ...Classifier L
x
A. Different combination schemes.
Combiner
Classifier 1 ...Classifier l ...Classifier L
x
B. Different classifier models.
Combiner
Classifier 1 ...Classifier l ...Classifier L
x
C. Different feature subsets.
Z1
Z2
Z3
D. Different training sets.

(1) Partition the data randomly into L parts and use a different part to train each classifier.
(2) Boosting: start with a classifier D_1 on the whole of $\mathbf{Z}$, filter out "difficult" objects and build D_2 on them. Continue with the cascade until D_L is built (e.g.,[21]).
(3) Bagging: design bootstrap samples by resampling from $\mathbf{Z}$ with a uniform distribution and train one D_i on each sample [10].
(4) Adaptive resampling: design bootstrap samples by resampling from $\mathbf{Z}$ with a non-uniform distribution. Update the distribution with respect to previous successes. Thus, more "difficult" data points will appear more often in the subsequent training samples [4, 10, 19].

Any integration of the four approaches can be applied too. For now, soft computing methods have been used in the context of approaches A, B and C.

15.2.2 Combination paradigms

There are generally two types of combinations: **classifier selection** and **classifier fusion** as named in [72]. The presumption in classifier selection is that each classifier is "an expert" in some local area of the feature space. When a feature vector $\mathbf{x} \in \Re^n$ is submitted for classification, the classifier responsible for the vicinity of $\mathbf{x}$ is given the highest credit to label $\mathbf{x}$. We can nominate exactly one classifier to make the decision, as in [60], or more than one "local expert", as in [1, 39, 67]. Classifier fusion assumes that all classifiers are trained over the whole feature space, and are thereby considered as *competitive* rather than *complementary* [57, 73].*

Fusion and selection are often merged. Instead of nominating one "expert" we can nominate a small group of them. We can then take their judgements and weight them by the level of expertise they have on $\mathbf{x}$. Thus, the classifier with the highest individual accuracy could be made the "leading expert" in the team. When many classifiers become involved, the scheme is shifted from classifier selection towards classifier fusion. This suggests that we rarely use the two strategies in their pure forms.

Two major types of multiple classifier outputs are

(1) A set of **class labels** (votes), $s_1, \ldots s_L$,

$$D_i(\mathbf{x}) = s_i \in \Omega. \tag{15.4}$$

For example, let $c = 3$, $L = 5$. The output can be

$$\boxed{\omega_3}\,\boxed{\omega_2}\,\boxed{\omega_2}\,\boxed{\omega_1}\,\boxed{\omega_2}.$$

(2) A matrix of soft labels, called the **decision profile** [52]

*In [62], classifier fusion is named *ensemble approach* and classifier selection is named *modular approach*.

Output of classifier $D_i(\mathbf{x})$

$$DP(\mathbf{x}) = \begin{bmatrix} d_{1,1}(\mathbf{x}) & \ldots & d_{1,j}(\mathbf{x}) & \ldots & d_{1,c}(\mathbf{x}) \\ \ldots & & & & \\ d_{i,1}(\mathbf{x}) & \ldots & d_{i,j}(\mathbf{x}) & \ldots & d_{i,c}(\mathbf{x}) \\ \ldots & & & & \\ d_{L,1}(\mathbf{x}) & \ldots & d_{L,j}(\mathbf{x}) & \ldots & d_{L,c}(\mathbf{x}) \end{bmatrix}. \qquad (15.5)$$

Support from classifiers $D_1 \ldots D_L$ for class ω_j

Some fusion methods calculate the support for class ω_j using only the jth column of $DP(\mathbf{x})$, regardless of what the support for the other classes is. Fusion methods that use the DP class-by-class will be called class-conscious (CC) fusion methods [52]. We refer to the alternative group as class-indifferent (CI) fusion methods, *i.e.*, methods that use the whole of the decision profile in calculating each $\mu_i(\mathbf{x})$. Notice the difference between the two groups. The former use the *context* of the DP, *i.e.*, recognizing that a column corresponds to a class, but disregard a part of the information. Class-indifferent methods use the whole DP but disregard its context.

The diagram in Fig. 15.1 depicts one possible grouping of classifier combination methods. The methods are placed in boxes at the leaves of the tree with a few corresponding references. Some of the methods will be described later while others are mentioned only for completeness. Among the class-conscious methods, the weighted linear combination is one of the most popular aggregation formulae. The support for class ω_i is calculated as the weighted average of the supports given by the L classifiers. Based on how the coefficients are obtained, we can distinguish between fixed-coefficient models and data-dependent coefficient models where the coefficients are recalculated for every input $\mathbf{x}$. It is interesting to observe that data-dependent coefficients can be so designed that the combination paradigm (starting off as a classifier fusion model) turns into a classifier selection model. For example, a linear classifier fusion model with data-dependent coefficients, so that only one coefficient is 1 and the remaining coefficients are 0s, is, in fact, selecting the classifier corresponding to the 1, to label $\mathbf{x}$.

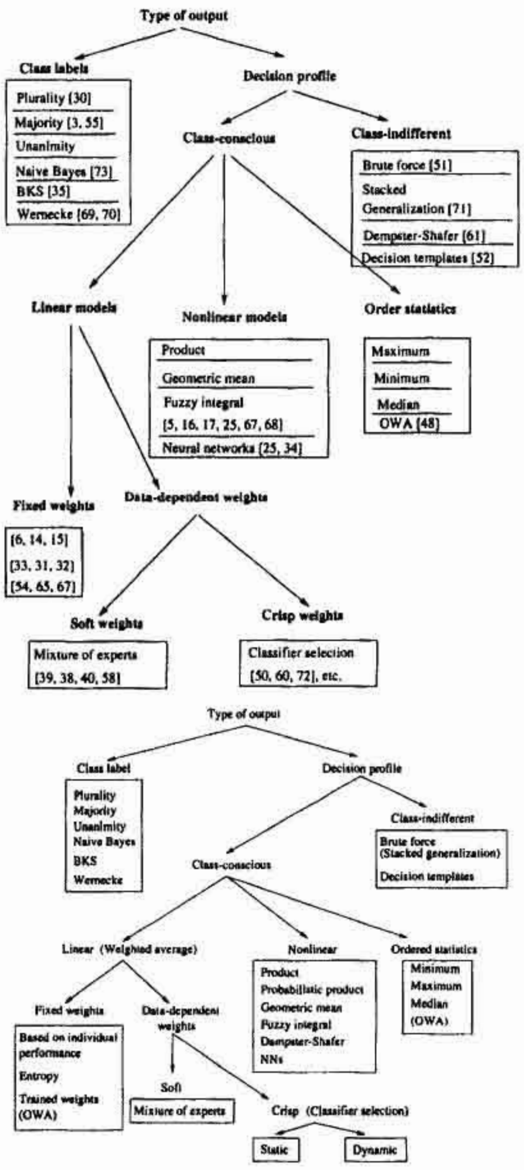


Fig. 15.1 One possible grouping of classifier combination methods

15.3 Soft computing in classifier combination

15.3.1 Neural networks

Neural networks (NNs) are the most popular choice for the individual classifiers in the team (**approach B**). Most of the studies on combining classifiers appears in the neural network literature, *e.g.*, the journals *IEEE Transactions on Neural Networks, Neural Networks, Neural Computation, Communication Science*. This choice, initially made by intuition, has now been justified theoretically. The classification error can be decomposed by algebraic manipulation into two terms: bias and variance with respect to individual classifier outputs (refer to [62]). Ideally, both terms should be small which is hardly possible for a single classifier model. Simple classifiers such as linear discriminant analysis have low variance and high bias. This means that these models are not very sensitive to small changes in the training data set (the calculation of the discriminant functions will not be much affected by small alterations in the training data) but at the same time are unable to reach low error rates. Conversely, neural networks have been shown to be ultimately versatile, *i.e.*, they can approximate any classification boundary with arbitrary precision. The price to pay for the low error rate is that neural classifiers may overtrain. Thus, neural classifiers have low bias (any classification boundary can be modeled) and high variance (small changes in the data set might lead to a very different neural network). Assume that we combine classifiers of the same bias and the same variance V by averaging the classifier outputs, *e.g.*,

$$\mu_i(\mathbf{x}) = \frac{1}{L} \sum_{k=1,L} d_{k,i}(\mathbf{x}).$$

Then the combination bias will be the same as that of the individual classifiers but the variance can be smaller than V, thereby reducing the error of the combination.

If $\mathcal{D}$ consists of *identical* classifiers, then no improvement will be gained by the combination as the variance of the team estimate will be V. If $\mathcal{D}$ consists of *statistically independent* classifiers, then the combination variance is $\frac{V}{L}$ and the error is subsequently reduced. Even better team can be constituted if the classifiers are *negatively dependent*, *i.e.*, they misclassify different objects. To be able to construct *diverse* classifiers of high accuracy, we need a versatile model. Neural networks are therefore an ideal choice for individual members of the team. The high variance should not be a concern as there are combination mechanisms that will reduce it.

Typically, MLP and radial basis function (RBF) networks are used but variants thereof are also considered [59]. Training of the individual neural classifiers may precede the design of the combination or be carried out with regard to the team performance. Some authors consider training an excessive amount of NONS and subsequently selecting the members of $\mathcal{D}$ [23, 24, 26]. If **approach D** is adopted, neural classifiers are trained on the subsequently generated training sets. Drucker [20] compares experimentally neural networks and classification trees (another classifier model found to be very suitable for classifier ensembles) and finds NONS to be superior.

Neural networks can be used as a class-indifferent (brute force or stacked generalization [71]) model and also as a class-conscious model for classifier combination (approach A) [34].

In summary, NONS are undoubtedly the most important intercept between soft computing and classifier combination.

15.3.2 Evolutionary computation

Evolutionary computation and mainly genetic algorithms (GAs) have been used at different stages of the design of classifier combination systems.

Approach A (Tuning the combiner): Genetic algorithms have been used to find a set of weights for combination through weighted sum [15, 54]. Lam and Suen [54] discuss GAs for finding L weights, one per classifier. Binary encoding of the weights is used with 10-bit representation of each weight. Similarly, Cho [15] uses a GA to find a matrix of $L \times c$ weights, one per classifier-class pair. Thus, $\mu_i(\mathbf{x})$, $i = 1, \ldots, c$ are obtained by c different linear combinations. Each weight is encoded by 8 bits. There are many publications on finding combination weights, both heuristic and more rigorous, e.g., [6, 14, 33, 31, 32, 65, 67]. Whether GAs lead to better results is unknown.

Approach B (Tuning the classifier models): All studies in this category use neural networks as individual members of the team. The neural networks are evolved by GAs with respect to both weights and structure. A standard GA, although evolving a population of networks, will converge to a *single solution*. That is, the last generation is likely to consist of exact clones or very close relatives, meaning almost identical D_is. Despite being highly accurate, these classifiers will hardly form a successful team because nothing can be gained from combining exact replicas of the same classifier. Therefore, a mechanism preserving diversity should be incorporated into the GA. One such option is *niching.*

Benediktsson *et al.* [5] apply a real-valued GA to train the network weights and a binary-coded GA for pruning weights off a trained network. The networks are evolved with respect to their individual classification performance, and the diversity of the population is enforced by special genetic operators: extinction and immigration. Friedrich [23, 24] evolves a population of neural networks and then selects a subset whose members are maximally negatively correlated. While in [5, 23, 24] a standard MLP is considered, Opitz and Shavlik [59] propose an evolutionary algorithm called ADDEMUP for knowledge-based neural networks (KBNNs). Each such network can be translated into sets of if-then rules.[†] The GA evolves a population of KBNNs to be the team $\mathcal{D}$. To maintain diversity, the fitness function of chromosome S_i (a single KBNN) is taken to be of the form

$$Fitness(S_i) = Accuracy(S_i) + \lambda\ Diversity(S_i). \tag{15.6}$$

The measure of diversity [44] is generally an estimate of the deviation of the output of the ith KBNN from the average of the team. The more diverse the ensemble, the higher the gain in classification accuracy. The parameter $\lambda > 0$ controls the balance between the two criteria. As a rule of thumb, the authors of [59] recommend to set λ to 0.1 and vary it by about 10% depending on the current accuracy-diversity dynamics. If the accuracy *of the team* is not decreasing over a number of generations but diversity decreases, then diversity is underemphasized and so λ is increased. If the accuracy starts decreasing and diversity is not decreasing, then diversity is overemphasized and so λ is decreased.

A problem with this group of methods is that the chromosomes correspond to the individual D_i's, and the fitness is not directly related to the overall classification accuracy of the team. It is possible to encode $\mathcal{D}$ as a single chromosome and evolve a population of teams. The search space, however, might become too large and the GA will demand a lot of computing resources and expert effort for tuning.

Approach C (Selecting feature subsets): One of the main uses of GAs in pattern recognition has been for selection of a subset of features. The aim is to have a space of dimensionality $t < n$, so that the classifier on $\Re^t$ is no worse than the classifier on $\Re^n$ (using all features). This problem is notoriously difficult and its optimal solution is guaranteed only if all feature subsets are checked

[†]No fuzzy systems connotation has been given by the authors.

(exhaustive enumeration). GAs are a natural option for feature selection [11, 63]. A feature selection GA for multiple classifier systems is proposed in [46]. A population of classifiers is evolved aiming at high individual accuracy. The binary chromosome S_i encodes a feature subset, and the respective classifier D_i is built using only this subset. The team $\mathcal{D}$ is then selected as the best group of L from the population. Again, the group aspiration criterion is not taken into account when the individual chromosomes (classifiers) are evaluated by their fitness. The diversity preserving adjustment in this model is that the best team is identified at each generation, and the chromosomes in it are retained for the next generation, regardless of their individual fitness. To overcome the "individualistic" approach, the whole team $\mathcal{D}$ is evolved in [53]. Two GA versions are proposed: Version 1, where D_is use disjoint subsets of features and Version 2, where the subsets of features may overlap. In Version 1, the chromosome has n genes, one for each feature. The values of each gene are in the set $\{0, 1, \ldots, L\}$. A value $i \in \{1, \ldots, L\}$ at position j means that (only) D_i uses feature x_j, and a value 0 means that feature x_j is not used by any classifier in this team.

Example 15.1 Let $n = 10$, and $L = 3$. A possible chromosome is

$$\boxed{2}\boxed{2}\boxed{1}\boxed{2}\boxed{2}\boxed{3}\boxed{2}\boxed{3}\boxed{0}\boxed{1}.$$

This chromosome represents a team $\mathcal{D}$ where D_1 uses a 2-dimensional feature vector $\mathbf{x} = [x_3, x_{10}]^T$, D_2 uses a 5-dimensional feature vector $\mathbf{x} = [x_1, x_2, x_4, x_5, x_7]^T$, D_3 uses a 2-dimensional feature vector $\mathbf{x} = [x_6, x_8]^T$, and feature x_9 is not used. □

Version 2 GA allows for 2^L values of each gene, accounting for all possible combinations of D_i (or none) that might share feature x_j.

There is an apparent analogy between the problem of evolving one member of the team and the whole team on the one hand, and the Michigan and Pittsburgh approaches for evolving fuzzy if-then systems on the other hand [51]. Within the Michigan approach, the chromosome represents one if-then rule, whereas within the Pittsburgh approach, the chromosome represents the whole fuzzy if-then system. The preferences in the literature are not clear-cut, so both approaches are used.

Approach D (Selecting training sets): The reason why this most promising approach has not been explored so far could be that if the data set $\mathbf{Z}$ is large, the same will be the chromosome, and the GA will be unacceptably slow.

Knowing the advantages of approach D, subset selection by GAs seems worth trying (see [47, 49]).

15.3.3 Fuzzy sets

Fuzzy set theory has been used predominantly at the combination stage (*Approach A*). Detailed below are several fuzzy combination schemes (*cf.* [51]).

15.3.3.1 *Simple fuzzy aggregation connectives*

These combination designs belong to the *class-conscious* group because each $\mu_i(\mathbf{x})$ is calculated using only the ith column of the decision profile $DP(\mathbf{x})$. We use the L-place operators *minimum, maximum, average* and *product* as the function $\mathcal{F}$ in

$$\mu_i(\mathbf{x}) = \mathcal{F}\ (d_{1,i}(\mathbf{x}), \ldots, d_{L,i}(\mathbf{x}))\,,\ \ i = 1, \ldots, c. \tag{15.7}$$

Example 15.2 Let $c = 3$ and $L = 5$. Assume that for a certain $\mathbf{x}$,

$$DP(\mathbf{x}) = \begin{bmatrix} 0.1\ 0.5\ 0.4 \\ 0.0\ 0.0\ 1.0 \\ 0.4\ 0.3\ 0.4 \\ 0.2\ 0.7\ 0.1 \\ 0.1\ 0.8\ 0.2 \end{bmatrix}.$$

Applying each of the operators columnwise, we obtain as the final soft class labels

$$\begin{aligned} \text{Minimum} &= [\,0.0,\ 0.0,\ 0.1\,]^T; \\ \text{Maximum} &= [\,0.4,\ 0.8,\ 1.0\,]^T; \\ \text{Average} &= [\,0.16,\ 0.46,\ 0.42\,]^T; \\ \text{Product} &= [\,0.0,\ 0.0,\ 0.0032\,]^T. \end{aligned}$$

If hardened, minimum, maximum, and product will label $\mathbf{x}$ in class ω_3, whereas the average will put $\mathbf{x}$ in class ω_2. □

15.3.3.2 *More sophisticated aggregation connectives*

Many such aggregation operations are available in the fuzzy set literature [9]. Ordered Weighted Averaging (OWA) operators can also be applied as $\mathcal{F}$ [48]. The OWA coefficients are not associated with a particular classifier D_i but with the *places* in the ordered outputs. The operation of OWA combination is shown in Fig. 15.2

OWA operators for combining classifiers,

(1) Pick L OWA coefficients such that

$$\mathbf{b} = [b_1, \ldots, b_L]^T, \quad \sum_{i=1}^{L} b_i = 1.$$

(2) For $k = 1, \ldots, c$,

(a) Sort $d_{i,k}(\mathbf{x}), i = 1, \ldots, L$ in descending order, so that

$$a_1 = \max_i d_{i,k}(\mathbf{x}), \quad \text{and} \quad a_L = \min_i d_{i,k}(\mathbf{x}).$$

(b) Calculate the support for class ω_k

$$\mu_k(\mathbf{x}) = \sum_{i=1}^{L} b_i a_i.$$

Fig. 15.2 OWA operators for combining classifiers

OWA prevents crediting one particular "expert" with the highest competence across $\Re^n$, as it would be the case if we assigned fixed weights to the classifiers. If the favorite expert (classifier) has received the credit because of overfitting the training data, then by praising it, we can face poor generalization. Thus, classifier fusion by OWA seems more robust than the weighted average, where the coefficients are derived on the basis of classifier performance. It is worth noticing that the fuzzy integral for classifier fusion takes this idea further so that OWA aggregation is a special case of it. OWA can model various operations as shown in Table 15.3. We can either pick the set of OWA coefficients or calculate them from $\mathbf{Z}$ by minimizing the classification error of $\mathcal{D}$.

Verikas *et al.* [67] consider aggregation by Zimmermann and Zysno's compensatory operator

$$\mu_i(\mathbf{x}) = \left(\prod_{k=1}^{L} [d_{k,i}(\mathbf{x})]^{w_k} \right)^{1-\gamma} \left(1 - \prod_{k=1}^{L} [1 - d_{k,i}(\mathbf{x})]^{w_k} \right)^{\gamma}, \qquad (15.8)$$

Table 15.3 Special cases of OWA operators

Minimum	$[0,0,\ldots,1]^T$,
Maximum	$[1,0,\ldots,0]^T$,
Median	$[\underbrace{0,\ldots 0}_{\frac{L-1}{2}},1,\underbrace{0,\ldots 0}_{\frac{L-1}{2}}]^T$, for odd L, $[\underbrace{0,\ldots 0}_{\frac{L-2}{2}},\frac{1}{2},\frac{1}{2},\underbrace{0,\ldots 0}_{\frac{L-2}{2}}]^T$, for even L,
Average	$[\frac{1}{L},\ldots,\frac{1}{L}]^T$,
Competition jury	$[0,\frac{1}{L-2},\ldots,\frac{1}{L-2},0]^T$.

where $w_k, k = 1,\ldots,L$ are coefficients of global "competence" (across the whole $\Re^n$), $\sum_{k=1}^{L} w_k = L$, and $\gamma \in [0,1]$ is the compensation parameter. Verikas *et al.* [67] propose also aggregation by BADD defuzzification[‡]

$$\mu_i(\mathbf{x}) = \frac{\sum_{k=1}^{L} d_{k,i}(\mathbf{x})[w_k(\mathbf{x})]^\delta}{\sum_{k=1}^{L}[w_k(\mathbf{x})]^\delta}, \quad i = 1,\ldots,c, \tag{15.9}$$

where δ is a parameter, and $w_k(\mathbf{x})$ are data-dependent weights calculated to express the "expertise" of classifier D_k for the input $\mathbf{x}$.

15.3.3.3 *Fuzzy integral*

Fuzzy integral can also be used as an aggregation connective [27, 29] and has been applied to classifier combination [5, 16, 17, 25, 68, 67].

We use a fuzzy measure to take into account the *importance of any subset* of classifiers from $\mathcal{D}$ with respect to a given ω_i. Let $\mathcal{P}(\mathcal{D})$ be the power set of

[‡]although used in a slightly different context

$\mathcal{D}$. A *fuzzy measure* on $\mathcal{D}$ is the set function

$$g : \mathcal{P}(\mathcal{D}) \rightarrow [0, 1], \tag{15.10}$$

such that

(1) $g(\emptyset) = 0, \;\; g(\mathcal{D}) = 1$;
(2) For any A and B, subsets of $\mathcal{D}$, $A \subset B \Rightarrow g(A) \leq g(B)$.

g is called a λ-fuzzy measure if for any A and B, subsets of $\mathcal{D}$, such that $A \cap B = \emptyset$,

$$g(A \cup B) = g(A) + g(B) + \lambda g(A)g(B), \quad \lambda \in (-1, \infty). \tag{15.11}$$

Two basic types of fuzzy integrals have been proposed: Sugeno type and Choquet type. Let H be a fuzzy set on $\mathcal{D}$. The Sugeno fuzzy integral with respect to a fuzzy measure g is obtained by

$$\mathcal{A}_g^{FI} = \max_{\alpha}\{\min(\alpha, g(H_\alpha))\}, \tag{15.12}$$

where H_α is the α-cut of H.

Example 15.3 Let $L = 3$, and let the fuzzy measure g be defined as follows:

Subset	D_1	D_2	D_3	D_1, D_2	D_1, D_3	D_2, D_3	D_1, D_2, D_3
g	0.3	0.1	0.4	0.4	0.5	0.8	1

Let $H = [0.1, 0.7, 0.5]^T$ be a fuzzy set on $\mathcal{D}$ accounting for the support for class ω_i by D_1, D_2, and D_3, respectively (the ith row of $DP(\mathbf{x})$). The α-cuts of H are

$$\begin{aligned}
\alpha = 0, \quad H_0 \;\; &= \{D_1, D_2, D_3\};\\
\alpha = 0.1, \; H_{0.1} &= \{D_1, D_2, D_3\};\\
\alpha = 0.5, \; H_{0.5} &= \{D_2, D_3\};\\
\alpha = 0.7, \; H_{0.7} &= \{D_2\};\\
\alpha = 1, \quad H_0 \;\; &= \emptyset.
\end{aligned}$$

Then

$$\begin{aligned}\mu_i(\mathbf{x}) &= \mathcal{A}_g^{FI} \\ &= \max\{\min(0,1), \min(0.1,1), \min(0.5,0.8), \\ &\quad \min(0.7,0.1), \min(1,0)\} \\ &= \max\{0, 0.1, 0.5, 0.1, 0\} \\ &= 0.5.\end{aligned} \tag{15.13}$$

□

The fuzzy measure g can be calculated from a set of L values g^j, called fuzzy densities, representing the individual importance of $D_1, \ldots, D_L$, respectively. We can find a λ-fuzzy measure which is consistent with these densities. The value of λ is obtained as the unique real root greater than -1 of the polynomial

$$\lambda + 1 = \prod_{j=1}^{L}(1 + \lambda g^j), \quad \lambda \neq 0. \tag{15.14}$$

The operation of fuzzy integral as a classifier combiner is shown in Fig. 15.3.

The support for ω_i, $\mu_i(\mathbf{x})$, can be thought of as a "compromise" between the competence (represented by the fuzzy measure g) and the evidence (represented by the i-th row of the decision profile $DP(\mathbf{x})$. Notice that the fuzzy measure vector $[g(1), \ldots, g(L)]^T$ might be different for each class, and is also specific for the current $\mathbf{x}$. Two fuzzy measure vectors will be the same only if the ordering of the classifier support is the same. The algorithm in Fig. 15.3 calculates a Sugeno fuzzy integral. For the Choquet fuzzy integral with the same λ-fuzzy measure, the last formula should be replaced by

$$\mu_k(\mathbf{x}) = d_{i_1,k}(\mathbf{x}) + \sum_{j=2}^{L}\left(d_{i_{j-1},k}(\mathbf{x}) - d_{i_j,k}(\mathbf{x})\right) g(j-1).$$

15.3.3.4 *Decision templates*

The idea of the decision template model is to "remember" the most typical decision profile for each class, called the *decision template*, DT_i, for that class, and then compare it with the current decision profile $DP(\mathbf{x})$. The closest match will label $\mathbf{x}$. Fig. 15.4 describes the training and Fig. 15.5, the operation of the decision templates model. The similarity between DT_i, $i = 1, \ldots, c$ on the one

Fuzzy integral for classifier fusion

(1) Fix the L *fuzzy densities* $g^1, \ldots, g^L$, e.g., by setting g^j to the estimated probability of correct classification of D_j.

(2) Calculate $\lambda > -1$ from (15.14).

(3) For a given $\mathbf{x}$ sort the kth column of $DP(\mathbf{x})$ to obtain $[d_{i_1,k}(\mathbf{x}), d_{i_2,k}(\mathbf{x}), \ldots, d_{i_L,k}(\mathbf{x})]^T$, $d_{i_1,k}(\mathbf{x})$ being the highest degree of support, and $d_{i_L,k}(\mathbf{x})$, the lowest.

(4) Sort the fuzzy densities correspondingly, *i.e.*, $g^{i_1}, \ldots, g^{i_L}$ and set $g(1) = g^{i_1}$.

(5) For $t = 2$ to L, calculate recursively

$$g(t) = g^{i_t} + g(t-1) + \lambda g^{i_t} g(t-1).$$

(6) Calculate the final degree of support for class ω_k by

$$\mu_k(\mathbf{x}) = \max_{t=1}^{L} \left\{ \min\{d_{i_t,k}(\mathbf{x}), g(t)\} \right\}.$$

Fig. 15.3 Fuzzy integral for classifier fusion

hand, and $DP(\mathbf{x})$ on the other hand, is calculated through Euclidean distance between the two.

Decision templates (training)

(1) For $i = 1, \ldots, c$, calculate the mean of the decision profiles $DP(\mathbf{z}_j)$ of all members of ω_i from the data set $\mathbf{Z}$. Call the mean *a decision template* DT_i

$$DT_i = \frac{1}{N_i} \sum_{\substack{\mathbf{z}_j \in \omega_i \\ \mathbf{z}_j \in \mathbf{Z}}} DP(\mathbf{z}_j), \qquad (15.15)$$

where N_i is the number of elements of $\mathbf{Z}$ from ω_i.

(2) Return $DT_1, \ldots, DT_c$.

Fig. 15.4 Training of the decision templates method

Decision templates (operation)

(1) Given the input $\mathbf{x} \in \Re^n$, construct $DP(\mathbf{x})$ as in (15.5).

(2) Calculate the squared Euclidean distance between $DP(\mathbf{x})$ and each DT_i, $i = 1, \ldots, c$

$$d_E(DP(\mathbf{x}), DT_i) = \sum_{j=1}^{c} \sum_{k=1}^{L} (d_{k,j}(\mathbf{x}) - dt_i(k,j))^2, \qquad (15.16)$$

where $dt_i(k,j)$ is the k,j-th entry in decision template DT_i (an $L \times c$ matrix).

(3) Calculate the components of the soft label of $\mathbf{x}$ by

$$\mu_i(\mathbf{x}) = 1 - \frac{1}{L \cdot c} d_E(DP(\mathbf{x}), DT_i). \qquad (15.17)$$

Fig. 15.5 Operation of the decision templates method

Example 15.4 Let $c = 3$ and $L = 2$, and

$$DT_1 = \begin{bmatrix} 0.6 & 0.4 \\ 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \quad \text{and} \quad DT_2 = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \\ 0.1 & 0.9 \end{bmatrix}. \qquad (15.18)$$

Assume that for an input $\mathbf{x}$, the following decision profile has been obtained:

$$DP(\mathbf{x}) = \begin{bmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}. \qquad (15.19)$$

Then the soft label of $\mathbf{x}$ is

$$\mu_1(\mathbf{x}) = 0.96, \quad \mu_2(\mathbf{x}) = 0.93. \qquad (15.20)$$

□

As both $DP(\mathbf{x})$ and DT_i are fuzzy sets on $\mathcal{D} \times \Omega$, any measure of similarity between fuzzy sets can be used.

Ishibuchi *et al.* [36, 37] propose voting schemes over a set of fuzzy if-then rules or systems of fuzzy if-then rules. Lu and Yamaoka [56] apply fuzzy inference to design the combiner.

Fuzzy set theory offers a great choice of combination ideas but these have not been explored in conjunction with approach $\mathbf{D}$. Many authors argue that the combination scheme is not as relevant for the final accuracy as is the diversity of the classifiers. Knowing that fuzzy combinations are capable of improving the accuracy beyond that of the majority vote or the averaging model, integration of fuzzy combination with approach $\mathbf{D}$ seems very promising.

15.4 Conclusions

This chapter explains classifier combination and some soft computing paradigms within it. The three components attributed to the term "soft computing": neural networks, evolutionary computing and fuzzy sets are considered separately. The purpose was to explain the techniques and methods used, not to advocate a particular example. Brief comments on each soft computing component are offered in the respective subsections. It goes without saying that all soft computing methods cited in this study have been compared experimentally against some rival methods and have been found to be better. However, since there is no accepted standard or "ultimate test", the superiority of some techniques over others cannot be empirically proven. This is the beauty and the curse of the heuristic methods such as these explained here, and the final choice is left to the experience and the intuition of the designer. Sorting and grouping the methods, as done here, might help with the choice, and so might the somewhat contradictory guideline: keep it *simple* and *accurate*.

References

[1] E. Alpaydin and M. I. Jordan. Local linear perceptrons for classification. *IEEE Transactions on Neural Networks*, 7:788–792, 1996.

[2] Y. L. Barabash. *Collective Statistical Decisions in Recognition*. Radio i Sviaz', Moscow, 1983. (In Russian).

[3] R. Battiti and A.M. Colla. Democracy in neural nets: voting schemes

for classification. *Neural Networks*, 7:691–707, 1994.

[4] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–142, 1999.

[5] J.A. Benediktsson, J.R. Sveinsson, J. I. Ingimundarson, H. Sigurdsson, and O.K. Ersoy. Multistage classifiers optimized by neural networks and genetic algorithms. *Nonlinear Analysis, Theory, Methods & Applications*, 30:1323–1334, 1997.

[6] J.A. Benediktsson and P.H. Swain. Consensus theoretic classification methods. *IEEE Transactions on Systems, Man and Cybernetics*, 22:688–704, 1992.

[7] J.C. Bezdek, J.M Keller, R. Krishnapuram, and N.R. Pal. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Kluwer Academic Publishers, Boston, 1999.

[8] C.M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[9] I. Bloch. Information combination operators for data fusion: a comparative review with classification. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, 26:52–67, 1996.

[10] L. Brieman. Combining predictors. In A.J.C. Sharkey, editor, *Combining Artificial Neural Nets*, pages 31–50. Springer-Verlag, London, 1999.

[11] E.I. Chang and R.P. Lippmann. Using genetic algorithms to improve pattern classification performance. volume 3 of *Neural Information Processing Systems*, pages 797–803, San Mateo, CA, 1991. Morgan Kaufmann, San Mateo, CA.

[12] K. Chen, L. Wang, and H. Chi. Methods of combining multiple classifiers with different features and their applications to text-independent speaker identification. *International Journal of Pattern Recognition and Artificial Intelligence*, 11:417–445, 1997.

[13] C.-C. Chiang and H.-C. Fu. A divide-and-conquer methodology for modular supervised neural network design. In *IEEE International Conference on Neural Networks*, pages 119–124, Orlando, Florida, 1994.

[14] C.C. Chibelushi, F. Deravi, and J.S.D. Mason. Adaptive classifier integration for robust pattern recognition. *IEEE Transactions on Systems, Man and Cybernetics*, Part B: Cybernetics, 29:902–907, 1999.

[15] S.-B. Cho. Pattern recognition with neural networks combined by genetic algorithm. *Fuzzy Sets and Systems*, 103:339–347, 1999.

[16] S.-B. Cho and J.H. Kim. Combining multiple neural networks by fuzzy integral and robust classification. *IEEE Transactions on Systems, Man and Cybernetics*, 25:380–384, 1995.

[17] S.B. Cho and J.H. Kim. Multiple network fusion using fuzzy logic. *IEEE Transactions on Neural Networks*, 6:497–501, 1995.

[18] B.V. Dasarathy and B.V. Sheela. A composite classifier system design: concepts and methodology. *Proceedings of IEEE*, 67:708–713, 1978.

[19] T.G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, Cagliari, Italy, 2000.

[20] H. Drucker. Boosting using neural networks. In A.J.C. Sharkey, editor, *Combining Artificial Neural Nets*, pages 51–78. Springer-Verlag, London, 1999.

[21] H. Drucker, C. Cortes, L.D. Jackel, Y. LeCun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6:1289–1301, 1994.

[22] E. Filippi, M. Costa, and E. Pasero. Multy-layer perceptron ensembles for increased performance and fault-tolerance in pattern recognition tasks. In *IEEE International Conference on Neural Networks*, pages 2901–2906, Orlando, Florida, 1994.

[23] C.M. Friedrich. Ensembles of evolutionary created artificial neural networks. In *Proc. 5th Int. Workshop Fuzzy-Neuro Systems'98 (FNS'98)*, pages 250–256, Munich, Germany, 1998.

[24] C.M. Friedrich. Ensembles of evolutionary created artificial neural networks and nearest neighbour classifiers. In *Proc. 3rd On-line Conference on Soft Computing in Engineering Design and Manufacturing (WSC3)*, pages 288–298, 1998.

[25] P.D. Gader, M.A. Mohamed, and J.M. Keller. Fusion of handwritten word classifiers. *Pattern Recognition Letters*, 17:577–584, 1996.

[26] G. Giacinto and F. Roli. Design of effective neural network ensembles for image classification processes. *Image Vision and Computing Journal*, 2000. (to appear).

[27] M. Grabisch. On equivalence classes of fuzzy connectives - the case of fuzzy integrals. *IEEE Transactions on Fuzzy Systems*, 3:96–109, 1995.

[28] M. Grabisch and F. Dispot. A comparison of some for fuzzy classification on real data. In *2nd International Conference on Fuzzy Logic and Neural Networks*, pages 659–662, Iizuka, Japan, 1992.

[29] M. Grabisch and M. Sugeno. Multi-attribute classification using fuzzy

integral. In *IEEE International Conference on Fuzzy Systems*, pages 47–54, San Diego, California, 1992.

[30] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001, 1990.

[31] S. Hashem. Optimal linear combinations of neural networks. *Neural Networks*, 10:599–614, 1997.

[32] S. Hashem. Treating harmful collinearity in neural network ensembles. In A.J.C. Sharkey, editor, *Combining Artificial Neural Nets*, pages 101–125. Springer-Verlag, London, 1999.

[33] S. Hashem, B. Schmeiser, and Y. Yih. Optimal linear combinations of neural networks: an overview. In *IEEE International Conference on Neural Networks*, pages 1507–1512, Orlando, Florida, 1994.

[34] Y.S. Huang and C.Y. Suen. A method of combining multiple classifiers - a neural network approach. In *12th International Conference on Pattern Recognition*, pages 473–475, Jerusalem, Israel, 1994.

[35] Y.S. Huang and C.Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:90–93, 1995.

[36] H. Ishibuchi, T. Morisawa, and T. Nakashima. Voting schemes for fuzzy rule-based classification systems. In *Proceedings of FUZZ/IEEE*, 1996.

[37] H. Ishibuchi, T. Nakashima, and T. Morisawa. Voting in fuzzy rule-based systems for pattern classification problems. *Fuzzy Sets and Systems*, 103:223–238, 1999.

[38] R.A. Jacobs. Methods for combining experts' probability assessments. *Neural Computation*, 7:867–888, 1995.

[39] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[40] M.I. Jordan and L. Xu. Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, 8:1409–1431, 1995.

[41] J.M. Keller, P. Gader, H. Tahani, J.-H. Chiang, and M. Mohamed. Advances in fuzzy integration for pattern recognition. *Fuzzy Sets and Systems*, 65:273–283, 1994.

[42] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,

20:226–239, 1998.

[43] J. Kittler, A. Hojjatoleslami, and T. Windeatt. Strategies for combining classifiers employing shared and distinct representations. *Pattern Recognition Letters*, 18:1373–1377, 1997.

[44] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation and active learning. In G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. MIT Press, Cambridge, MA, 1995.

[45] L.I. Kuncheva. Change-glasses approach in pattern recognition. *Pattern Recognition Letters*, 14:619–623, 1993.

[46] L.I. Kuncheva. Genetic algorithm for feature selection for parallel classifiers. *Information Processing Letters*, 46:163–168, 1993.

[47] L.I. Kuncheva. Editing for the k-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16:809–814, 1995.

[48] L.I. Kuncheva. An application of OWA operators to the aggregation of multiple classification decisions. In R. R. Yager and J. Kacprzyk, editors, *The Ordered Weighted Averaging operators. Theory and Applications*, pages 330–343. Kluwer Academic Publishers, Boston, 1997.

[49] L.I. Kuncheva. Fitness functions in editing k-NN reference set by genetic algorithms. *Pattern Recognition*, 30:1041–1049, 1997.

[50] L.I. Kuncheva. Clustering-and-selection model for classifier combination. In *Proc. Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, Brighton, UK, 2000.

[51] L.I. Kuncheva. *Fuzzy Classifier Design*. Studies in Fuzziness and Soft Computing. Springer-Verlag, Heidelberg, 2000.

[52] L.I. Kuncheva, J.C. Bezdek, and R.P.W. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 1999. (accepted).

[53] L.I Kuncheva and L.C. Jain. Designing classifier fusion systems by genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 2000. (accepted).

[54] L. Lam and C.Y. Suen. Optimal combination of pattern classifiers. *Pattern Recognition Letters*, 16:945–954, 1995.

[55] L. Lam and C.Y. Suen. Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on Systems, Man and Cybernetics*, 27:553–568, 1997.

[56] Y. Lu and F. Yamaoka. Fuzzy integration of classification results. *Pattern Recognition*, 30:1877–1891, 1997.

[57] K.-C. Ng and B. Abramson. Consensus diagnosis: A simulation study. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:916–928, 1992.

[58] S.J. Nowlan and G.E. Hinton. Evaluation of adaptive mixtures of competing experts. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems Volume 3*, pages 774–780, 1991.

[59] D. Opitz and J. Shavlik. A genetic algorithm approach for creating neural network ensembles. In A.J.C. Sharkey, editor, *Combining Artificial Neural Nets*, pages 79–99. Springer-Verlag, London, 1999.

[60] L.A. Rastrigin and R.H. Erenstein. *Method of Collective Recognition*. Energoizdat, Moscow, 1981. (In Russian).

[61] G. Rogova. Combining the results of several neural network classifiers. *Neural Networks*, 7:777–781, 1994.

[62] A.J.C. Sharkey, editor. *Combining Artificial Neural Nets. Ensemble and Modular Multi-Net Systems*. Springer-Verlag, London, 1999.

[63] W. Siedlecki and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10:335–347, 1989.

[64] F. Smieja. The pandemonium system of reflective agents. *IEEE Transactions on Neural Networks*, 7:97–106, 1996.

[65] V. Tresp and M. Taniguchi. Combining estimators using non-constant weighting functions. In G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems Volume 7*, Cambridge, MA, 1995. MIT Press.

[66] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8:385–404, 1996.

[67] A. Verikas, A. Lipnickas, K. Malmqvist, M. Bacauskiene, and A. Gelzinis. Soft combination of neural classifiers: A comparative study. *Pattern Recognition Letters*, 20:429–444, 1999.

[68] D. Wang, J. M. Keller, C.A. Carson, K.K. McAdoo-Edwards, and C.W. Bailey. Use of fuzzy-logic-inspired features to improve bacterial recognition through classifier fusion. *IEEE Transactions on Systems, Man and Cybernetics*, 28B:583–591, 1998.

[69] K.-D. Wernecke. On classification strategies in medical diagnostics (with special preference to mixed models). In H.H. Bock, editor, *Classification and Related Methods of Data Analysis*, pages 299–306. Elsevier Science, Amsterdam, 1988.

[70] K.-D. Wernecke. A coupling procedure for discrimination of mixed data. *Biometrics*, 48:497–506, 1992.

[71] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–260, 1992.

[72] K. Woods, W.P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:405–410, 1997.

[73] L. Xu, A. Krzyzak, and C.Y. Suen. Methods of combining multiple classifiers and their application to handwriting recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 22:418–435, 1992.

Chapter 16

AUTOMATED GENERATION OF QUALITATIVE REPRESENTATIONS OF COMPLEX OBJECTS BY HYBRID SOFT-COMPUTING METHODS

E. H. Ruspini* and I. S. Zwir†

**Artificial Intelligence Center*
SRI International
Menlo Park, California, U.S.A.
e-mail: *ruspini@ai.sri.com*

†*Department of Informatics*
School of Science
University of Buenos Aires
Buenos Aires, ARGENTINA
e-mail: *zwir@dc.uba.ar*

Abstract

A soft-computing based approach for the generation of representations of complex objects by identification of significant qualitative features and by automatic recognition of significant relationships between those features, is described. Fuzzy clustering techniques that explore object descriptions or large data samples seeking to find structures that match approximately certain prototypical or paradigmatic structures are presented for this purpose. Posing the related clustering problem as a large-scale optimization problem, evolutionary computation techniques are employed to isolate *extensive* structures. A local Pareto-optimal methodology intended to discover all structures that are non-dominated in a Pareto-optimal sense, is used in this context. The output of this process is then organized and related in terms of approximate relations. These relations are, once again, identified by their proximity, according to a user-defined similarity, to certain paradigmatic, or "interesting" relations, also provided by the user. The methodology is illustrated through examples of its application to economic time-series analysis.

16.1 Introduction

The increased availability of databases containing representations of complex objects such as time series, biological molecules, metabolic pathways, or organizational plans, permits access to vast amounts of data about the real-world systems where these objects may be found, observed, or developed. The underlying object representations employed in these databases, however, are typically based on computational convenience considerations. Often, these representations are also closely linked to the measurement techniques and devices employed to observe those objects. Users of these repositories, on the other hand, are concerned with structural and functional properties of the represented objects and systems that shed light on the nature and behavior of the underlying systems. While such properties are related to their computational representation, they are not readily derivable from them. In fact, in many situations, the computational representation methods hinder rather than facilitate understanding of either individual objects or of the full database as an ensemble of related structures. This lack of a clear connection between data representations developed for computational convenience and those characteristics that are deemed to be interesting by the intended users of the data repositories limits their accessibility and usefulness.

A good example of this situation is given by representations of complex biological molecules as arrays of atoms positions and characteristics, which do not readily permit the visualization of important characteristics such as surface features or structural patterns. This approach to the description of biomolecules, while promoting computational efficiency and representational accuracy, hinders search and retrieval in terms of structural and functional properties deemed to be important by molecular biologists. Another example is provided by representation of economical or financial time series as long sequences of real numbers, which conceals rather than reveals important temporal features such as trends or non-stationary patterns.

This undesirable situation may also be traced to an undesirable tendency to overemphasize accuracy and precision at the expense of understandability. This emphasis on computational efficiency and representational accuracy leads to the paradoxical situation where the measurement-based data models thus provided obfuscate comprehension of the systems being modeled. This situation is one more instance—involving in this case data-based models—of the undesirable practice of attempting to understand complex systems through models having a comparable level of complexity. Under these circumstances, those models, however precise and accurate, detract from the very purpose, leading, in the first place, to the initial measurement of variables related to the object structure and characteristics.

To use a term recently employed by Zadeh [16] to discuss the causes of this type of problem, the data repositories are concerned with measurements rather than with "perceptions." The emphasis on representation of model parameters linked, more or less directly, to measurements also results in search tools and structures (*e.g.*, indexes) that do not permit retrieval of contents according to criteria that match the needs of their users. The accessibility and usefulness of these databases might be, however, considerably increased by the development of automated tools for the description of objects in terms of structures of interest to users (*e.g.*, surface "pockets" having certain characteristics in biological molecules) and by the concomitant indexing of digital libraries on the basis of those structures.

In this work, we present results of research concerned with the identification of qualitative or "approximate" structures in data representations of complex data objects. Our primary goal is to uncover portions of those objects that approximately match one of a set of models that have been pre-specified by experts as being of particular interest to them. In a time-series database, these models might include structures such as *uptrends*, *downtrends*, and various oscillatory patterns. These models of interesting structures are qualitative in the sense that they measure the degree (in a [0,1] scale) by which a portion of the object being described (*i.e.*, a set of time-series values corresponding to an interval) matches an instantiation (*i.e.*, specific parameterization) of the approximate model. These studies are part of a larger program of research aimed also at the description of approximate interesting relations between interesting structures (*e.g.*, "a long downtrend preceded by a few months a sharp uptrend"), producing hypertext representations and indexes based on those representations, and, ultimately, processing the full database of qualitative representations seeking to uncover important knowledge about the systems being represented (*i.e.*, "data mining").

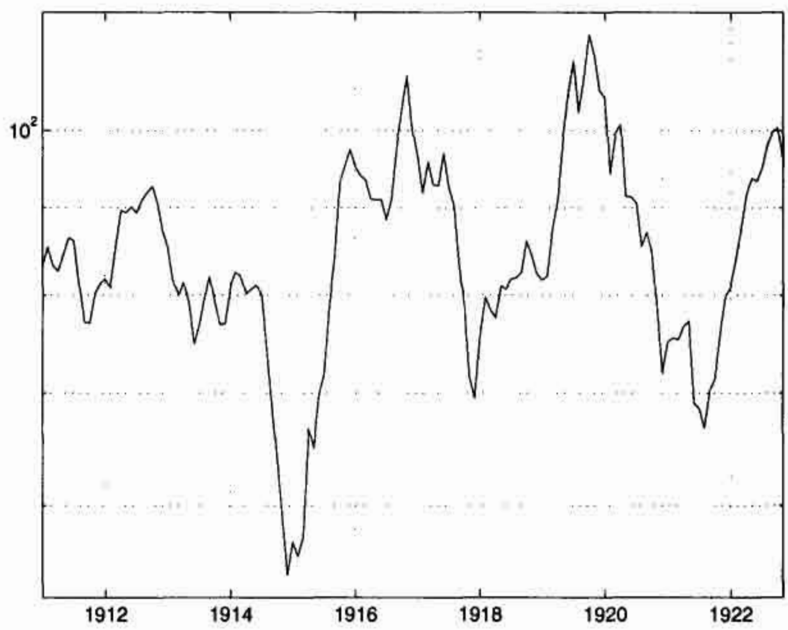


Fig. 16.1 Monthly averages of closing values of the Dow-Jones Industrial Average index (DJIA) between 1912 and 1922

16.2 Problem

We are concerned with the problem of discovering interesting qualitative structures in complex data objects and the associated problem of determining interesting relations between them. It is assumed that the notion of interestingness, which is problem-dependent, is formally defined by means of a family of parameterized models $\mathcal{M} = \{M_\alpha\}$ and by a set of relations between them that are provided beforehand by domain experts.

The models contained in the collection $\mathcal{M}$ are approximate or qualitative in the sense that they measure the *degree of matching* between substructures—corresponding to a subset of the dataset representing the object—and prototypical instances of the interesting feature. In a time-series, for example, uptrends are usually defined by a function that measures, on a [0,1] scale, the degree by which series values in a subinterval of the temporal span of the series match a prespecified definition of uptrend. Typically, the degree of matching is defined by means of a measure $Q(F, M)$ assigning to each instantiated model M and to every subset F of the dataset, a value between 0 and 1. The functionals Q defining the degree of matching are typically derived from expert knowledge about the underlying real-world systems.

Similarly, interesting relations, such as *follows* or *closely follows*, are modeled by approximate-relation models, which are [0,1]-functions of pairs of structures defining the degree by which those structures are in the desired relationship.

Our preferred approach to the characterization of qualitative features relies on application of fuzzy-logic concepts and techniques to define degree of matching by means of logical expressions containing fuzzy predicates. These expressions, essentially the conjunction of predicates that any candidate structure F satisfy to some degree, readily lead to the required measure of explanatory quality (*i.e.*, to a measure of the extent by which the structure F meets the elastic constraints of a fuzzy model M). Alternative characterizations rely on extensions to the fuzzy domain of statistical measures, such as least-square approximation error, that measure the degree by which a dataset satisfies a hypothesis. Measures of quality of matching may also be obtained applying learning techniques, such as neural-network methods, to a training set of typical examples of the structure being modeled.

Discovery of interesting features, in our treatment, is formulated as a generalized clustering problem where the clusters correspond to subsets of data that match, to acceptable degrees, the definitions provided by the model collection $\mathcal{M}$. This form of clustering, however, differs from the customary interpretation of this family of unsupervised pattern-recognition procedures in a number of respects.

In first place, our treatment is not based on a common interpretation of clusters as subsets of the dataset where each pair of points in the subset is close or "similar" (on the basis of pre-specified notions of distance or similarity). We seek, more generally, cohesive subsets in the sense that the subset, as a whole, exhibits certain relationships and meets certain constraints specified in the collection of interesting models $\mathcal{M}$. For example, an uptrend is not simply a subset of similar (time, value) pairs but, rather, a collection of such tuples satisfying the constraints specified by the uptrend model.

Our methodology, following the original ideas of Ruspini [13], is based on the formulation of the clustering problem as a continuous-variable optimization problem defined over the space of fuzzy partitions of a dataset. In this early formulation, the emphasis was placed on the determination of the *clustering* as a whole, *i.e.*, of an exhaustive, disjoint, fuzzy partition of the data set into interesting structures. The notion of interesting structure was based upon an existing notion of similarity or resemblance, that is, subsets were interesting (or were "clusters") if their members were all similar to each other and dissimilar from points in other clusters.

This view of the clustering problem as the determination of optimal fuzzy partitions of a data set has conceptual as well as methodological advances. At the conceptual level, the space of fuzzy partitions—being richer in descriptive power than the more restricted set of conventional partitions—permits a better characterization of the relations between points and clusters [14] while providing also a better correspondence between the metric structure defined in the data set (*i.e.*, the given similarity structure) and related notions of resemblance defined in the classification space. From a methodological viewpoint, the clustering problem, previously thought of as the domain of discrete mathematics, may now be treated by a number of optimization techniques based on continuous-variable analysis. Extensions of this idea are the bases for numerous generalized clustering methods [5].

Our approach to the generation of qualitative descriptions of complex objects is based on generalizations of this idea. Recognizing first that not every structure in the object might be interesting, we do not require that the clustering partition be exhaustive. This generalization permits us to shift the attention from the determination of an optimizing partition or *clustering* to that of the solution of a family of related optimization problems concerned with the sequential isolation of individual clusters. (Our optimization process is actually based on the determination of multiple, locally-optimal, solutions of functionals defining individual cluster quality.) In this regard, our approach is conceptually similar to the classification methods known as *possibilistic clustering* [10]. This reformulation of the problem also provides the ability to determine overlapping clusters while removing the requirement for prior knowledge of their number.

Our methods are also noteworthy in that they rely on a variety of definitions of cluster (provided by the family $\mathcal{M}$ of interesting features). A key contribution, extended in our approach, permitting the application of various notions of paradigmatic cluster was made by Bezdek [3], who introduced a number of prototype-based methods in his generalization of classification algorithms of Ball and Hall [2]. These methods are based on the summarization of the dataset by a number of prototypes that, initially, corresponded to the centroid of each cluster. This concept of prototype was later extended by considering other forms of paradigmatic structures, such as line segments or ellipsoids [4].

In summary, our methodology is primarily based on formulation of the qualitative-feature identification problem as that of finding parameterized models M_α and subsets F of the dataset representing the object being described such that some functional $Q(F, M_\alpha)$—measuring the degree of matching between model and subset—is optimized. Formulation of the qualitative description problem, however, in such a manner would result, however, in the generation of many structures with small extent (*e.g.*, two successive time-series values will always be an uptrend, a downtrend, or a stationary period) as it easier to explain (or model-match) smaller data subsets than those that constitute a significant portion of the dataset. For this reason, any successful methodology should also consider additional criteria based on the size of the substructure being explained. Other considerations might also lead to the definition of additional criteria to be satisfied by the clusters such as requirements intended to forbid the selective choice of data points satisfying some condition while arbitrarily ignoring others that do not meet that criterion (*e.g.*, picking selected time-series values that lie along a line while ignoring intermediate values that fall outside it).

A possible approach to the treatment of the resulting multiobjective optimization problems, which is close in spirit to minimum description-length methods [12], is based on the aggregation of the various objectives into a global measure of cluster quality. (Note, however, that our approach is based on measures of quality of description rather than on information-theoretic measures of model parsimoniousness.) This methodology was applied recently by Thranberend and Ruspini [15] to discover linear clusters in econometric time series.

A basic problem with this formulation however, is that it is inflexible, context-independent, and involves weighting of objectives. Clearly, in many applications, it might be desirable to relax somewhat requirements of explanatory quality (*i.e.*, the degree by which a subset matches a parameterized model) for large subsets while making them more stringent for smaller substructures. In the treatment presented in this work, we have approached this problem using a two-step method based first on identification of potentially interesting features, followed by a process of summarization intended to retain only selected features that summarize a subset of similar explanations.

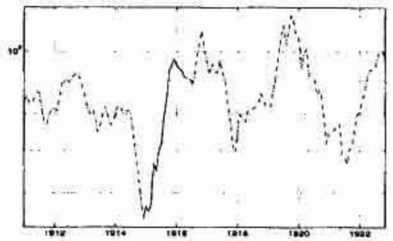
The first step of this process is based on a multiobjective optimization (Pareto optimality), which considers all applicable criteria, seeking to identify the *effective frontier* of the problem, *i.e.*, the set of all structures where it is necessary to consider tradeoffs between objectives. In our approach, this effective frontier is actually a collection of local multiobjective optima in the sense that its members are not *dominated* by other structures in their neighborhood. In our terminology, a solution (F, M) is non-dominated if there does not exist another model M' and a neighboring set F' (in some suitable topology defined in the power set of the dataset representing the object being studied) such that the values of all classification quality objectives are equal or larger for (F', M') than for (F, M), while one or more of those functionals have actually larger values for (F', M') than they do for (F, M).

In our experiments with the description of financial time series, we have employed two objective functionals, Q and S, measuring the degree of model matching and the feature size, respectively. These objectives are conflicting in the sense that, typically, good explanations tend to explain smaller subsets, while those describing larger structures have lower values of the matching quality functional Q. If the degree of matching $Q(F, M)$ between the model M and a subset F (in our case, the values of a time series on a subinterval of its temporal span) is sufficiently high, and so is the extent (or cardinality) $S(F)$ of F, and if (F, M) is non locally dominated, then (F, M) is incorporated as a potential component of the qualitative description.

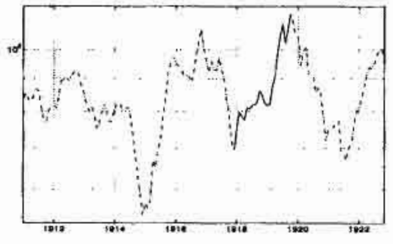
The second step is based on heuristics for the summarization, by a representative prototype, of similar solutions (*i.e.*, same model, close subsets) of the multiobjective optimization problem solved in the first step. More generally, the summarization process may be thought of as the discovery of interesting relations *within the effective frontier* on the basis of notions of importance of relations (of which *solution similarity* is but one example) provided, once again, by domain experts.

16.3 Approach

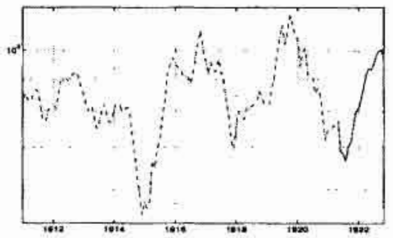
The process of generating qualitative descriptions of complex objects is composed of two sequential steps. The first step consists of the Pareto (or vector) optimization of multiple functionals characterizing various aspects of the adequacy of the generated features. The set of solutions of this problem, or *Pareto optimal frontier*, is then organized and summarized along relations deemed to be important by domain experts.



(a) Uptrend (12/1914–5/1916)



(b) Uptrend (12/1917–10/1919)

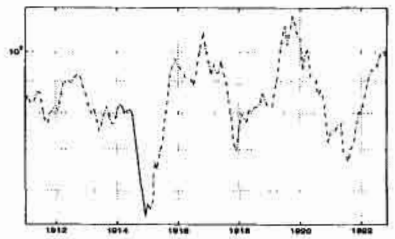


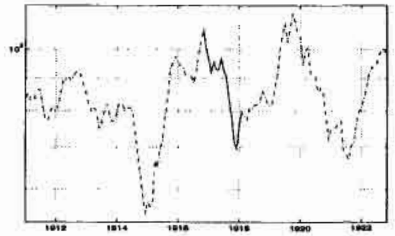
(c) Uptrend (5/1921–10/1922)

Fig. 16.2 Examples of uptrends identified by our optimization-based approach

16.3.1 Generating the effective frontier

In developing our approach to the generation of qualitative descriptions, we sought to attain the maximum possible flexibility in the definition of the approximate-model families characterizing both interesting structures and important relations between them. These minimal assumptions on the definition of interesting structures limit the application of insights derived from the model structure to


(a) Downtrend (2/1914–1/1915)


(b) Downtrend (11/1916–2/1918)

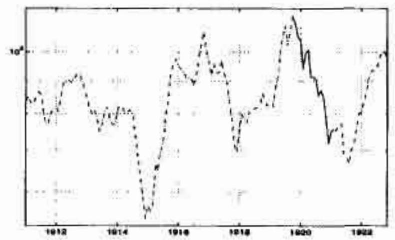
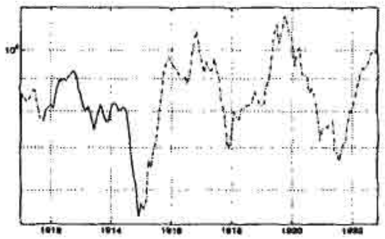
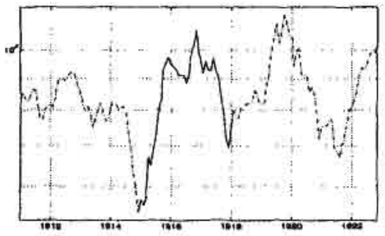

(c) Downtrend (10/1919–2/1921)

Fig. 16.3 Examples of downtrend intervals determined by our optimization-based approach

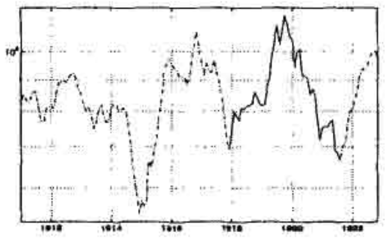
facilitate solution of the related optimization problems. While such exercises are possible in some instances, as is the case—to some extent—when employing functionals based on measures of functional approximation (*i.e.*, least-squares approximation errors), the cost of such minimal assumptions about model structure is, in general, that of the considerable computational effort associated with extensive searches in high-dimensional optimization spaces. In



(a) H&S (10/1911–1/1915)



(b) H&S (12/1914–2/1918)



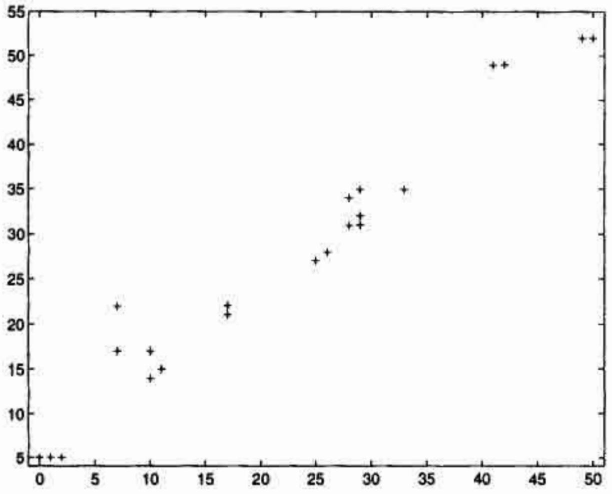
(c) H&S (12/1917–8/1921)

Fig. 16.4 Examples of head and shoulders (H&S) determined by our optimization-based approach

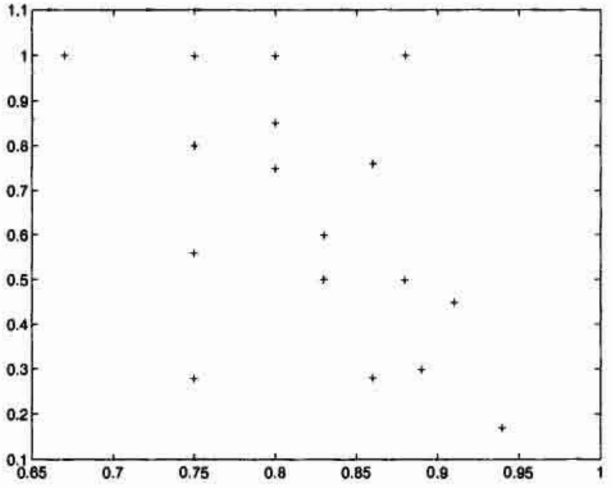
this regard, however, it is important to note that each object in the database must only be described once. Even in cases when it is desired to produce new descriptions based on a richer family of models, the required effort is generally limited to the discovery of instances of the new structures.

Seeking to develop a general methodology capable of solving the optimization problems associated with the generation of qualitative descriptions, we sought techniques based on the application of evolutionary-computation [1]. Past applications of genetic-algorithm (GA) techniques to the solution of optimization problems have been limited primarily to the minimization or maximization of single objectives. The typical GA approach to the treatment of multiple-objective optimization problems has been based on the introduction of weighted linear combinations of objective and penalty functions. (This aggregation approach was employed in earlier investigations on qualitative-description generation [15].) We have earlier commented on some of the problems associated with the lack of context dependence handicapping this type of treatment. In addition, it has also been found experimentally that the solutions of these aggregated-objective problems are very sensitive to small changes in the penalty function coefficients and in the weighting factors determining the relative importance of each of the aggregated objectives.

The results reported in this work were produced by application of a different approach to multiobjective optimization. Instead of combining multiple objectives into a single measure of descriptive-feature quality, we seek instead features that are not locally dominated, *i.e.*, that are locally optimal in the sense that there are no neighboring solutions (in some topology defined on the set of all subsets F of the dataset) that are at least equal in all objectives and strictly superior in at least one of them (*i.e.*, improvement of one objective results in a lower value for another). Solutions in this set, called the local Pareto-optimal frontier, cannot be improved by considering neighboring solutions as, necessarily, improvement in values of one functional requires lowering the value of another. Our algorithm is based on the niched Pareto method of Horn, Nafpliotis, and Goldberg [7, 8, 9] to solve multiobjective optimization problems and to generate the effective frontier of the optimization problem. A significant feature of this method is its reliance on restricted competition ("niches") between chromosomes to determine all non-dominated solutions of the multiobjective optimization problem. Another key characteristic of this type of method is the use of binary tournaments, known as *Pareto domination tournaments*, to determine the dominance status of two competitors A and B selected from the current chromosome population. Rather than comparing the selected competitors directly, a sample of the population is chosen to conduct a competitive tournament between members of the sample and each of the selected competitors. If one of the competitors is dominated by a member of the sample while the other competitor is not dominated at all, the nondominated individual wins the tour-



(a) Interval space



(b) Objective space

Fig. 16.5 Visualizing the effective frontier

nament. If both or neither are dominated, then fitness-sharing considerations are employed to determine the winner (whichever has the lower niche count). The sample size is used to control Pareto selection pressure in a manner similar to that employed to regulate tournament size in normal (single-objective) tournament selection.

16.3.2 Models

We have sought the maximum possible generality in the definition of interesting structures by relying on fuzzy-logic expressions that specify the requirements to be met by a good fit between model and substructure. These expressions employ fuzzy predicates (*e.g.*, `large peak`) to define an elastic constraint that measures the extent of compliance of any candidate structure with the constraint. In this formulation, a model is equivalent to a collection of elastic constraints, *i.e.*, to the conjunction of the fuzzy predicates defining each of the constraints in the collection. Employing the truth-combination formulas of fuzzy logic, it is possible, therefore, to measure, on a [0,1] scale, the quality of fitness of a substructure of the object being described to the model in question (with 1 being a perfect match and 0 corresponding to a very poor fit).

In our time series application we have used, for example, the following definition of uptrend:

```
Uptrend(interval)  ⊨ (∀ peaks in interval
                         peak  ⪯  next-peak),
                          ∧
                      (∀ valleys in interval
                         valley  ⪯  next-valley),
```

where $\preceq$ stands for the fuzzy predicate `approximately lower or equal`. This expression states that, in an uptrend, every peak is (approximately) lower than or equal to its successor and every valley is (approximately) lower than or equal its successor. The ground predicate `approximately lower or equal` is modeled, using standard conventions, by a trapezoidal function having a soft discontinuity at 0. This formula permits to compute, by application of the combination formulae of fuzzy logic, the degree by which the values in any time interval, satisfy the definition of financial uptrend.

In addition to its generality, this logic-based approach is noteworthy because of its ability to produce clear, explicit, descriptions of the conditions that must be met to qualify a structure as being explained by a model. These logical expressions are also found to be, in practice, easy to modify or correct whenever the solutions produced by a qualitative-description effort do not correspond to the intuitive notion that is being modeled, the logical expression may be readily analyzed and corrected.

16.3.3 Experimental optimization results

We applied our GA-based multiobjective optimization approach to the identification of significant technical-analysis [11] patterns in financial time series. Our extension of the niched Pareto optimization method of Horn, Napfliotis, and Goldberg was applied to time series of monthly averages of closing prices of various financial commodities and indexes. We present results of efforts for the selection of three types of technical analysis patterns: *uptrends*, *downtrends*, and *heads and shoulders* (H&S). In this particular application, a solution is a crisp interval that meets, to some significant extent, the logic-based definition of those patterns.

In particular, we illustrate results of the application of our GA approach to the monthly averages of closing values of the Dow-Jones Industrial Average index (DJIA) between 1912 and 1922 (Fig. 16.1).

In this particular problem we considered two conflicting objectives, each measuring the extent by which time-series values in a crisp time interval, meet explicit criteria for qualitative-feature quality. The first objective—*quality of fit*— measures the extent to which the time-series values correspond to a financial uptrend, downtrend, or H&S interval. The second objective—*extent*—measures, through a simple linear functional, the length of the interval being explained.

Pair of numbers representing an interval of time were coded as GA chromosomes. A population of size 200 was modified by the GA over a total period of 600 generations. Cross-over probabilities were chosen in the [0.7, 0.9] range, while the mutation probability was 0.1. The niche size (*i.e.*, the proportion of the population where the sharing function is applied) varied from 1% to 10% of the maximum value encoded as an interval end in the chromosomes. The niche size allows the distribution of the population over different solutions in the search space (*i.e.*, it prevents all chromosomes from converging to a few solutions). For computational simplicity, niche counts are calculated on the partly filled next-generation population rather than on the current population [9].

In our experiments we employed a tournament size between 4 and 20 to control the selection pressure. In this regard, it is important to note that our evaluations have showed that the niched Pareto algorithm is somewhat sensitive to the selection pressure and to the sharing pressure applied. Small values of the tournament size (close to 1–2% of the population size) result in too many dominated individuals (*i.e.*, a very fuzzy front) while higher values (more than 20%) result in premature convergence to a small portion of the front [6]. Finally, a small percentage of random individuals were introduced in each generation to make the GA more sensible to new zones [6]. These individuals were assigned to zones not represented till then by any of the features that we sought to identify.

Uptrends were defined, as previously detailed, by means of fuzzy-logic expressions based on comparison of the values of successive peaks and valleys (*i.e.*, local maxima and minima) in the time series. Examples of uptrends identified by our optimization-based approach are shown in Fig. 16.2. Examples of downtrend intervals determined by our optimization-based approach are shown in Fig. 16.3.

Finally, Fig. 16.4 shows examples of a more complex structure—head & shoulders— that were also identified by our approach. Qualitative head & Shoulder models (H&S) are more complex than those characterizing uptrends and downtrends.

16.3.4 Describing the effective frontier

The niched Pareto GA produces a set of nondominated solutions of the multiobjective optimization problem. The set of such solutions—corresponding to various qualitative features of the object being described—may then be summarized and related according to several criteria.

The first type of relations that may be employed to enhance a description consisting solely of the Pareto-optimal frontier, are domain-specific relations deemed to be important by domain experts. In our financial time-series application, we have considered two such relations: temporal inclusion (*i.e.*, interval I is a subset of interval I'), and temporal succession (*i.e.*, interval I temporally follows interval I').

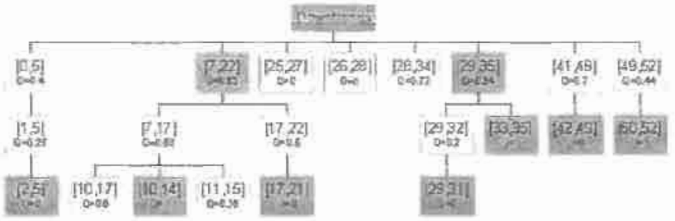
In addition to this relation, examination of the effective frontier indicates that many non-dominated solutions are themselves similar in the sense that, while being different from a crisp viewpoint, they represent—by means of the same explanatory model— similar intervals (*i.e.*, intervals having similar end points). This observation immediately suggests the application of clustering procedures to the effective frontier to produce a more compact and understandable description of the salient structures of the time series.

To better understand the nature of the similarity between members of the effective frontier, the solutions of a typical optimization exercise, corresponding to a single model of interesting structure, are plotted in the solution state (i, I), where i is the leftmost point of the interval and I is its rightmost point. The proximity of solutions in this space suggests the application of similarity-based clustering techniques to group neighboring solutions and to summarize the effective frontier by simple specification of cluster prototypes.

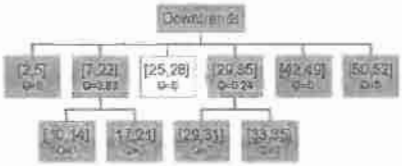
Fig. 16.5(b) which plots values of the two objective functionals employed to determine the effective frontier suggests another mechanism to further simplify the description of the effective frontier. (In Fig. 16.5(b), low values of the quality index Q and the extent index S correspond, because of the particular definition used in our experiments, to better or more desirable solutions.) This additional simplification is based on the observation that certain solutions of the optimization problem—while not being dominated by their neighbors—are, in fact, dominated by other, similar solutions and may then be eliminated from consideration.

Fig. 16.5(a) is also useful for visualizing relations of temporal inclusion in the effective frontier. Note that, since it is true that, for any interval $[i, I]$, $i \leq I$, all points on the effective frontier lie above the diagonal $i = I$, with larger intervals lying close to the upper-left hand corner of the diagram and singletons lying on the diagonal $i = I$. Solutions related by an inclusion relation are easily visualized because they lie on the same perpendicular to the diagonal $i = I$. Indexes measuring the separation between two different perpendiculars to that diagonal may be employed to define notions of approximate inclusion.

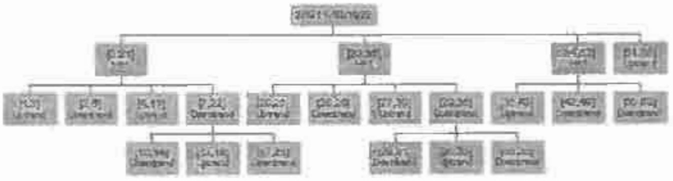
Our summarization algorithm aims to produce compact representations of the Pareto-optimal frontier by summarization of its significant characteristics and description of important relations between features. In our time-series application, this process involved the following steps: (1) organization of effective frontier, on the basis of the notion of approximate inclusion, as interval hierarchies (trees), (2) exclusion of solutions of the multiobjective optimization problem that are dominated by similar solutions (*fuzzy domination)*, (3) grouping of similar solutions (clustering) and summarization of those clusters



(a) GA output organized by inclusion (downtrends)



(b) Summary of downtrends in the effective frontier



(c) Summarizing interesting epochs (all models)

Fig. 16.6 Describing the frontier

by prototypes, (4) merging of remaining similar intervals having close values of the objective functional Q.

Fig. 16.6 illustrates this summarization and hierarchical interrelation process. Fig. 16.6(a) shows an initial organization of all solutions in the effective frontier as a tree showing temporal inclusion relationships. Non-shaded boxes indicate solutions that are later eliminated by our description algorithm (be-

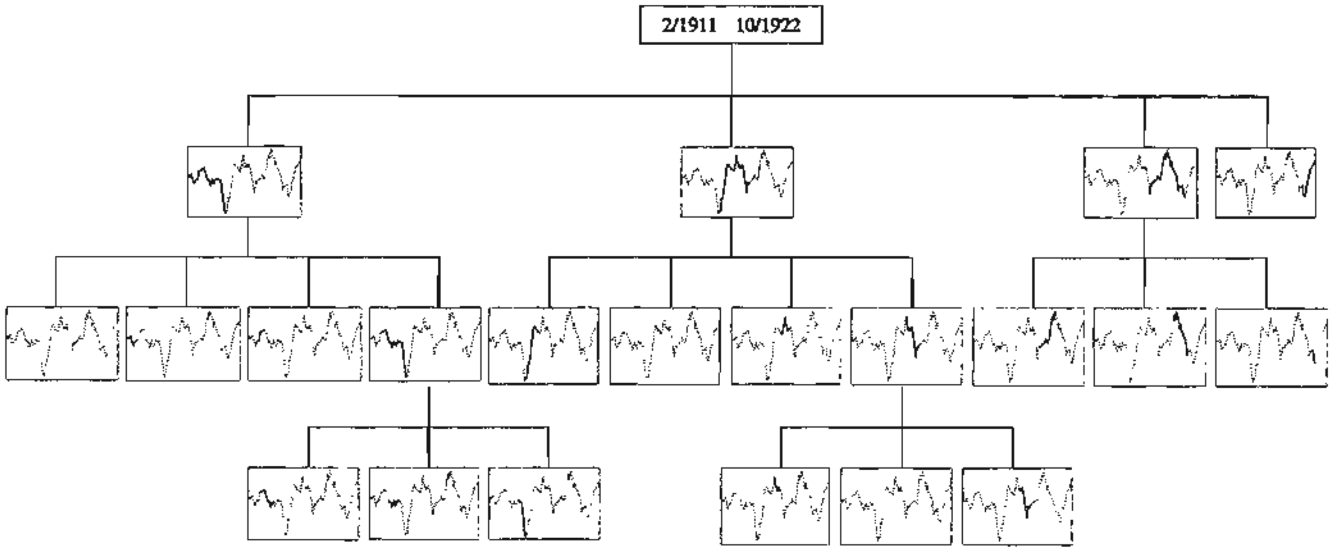


Fig. 16.7 Summarizing interesting epochs in the DJIA (1911–1922)

cause of poor quality) or summarized by replacement with prototypes of clusters of similar solutions. The outcome of these actions, combined with the merging close intersecting intervals (indicated also by a non-shaded box), is shown in Fig. 16.6(b). Fig. 16.6(c) shows the results of the application of these summarization and interrelation operations to the output of a GA optimization process seeking structures matching all definitions of interesting feature (*i.e.*, uptrends, downtrends, and head & shoulders (H&S)). The same results are also shown in Fig. 16.7 employing, in this case, illustrations of the actual time-series epochs, rather than values of the end points of the corresponding intervals.

16.4 Conclusions

In this chapter, we have described soft-computing based approaches for the generation of representations of complex objects by identification of significant qualitative features (called "perceptions" by Zadeh) and by automatic recognition of significant relationships between those features. The objectives of this representation are both the annotation of complex objects in terms of features meaningful to database users and the eventual analysis of the underlying systems by knowledge-discovery techniques that operate on the qualitative representations of the objects rather than on the original data. We have presented, for this purpose, a group of novel fuzzy clustering techniques that explore object descriptions or large data samples seeking to find structures that match approximately certain prototypical or paradigmatic structures. Posing the related clustering problem as a large-scale optimization problem, evolutionary computation techniques have been employed to isolate structures that are extensive, in the sense that they describe significant portions of the data sample, that they are interesting, in the sense that they have a good degree of matching with prototypical examples, and that may also satisfy other desirable conditions. Our approach relies on a local Pareto-optimal methodology intended to discover all structures that are non-dominated in a Pareto-optimal sense. The algorithm employed to solve the underlying multiobjective optimization problem is a generalization of the niched Pareto method of Horn, Napfliotis, and Goldberg. The output of this process is then organized and related in terms of approximate relations. These relations are, once again, identified by their proximity, according to a user-defined similarity, to certain paradigmatic, or "interesting" relations, also provided by the user. The methodology to extract qualitative features and to relate such features through approximate relations of user interest is illustrated through examples of its application to economic time-series analysis.

References

[1] T. Bäck, D. Fogel, and Z. Michalewicz, eds., *Handbook of Evolutionary Computation.* Institute of Physics Publishing and Oxford University Press, 1997.

[2] G. Ball and D. Hall, "Clustering technique for summarizing multivariate data," *Behavioural Sciences*, vol. 12, pp. 153–155, 1967.

[3] J. C. Bezdek, *Fuzzy Mathematics in Pattern Classification.* PhD thesis, Cornell University, 1973.

[4] J. C. Bezdek, "Fuzzy clustering," in *Handbook of Fuzzy Computation* (E. H. Ruspini, P. P. Bonissone, and W. Pedrycz, eds.), ch. F6.2, Institute of Physics Press, 1998.

[5] J. C. Bezdek and S. K. Pal, eds., *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data.* IEEE Press, 1992.

[6] C. Fonseca and P. Fleming, "Multiobjective genetic algorithms made easy: Selection, sharing and mating restriction," in *Proc. First IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems*, pp. 44–52, 1995.

[7] C. Fonseca and P. J. Fleming, "Multiobjective optimization," in *Handbook of Evolutionary Computation* (T. Bäck, D. Fogel, and Z. Michalewicz, eds.), Institute of Physics Publishing and Oxford University Press, 1997.

[8] J. Horn, "Multicriterion decision making," in *Handbook of Evolutionary Computation* (T. Bäck, D. Fogel, and Z. Michalewicz, eds.), Institute of Physics Publishing and Oxford University Press, 1997.

[9] J. Horn, N. Nafpliotis, and D. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proc. First IEEE Conf. on Evolutionary Computation*, pp. 82–87, 1994.

[10] R. Krishnapuram and J. Keller, "A possibilistic approach to clustering," *IEEE Transactions on Fuzzy Systems*, pp. 98–110, 1993.

[11] M. J. Pring, *Technical Analysis Explained : The Successful Investor's Guide to Spotting Investment Trends and Turning Points.* McGraw-Hill, 5th ed., 1991.

[12] J. Rissanen, *Stochastic Complexity in Statistical Inquiry.* World Scientific, 1989.

[13] E. H. Ruspini, "A new approach to clustering," *Information and Control*, vol. 15, pp. 22–32, 1969.

[14] E. H. Ruspini, "A theory of fuzzy clustering," in *Proc. 1978 IEEE Intl. Conf. on Decision and Control*, IEEE Press, 1978.

[15] K. Thranberend and E. H. Ruspini, "Subtractive optimization methods for hierarchical fuzzy clustering," in *Proc. 1996 Conference International Fuzzy Systems Association*, 1996.

[16] L. A. Zadeh, "From computing with numbers to computing with words–from manipulation of measurements to manipulation of perceptions," *IEEE Transactions on Circuits and Systems*, vol. 45, pp. 105–119, 1999.

Chapter 17

NEURO-FUZZY MODELS FOR FEATURE SELECTION AND CLASSIFICATION

R. K. De and S. K. Pal

Machine Intelligence Unit,
Indian Statistical Institute,
Calcutta 700035, INDIA
e-mail: {*rajat,sankar*}*@isical.ac.in*

Abstract

The chapter describes neuro-fuzzy models for feature selection and classification. Feature selection is performed under both supervised and unsupervised learning. The task of classification is done using a knowledge-based system under supervised learning. The methodology for feature selection involves minimization of fuzzy feature evaluation indices, defined in terms of membership functions, in connectionist framework. For the unsupervised method, the algorithm does not need to know the number of clusters *a priori*. The methodology for designing a knowledge-based system involves development of a technique for generating an appropriate architecture of a neural network in terms of hidden nodes and links. The effectiveness of these systems, along with comparisons, is adequately demonstrated on various real life applications.

17.1 Introduction

Feature selection or extraction is a process of selecting a mapping of the form $\mathbf{x}' = f(\mathbf{x})$ by which a sample $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in an n-dimensional measurement space ($\Re^n$) is transformed into a point $\mathbf{x}' = (x'_1, x'_2, \ldots, x'_{n'})$ in an n'-dimensional ($n' < n$) feature space ($\Re^{n'}$). The problem of feature selection (extraction) deals with choosing (generating) some x_is (called x'_js) from the measurement space to constitute the feature space. The main objective of these processes is to retain the optimum salient characteristics necessary for the recognition process, and to reduce the dimensionality of the measurement space so that effective and easily computable algorithms can be devised for efficient categorization.

After the feature space is obtained, the next task of a pattern recognition system is classification. Classification can be viewed as a two-fold task, consisting of learning the invariant and common properties (features) of a set of samples characterizing a class, and of deciding that a new sample is a possible member of the class by noting that it has the features common to those of the set of samples.

Incorporation of fuzzy set theory enables one to deal with uncertainties in the different tasks of a pattern recognition system — arising from deficiency (in terms of, *e.g.*, vagueness, incompleteness) in information — in an efficient manner. Artificial neural networks (ANNs), being fault tolerant, adaptive, generalizable, and being suited to massive parallelism, are widely used in learning and optimization tasks. In the last few years, numerous attempts have been made to integrate the merits of fuzzy set theory and ANNs, under the heading 'neuro-fuzzy computing', with an aim of making the systems more intelligent [10]. The theories of fuzzy sets, neural networks and neuro-fuzzy computing constitute, among others, some important tools of a new paradigm of research called 'soft computing' [6, 25, 26, 29, 31, 32].

The present chapter, in this regard, has two parts. The first describes neuro-fuzzy systems that were recently developed for feature selection under both supervised and unsupervised learning. The methodology involves connectionist minimization of fuzzy feature evaluation indices which are defined based on the membership functions. The lower the value of the indices, higher is the compactness of individual classes/clusters and separation between various classes/clusters. For the supervised method, the membership function denotes the degree of belongingness of a pattern to a class, whereas that for the unsupervised method provides the degree of similarity between two patterns in both the original and transformed feature spaces. A set of weighting coefficients corresponding to the features is introduced which provides flexible modeling of the class/cluster structures and reflects the individual importance of the features after minimization of the indices. For the unsupervised method the transformed space is obtained through this set of weighting coefficients. Both algorithms consider interdependence of the original features.

In another part, we have provided the design of a neuro-fuzzy knowledge-based system for pattern classification, where a new idea of knowledge encoding among the connection weights of neural networks, particularly of a fuzzy multilayer perceptron (MLP) [28], is described. The methodology involves development of a technique for generating an appropriate architecture of the fuzzy MLP in terms of hidden nodes and links. The model is capable of handling input in numerical, linguistic and set forms, and can tackle uncertainty due to overlapping classes.

The effectiveness of the algorithms along with comparisons is demonstrated extensively on various real life problems. Here we include the results on two data sets concerning classification of speech (vowel) sounds [25] and Iris flowers [10]. The outcome of these methods is adequately justified using k-NN classifier, scatter plots and various graphical analysis. The readers may refer to [9, 19, 24] for other results.

17.2 A brief review

Here we provide a brief review of literature on connectionist approaches for feature selection and neuro-fuzzy knowledge-based systems for classification. Note that in the area of pattern recognition, neuro-fuzzy approaches have been adopted mostly for designing classification/clustering methodologies. Comparatively, the problem of feature selection has not been addressed much.

17.2.1 Feature selection

Some of the attempts made for feature selection in connectionist framework are mainly based on multilayer feedforward networks [5, 8, 13, 34, 35]. These methods adopt supervised learning scheme.

Battiti [3] has investigated an application of mutual information criterion to evaluate a set of candidate features and to select an informative subset to be used as input data for a neural network classifier. Mutual information, being a measure of arbitrary dependencies between random variables, is used for assessing the "information content" of features in complex classification tasks.

The fact that mutual information is independent of the coordinates chosen permits a robust estimation. In addition, the use of mutual information for tasks characterized by high input dimensionality requires suitable approximations to reduce its computation time. An algorithm is described based on a "greedy" selection of the features. It takes the mutual information with respect to the output class as well as the already-selected features into account.

Setino *et al.* [36] have demonstrated how a three-layer feedforward neural network can be used to select the input attributes that are most important for discriminating classes in a given set of input patterns. The algorithm is based on network pruning. By adding a penalty term to the error function of the network, redundant network connections can be distinguished from those relevant ones, by their small weights, when the network training process is over. A simple criterion to remove an attribute, based on the accuracy rate of the network, is developed. The network is trained after removal of an attribute, and the selection process is repeated until no attribute meets the criterion for removal.

The MLP-based method of De *et al.* [8] is developed on the idea that the absence of an important feature is likely to influence the output of a trained MLP significantly. On the other hand, for a less important feature, the output is not expected to change much with the variation of its value. First of all, an MLP is trained with a data set. Then the patterns with only one features absent (*i.e.*, by setting the corresponding feature value to zero) are presented to the network, and the outputs are noted. The deviation of an output vector from the one generated by the corresponding original data point is computed. A feature is considered more important if the average deviation, computed over the entire data set, for that feature is more.

Laar and Heskes [14] have presented an algorithm that performs input (feature) selection based on an ensemble of neural networks. Initially, the algorithm considers an ensemble of neural networks with all the features. Then it starts eliminating one after another until all the features are removed. Finally, the optimal ensemble is determined.

The method of Castellano *et al.* [7] performs a backward selection of features by successively removing input nodes in a network trained with the complete set of input features. Input nodes are removed, along with their connections, and the remaining weights are adjusted in such a way that the overall input-output behavior learnt by the network is kept approximately unchanged. A simple criterion to select input nodes to be removed is developed.

Bauer Jr. *et al.* [4] have presented a method of feature selection using an MLP, which is based on a signal-to-noise (SNR) saliency measure. The measure determines the saliency of a feature by comparing it to that of an injected noisy feature. The method is applied on breast cancer diagnosis, the US Congressional voting records and the Pima Indians Diabetes problems.

17.2.2 Knowledge-based systems

Generally, ANNs consider a fixed topology of neurons connected by links in a pre-defined manner. These connection weights are usually initialized by small random values. Knowledge-based networks [11, 38] constitute a special class of ANN that consider crude domain knowledge to generate the initial network architecture which is later refined in the presence of training data. This process helps in reducing the searching space and time while the network traces the optimal solution. Growing of links and/or pruning of nodes is also done in order to generate the optimal network architecture.

Several attempts, based on neuro-fuzzy approach, to the design of knowledge-based systems have been reported. Masuoka *et al.* [18] have used knowledge in the form of membership functions and fuzzy rules (in *And-Or* form), extracted from experts, to build and preweigh the structured neural network which is then tuned using selected learning data. This neural model consists of the input variable membership net, the rule net, and the output variable net. Modified fuzzy rules, extracted from the trained neural network using pruning, are then evaluated and unsuitable rules corrected using relearning.

Machado and Rocha [16] have used a connectionist knowledge base involving fuzzy numbers at the input layer, fuzzy *And* at the hidden layers and fuzzy *Or* at the output layer. The input data, in symbolic or numeric forms, is converted to possibility degrees. The hidden layers chunk input evidences into clusters of information for representing regular patterns of the environment. The output layer computes the degree of possibility of each hypothesis. The initial network architecture is generated using *knowledge graphs* elicited from experts by the application of the knowledge acquisition technique of [15]. The experts express their knowledge about each hypothesis of the problem domain by selecting an appropriate set of evidences and building an acyclic weighted *And-Or* graph to describe how these may be combined to support decision making. Inference, inquiry and explanation are possible during consultation.

Pedrycz and Rocha [33] have used basic aggregation neurons (*And/Or*) and referential processing units (matching, dominance and inclusion neurons) to design knowledge-based networks. The inhibitory and excitatory characteristics are captured by embodying direct and complemented input signals. Applications in decision-making, diagnostic and control problems are outlined, employing fully supervised learning. Another related approach by Hirota and Pedrycz [12] has incorporated fuzzy clustering for developing the geometric constructs leading to the design of knowledge-based networks. Its applications to classification problems are also described.

Recently, the theory of rough sets has been used to develop knowledge-based neural networks [1]. Crude domain knowledge in the form of logical rules is generated from the data set using rough set theory, and is encoded among the connection weights. This helps one to automatically generate an appropriate network architecture in terms of hidden nodes and links. The method models arbitrary decision regions with multiple object representatives. Later Mitra *et al.* [20] have provided a method of hybridizing the various tools of soft computing— fuzzy sets, rough sets, neural networks and genetic algorithms— in order to develop modular networks for the purpose of classification and rule generation.

17.3 Neuro-fuzzy methods for feature selection

In this section we describe two recently-developed neuro-fuzzy feature selection methods [2, 9, 22, 24] under both supervised and unsupervised learning. The methodologies involve formulation of fuzzy feature evaluation indices followed by their minimization using connectionist models.

17.3.1 Supervised feature selection

Let us consider an n-dimensional feature space Ω containing $x_1, x_2, x_3, \ldots, x_i, \ldots, x_n$ features (components). Let there be M classes $C_1, C_2, \ldots, C_M$.

Fuzzy feature evaluation index

The feature evaluation index for a subset $(\Omega_{\mathbf{x}})$ containing few of these n features is defined as

$$E = \sum_k \sum_{\mathbf{x} \in C_k} \frac{H_k(\mathbf{x})}{\sum\limits_{k' \neq k} H_{kk'}(\mathbf{x})} \times \alpha_k, \tag{17.1}$$

where $\mathbf{x}$ is constituted by the features of $\Omega_{\mathbf{x}}$ only.

$$H_k(\mathbf{x}) = \mu_{C_k}(\mathbf{x}) \times (1 - \mu_{C_k}(\mathbf{x})) \tag{17.2}$$

and

$$H_{kk'}(\mathbf{x}) = \frac{1}{2}[\mu_{C_k}(\mathbf{x}) \times (1 - \mu_{C_{k'}}(\mathbf{x}))] + \frac{1}{2}[\mu_{C_{k'}}(\mathbf{x}) \times (1 - \mu_{C_k}(\mathbf{x}))]. \tag{17.3}$$

$\mu_{C_k}(\mathbf{x})$ and $\mu_{C_{k'}}(\mathbf{x})$ are the membership values of the pattern $\mathbf{x}$ in classes C_k and $C_{k'}$ respectively. α_k is the normalizing constant for class C_k which takes care of the effect of the relative size of the classes.

Note that, H_k is zero (minimum) if $\mu_{C_k} = 1$ *or* 0, and is 0.25 (maximum) if $\mu_{C_k} = 0.5$. On the other hand, $H_{kk'}$ is zero (minimum) when $\mu_{C_k} = \mu_{C_{k'}} = 1$ *or* 0, and is 0.5 (maximum) for $\mu_{C_k} = 1$, $\mu_{C_{k'}} = 0$ or *vice-versa*.

Therefore, the term $\frac{H_k}{\sum_{k' \neq k} H_{kk'}}$ is minimum if $\mu_{C_k} = 1$ and $\mu_{C_{k'}} = 0$ for all $k' \neq k$ *i.e.*, if the ambiguity in the belongingness of a pattern $\mathbf{x}$ to classes C_k and $C_{k'}$ $\forall k' \neq k$ is minimum (the pattern belongs to only one class). It is maximum when $\mu_{C_k} = 0.5$ for all k. In other words, the value of E decreases as the belongingness of the patterns increases for only one class (*i.e.*, compactness of individual classes increases) and at the same time decreases for other classes (*i.e.*, separation between classes increases). E increases when the patterns tend to lie at the boundaries between classes (*i.e.*, $\mu \to 0.5$). Our objective is, therefore, to select those features for which the value of E is minimum.

Note that the factor α_k, corresponding to the class C_k, is introduced for normalizing the effect of the size of the classes. In the present investigation, we have chosen $\alpha_k = 1 - P_k$, where P_k is *a priori* probability for class C_k. However, other expressions like $\alpha_k = \frac{1}{|C_k|}$ or $\alpha_k = \frac{1}{P_k}$ could also have been used.

The membership ($\mu_{C_k}(\mathbf{x})$) of a pattern $\mathbf{x}$ to a class C_k is defined with a multi-dimensional π-function [30] which is given by

$$\begin{aligned} \mu_{C_k}(\mathbf{x}) &= 1 - 2d_k^2(\mathbf{x}), \quad 0 \leq d_k(\mathbf{x}) < \tfrac{1}{2}, \\ &= 2[1 - d_k(\mathbf{x})]^2, \ \tfrac{1}{2} \leq d_k(\mathbf{x}) < 1, \\ &= 0, \qquad\qquad\quad \text{otherwise}. \end{aligned} \tag{17.4}$$

$d_k(\mathbf{x})$ is the distance of the pattern $\mathbf{x}$ from $\mathbf{m}_k$ (the center of class C_k). It can be defined as

$$d_k(\mathbf{x}) = \left[\sum_i w_i^{r_k} \left(\frac{x_i - m_{ki}}{\lambda_{ki}}\right)^{r_k}\right]^{\frac{1}{r_k}}, \; w_i \in [0,1], \tag{17.5}$$

where

$$\lambda_{ki} = 2 \max_{\mathbf{x} \in C_k} [|x_i - m_{ki}|], \tag{17.6}$$

and

$$m_{ki} = \frac{\sum\limits_{\mathbf{x} \in C_k} x_i}{|C_k|}. \tag{17.7}$$

Equations (17.4)-(17.7) are such that the membership $\mu_{C_k}(\mathbf{x})$ of a pattern $\mathbf{x}$ is 1 if it is located at the mean of C_k, and 0.5 if it is on the boundary (*i.e.*, ambiguous region).

The membership values (μ) of the sample points of a class are dependent on w_i. The values of w_i (< 1) make the function of (17.4) flattened along the axis of x_i. The lower the value of w_i, the higher is the extent of flattening. In the extreme case, when $w_i = 0$, $d_k = 0$ and $\mu_{C_k} = 1$ for all the patterns.

In pattern recognition literature, the weight w_i (defined by (17.5)) can be viewed to reflect the relative importance of the feature x_i in measuring the similarity (in terms of distance) of a pattern to a class. It is such that the higher the value of w_i, the greater is the importance of x_i in characterizing/discriminating a class/between classes. $w_i = 1(0)$ indicates most (least) importance of x_i.

Therefore, the compactness of the individual classes and the separation between the classes, as measured by E (given by (17.1)), is essentially a function of $\mathbf{w}$ ($= [w_1, w_2, \ldots w_n]$), if we consider all the n features together. The problem of feature selection/ranking thus reduces to finding a set of w_is for which E becomes minimum; w_is indicate the relative importance of x_is in characterizing/discriminating classes. The task of minimization has been performed using gradient descent technique in a connectionist framework. A new connectionist model is developed for this purpose. This is described below.

Connectionist model

The network (Fig. 17.1) consists of two layers, namely, input and output. The input layer represents the set of all n features and the output layer corresponds to the pattern classes. Input nodes accept activations corresponding to the

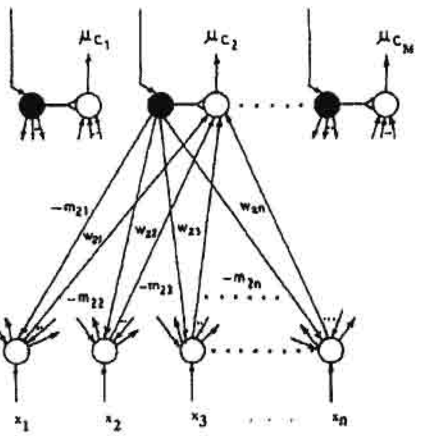


Fig. 17.1 A schematic diagram of the neural network model for supervised feature selection. Black circles represent the auxiliary nodes, and white circles represent input and output nodes. Small triangles attached to the output nodes represent the modulatory connections from the respective auxiliary nodes

feature values of the input patterns. The output nodes produce the membership values of the input patterns corresponding to the respective pattern classes. With each output node, an auxiliary node is connected which controls the activation of the output node through modulatory links. An output node can be activated from the input layer only when the corresponding auxiliary node remains active. Input nodes are connected to the auxiliary nodes through feedback links. The weight of the feedback link from the auxiliary node, connected to the kth output node (corresponding to the class C_k), to the ith input node (corresponding to the feature x_i) is equated to $-m_{ki}$. The weight of the feedforward link from the ith input node to the kth output node provides the degree of importance of the feature x_i, and is given by

$$W_{ki} = (\frac{w_i}{\lambda_{ki}})^{r_k}. \tag{17.8}$$

During training, the patterns are presented at the input layer and the membership values are computed at the output layer. The feature evaluation index for these membership values is computed (using (17.1)) and the values of w_is are updated in order to minimize this index. Note that, λ_{ki}s and m_{ki}s are directly computed from the training set and kept fixed during updating of w_is. The auxiliary nodes are activated (*i.e.*, activation values are equated to unity) one at a time, while the others are made inactive (*i.e.*, the activation values are kept fixed at 0). Thus, during training, at a time only one output node is allowed to get activated. For details concerning the operation of the network, refer to the Appendix.

The training phase of the network takes care of the task of minimization of E (defined by (17.1)) with respect to $\mathbf{w}$ which is performed using simple gradient-descent technique. The change in w_i ($\triangle w_i$) is computed as

$$\triangle w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i, \tag{17.9}$$

where η is the learning rate. The term $\frac{\partial E}{\partial w_i}$ is computed using various node activations and link weights of the network. For computation of $\frac{\partial E}{\partial w_i}$, one may refer to Appendix 17.6.

Algorithm for learning $\mathbf{w}$

- Calculate the mean vectors ($\mathbf{m}_k$) of all the classes from the data set. Set the weight of the feedback link from the auxiliary node corresponding to the class C_k to the input node i as $-m_{ki}$ (for all i and k).
- Get the values of r_ks (in (17.5)) and λ_{ki}s (in (17.6)) from the data set, and initialize the weight of the feedforward link from ith input node to kth output (for all values of i and k) node.
- For each input pattern :
  - Present the pattern vector to the input layer of the network.
  - Activate only one auxiliary node at a time.
    Whenever an auxiliary node is activated, it sends the feedback to the input layer. The input nodes, in turn, send the resultant activations to the output nodes. The activation of the output node (connected to the active auxiliary node) provides the membership value of the input pattern to the corresponding class. Thus, the membership values of the input pattern corresponding to all the classes are computed by sequentially activating the auxiliary nodes one at a time.
  - Compute the desired change in weights of the feedforward links to be made using the updating rule given in (17.9).

- Compute total change in w_i for each i, over the entire set of patterns. Update w_i (for all i) with the average value of $\triangle w_i$.
- Repeat the entire process until convergence, *i.e.*, the change in E becomes less than certain predefined small quantity.

After the training phase is over *i.e.*, after convergence, $E(\mathbf{w})$ attains a local minimum. In that case, the weights of the feedforward links indicate the order of importance of the features. Note that, the method considers the effect of interdependencies of the features unlike those in [9, 21, 23]. A detailed theoretical analysis, concerning the convergence of E and validation of the ordering of features, is described in [9].

17.3.2 Unsupervised feature selection

Neuro-fuzzy unsupervised feature selection methodology, as in the case of supervised algorithm (Section 17.3.1), also involves a formulation of a fuzzy feature evaluation index and then its minimization in a connectionist framework. The membership function for the realization of this index is defined in terms of distance measure and weighting coefficients.

Fuzzy feature evaluation index

Let, μ^O_{pq} be the degree that both the pth and qth patterns belong to the same cluster in the n-dimensional original feature space, and μ^T_{pq} be that in the n'-dimensional ($n' \leq n$) transformed feature space. μ values determine how similar a pair of patterns are in the respective features spaces. That is, μ may be interpreted as the membership value of a pair of patterns belonging to the fuzzy set "similar". Let, s be the number of samples on which the feature evaluation index is computed.

The feature evaluation index for a set (Ω) of transformed features is defined as

$$E = \frac{2}{s(s-1)} \sum_{p} \sum_{q \neq p} \frac{1}{2} [\mu^T_{pq}(1 - \mu^O_{pq}) + \mu^O_{pq}(1 - \mu^T_{pq})]. \qquad (17.10)$$

It has the following characteristics:
(i) For $\mu^O_{pq} < 0.5$ as $\mu^T_{pq} \to 0$, E decreases. For $\mu^O_{pq} > 0.5$ as $\mu^T_{pq} \to 1$, E decreases. In both the cases, the contribution of the pair of patterns to the evaluation index E becomes minimum ($= 0$) when $\mu^O_{pq} = \mu^T_{pq} = 0$ or 1. (ii) For

$\mu^O_{pq} < 0.5$ as $\mu^T_{pq} \to 1$, E increases. For $\mu^O_{pq} > 0.5$ as $\mu^T_{pq} \to 0$, E increases. In both the cases, the contribution of the pair of patterns to E becomes maximum ($= 0.5$) when $\mu^O_{pq} = 0$ and $\mu^T_{pq} = 1$, or $\mu^O_{pq} = 1$ and $\mu^T_{pq} = 0$. (iii) If $\mu^O_{pq} = 0.5$, the contribution of the pair of patterns to E becomes constant ($= 0.25$), *i.e.*, independent of μ^T_{pq}.

Characteristics (i) and (ii) can be verified as follows. From (17.10) we have

$$\frac{\partial E}{\partial \mu^T_{pq}} = \frac{1}{s(s-1)}(1 - 2\mu^O_{pq}). \tag{17.11}$$

For $\mu^O_{pq} < 0.5$, $\frac{\partial E}{\partial \mu^T_{pq}} > 0$. This signifies that E decreases (increases) with decrease (increase) in μ^T_{pq}. For $\mu^O_{pq} > 0.5$, $\frac{\partial E}{\partial \mu^T_{pq}} < 0$. This implies that E decreases (increases) with increase (decrease) in μ^T_{pq}. Since $\mu^T_{pq} \in [0,1]$, E decreases (increases) as $\mu^T_{pq} \to 0$ (1) in the former case, and $\mu^T_{pq} \to 1$ (0) in the latter. ♣

Therefore, the feature evaluation index decreases as the membership value representing the degree of belonging of pth and qth patterns to the same cluster in the transformed feature space tends to either 0 (when $\mu^O < 0.5$) or 1 (when $\mu^O > 0.5$). In other words, the feature evaluation index decreases as the decision on the similarity between a pair of patterns (*i.e.*, whether they lie in the same cluster or not) becomes more and more crisp. This means, if the intercluster/intracluster distances in the transformed space increase/decrease, the feature evaluation index of the corresponding set of features decreases. Therefore, our objective is to select those features for which the evaluation index becomes minimum; thereby optimizing the decision on the similarity of a pair of patterns with respect to their belonging to a cluster.

The membership function μ_{pq} in a feature space, satisfying the characteristics of E (given by (17.10)), may be defined as

$$\begin{aligned} \mu_{pq} &= 1 - \tfrac{d_{pq}}{D}, \; if \; d_{pq} \le D, \\ &= 0, \qquad otherwise. \end{aligned} \tag{17.12}$$

d_{pq} is a distance measure which provides similarity (in terms of proximity) between the pth and qth patterns in the feature space. The higher the value of d_{pq}, the lower is the similarity between pth and qth patterns, and *vice versa*. D is a parameter which reflects the minimum separation between a pair of patterns belonging to two different clusters. When $d_{pq} = 0$ and $d_{pq} = D$, we have $\mu_{pq} = 1$ and 0, respectively. If $d_{pq} = \frac{D}{2}$, $\mu_{pq} = 0.5$. That is, when the distance between the patterns is just half the value of D, the difficulty in making a decision, whether both the patterns are in the same cluster or not, becomes maximum; thereby making the situation most ambiguous.

The term D (in (17.12)) may be expressed as

$$D = \beta d_{max}, \tag{17.13}$$

where d_{max} is the maximum separation between a pair of patterns in the entire feature space, and $0 < \beta \leq 1$ is a user defined constant. β determines the degree of flattening of the membership function (17.12). The higher the value of β, more will be the degree, and *vice-versa*.

The distance d_{pq} (in (17.12)) can be defined in many ways. Considering Euclidean distance, we have

$$\begin{aligned} d_{pq} &= [\sum_i w_i^2 (x_{pi} - x_{qi})^2]^{\frac{1}{2}}, \\ &= [\sum_i w_i^2 \chi_i^2]^{\frac{1}{2}}, \quad \chi_i = (x_{pi} - x_{qi}), \end{aligned} \tag{17.14}$$

where $w_i \in [0, 1]$ represents weighting coefficient corresponding to ith feature. The terms x_{pi} and x_{qi} are values of ith feature (in the corresponding feature space) of pth and qth patterns, respectively. d_{max} is defined as

$$d_{max} = [\sum_i (x_{maxi} - x_{mini})^2]^{\frac{1}{2}}, \tag{17.15}$$

where x_{maxi} and x_{mini} are the maximum and minimum values of the ith feature in the corresponding feature space.

As in Section 17.3.1, the membership value μ_{pq} is dependent on w_i. The values of w_i (< 1) make μ_{pq} in (17.12) flattened along the axis of d_{pq}. The lower the value of w_i, the higher is extent of flattening. In the extreme case, when $w_i = 0$, $\forall i$, $d_{pq} = 0$ and $\mu_{pq} = 1$ for all pair of patterns, *i.e.*, all the patterns lie on the same point making them indiscriminable.

The weight w_i (in (17.14)) reflects the relative importance of the feature x_i in measuring the similarity (in terms of distance) of a pair of patterns. The higher the value of w_i, the more is the importance of x_i in characterizing a cluster or discriminating various clusters. $w_i = 1(0)$ indicates most (least) importance of x_i.

Note that, one may define μ_{pq} in a different way satisfying the above mentioned characteristics. The computation of μ_{pq} in (17.12) does not require the information on class label of the patterns.

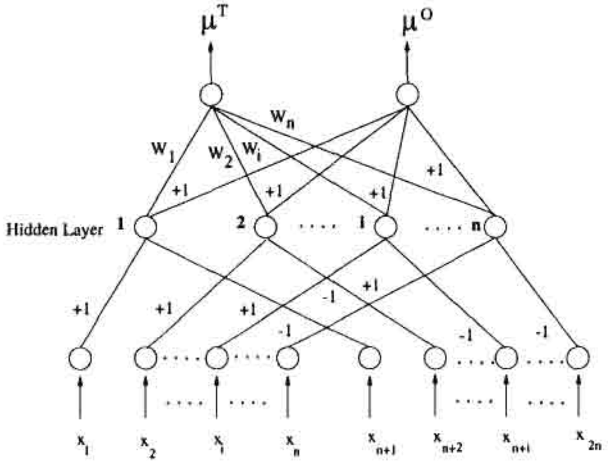


Fig. 17.2 A schematic diagram of the neural network model for unsupervised feature selection

As mentioned earlier, our objective is to minimize the evaluation index E (defined by (17.10)) which involves the terms μ^O and μ^T. The computation of μ^T requires (17.12)–(17.15), while μ^O needs these equations with $w_i = 1, \forall i$. Therefore, E is a function of $\mathbf{w}$, if we consider ranking of n features in a set. The problem of feature selection/ranking thus reduces to finding a set of w_is for which E becomes minimum; w_is indicating the relative importance of x_i's. The task of minimization is performed using gradient-descent technique in a connectionist framework under unsupervised mode. This is described below.

Connectionist model

The network (Fig. 17.2) consists of an input, a hidden and an output layer. The input layer consists of a pair of nodes corresponding to each feature, *i.e.*, the number of nodes in the input layer is $2n$, for n-dimensional (original) feature space. The hidden layer consists of n number of nodes which compute the part χ_i^2 of (17.14) for each pair of patterns. The output layer consists of two nodes. One of them computes μ^O, and the other μ^T. The feature evaluation index E (defined by (17.44)) is computed from these μ-values off the network.

Input nodes receive activations corresponding to feature values of each pair of patterns. A jth node in the hidden layer is connected only to an ith and $(i+n)$th input nodes via connection weights $+1$ and -1, respectively, where $j, i = 1, 2, \ldots, n$ and $j = i$. The output node computing μ^T-values is connected to a jth node in the hidden layer via connection weight W_j $(= w_j^2)$, whereas that computing μ^O-values is connected to all the nodes in the hidden layer via connection weights $+1$ each.

During learning, each pair of patterns are presented at the input layer and the evaluation index is computed. The weights W_js are updated using gradient-descent technique in order to minimize the index E. Note that, d_{max} is directly computed from the unlabeled training set. The values of d_{max} and β are stored in both the output nodes for the computation of D. The readers may refer to the Appendix for details concerning the operation of the network.

As mentioned before, the task of minimization of E (defined by (17.10)) with respect to $\mathbf{W}$ is performed using gradient-descent technique, where the change in W_j $(\triangle W_j)$ is computed as

$$\triangle W_j = -\eta \frac{\partial E}{\partial W_j}, \forall j, \tag{17.16}$$

where η is the learning rate. The term $\frac{\partial E}{\partial w_i}$, as in Section 17.3.1, is computed using various node activations and link weights of the network (see the Appendix).

Algorithm for learning $\mathbf{W}$

- Calculate d_{max} from the unlabeled training set and store it in both the output nodes. Store β (user specified) in both the output nodes.
- Initialize W_j with small random values in $[0, 1]$.
- Repeat until convergence, *i.e.*, until the value of E becomes less than or equal to certain predefined small quantity, or the number of iterations attains certain predefined value:
  - For each pair of patterns:
    - Present the pattern pair to the input layer.
    - Compute $\triangle W_j$ for each j using the updating rule in (17.16).
  - Update W_j for each j with $\triangle W_j$ averaged over all the patterns.

As in the case of neuro-fuzzy supervised feature selection method (Section 17.3.1), $E(\mathbf{W})$, after convergence, attains a local minimum. Then the weights $(W_j = w_j^2)$ of the links connecting hidden nodes and the output node computing μ^T-values, indicate the order of importance of the features. Note that this unsupervised method performs the task of feature selection without clustering the feature space explicitly and does not need to know the number of clusters present in the feature space.

17.4 Neuro-fuzzy knowledge-based classification

In this section, we describe a methodology [19] for encoding *a priori* initial domain knowledge in a connectionist model, particularly in a fuzzy MLP [28], under supervised learning. The concept is based on the fact that if a classifier is initially provided with some knowledge from the data set, the resulting searching space is reduced thereby leading to a more efficient learning. The architecture of the network may become simpler due to the inherent reduction of the redundancy among the connection weights. The network topology is then refined using the training data.

17.4.1 Input representation

An n-dimensional pattern $\mathrm{x_i} = [\mathrm{x_{i1}}, \mathrm{x_{i2}}, \ldots, \mathrm{x_{in}}]$ is represented as a $3n$-dimensional vector [27]

$$\mathbf{x}_\mathrm{i} = \left[\mu_{\mathrm{low}(\mathrm{x_{i1}})}(\mathbf{x}_\mathrm{i}), \mu_{\mathrm{medium}(\mathrm{x_{i1}})}(\mathbf{x}_\mathrm{i}), \mu_{\mathrm{high}(\mathrm{x_{i1}})}(\mathbf{x}_\mathrm{i}), \ldots, \mu_{\mathrm{high}(\mathrm{x_{in}})}(\mathbf{x}_\mathrm{i})\right], \tag{17.17}$$

where μ values indicate the membership functions of the corresponding linguistic π-sets (as in (17.4)) [25, 28] along each feature axis. The input can be in numeric, linguistic or set form and can have modifiers very, more or less (mol), or not, attached to it as described in [27]. We ensure that any feature value along the jth axis for pattern $\mathbf{x}_i$ is assigned membership value combinations in the corresponding 3-dimensional linguistic space of (17.17) in such a way that at least one of $\mu_{low(x_{ij})}(\mathbf{x}_\mathrm{i})$, $\mu_{medium(x_{ij})}(\mathbf{x}_\mathrm{i})$ or $\mu_{high(x_{ij})}(\mathbf{x}_\mathrm{i})$ is greater than 0.5. This heuristic ensures that each pattern point belongs positively to at least one of the linguistic sets low, medium or high along each feature axis.

17.4.2 Output representation

Consider an M-class problem domain such that we have M nodes in the output layer. The desired output ($d_k \in [0,1]$) of the kth output node for the ith input pattern is defined as [25]

$$d_k = \mu_k(\mathbf{x}_i) = \frac{1}{1 + \left(\frac{z_{ik}}{f_d}\right)^{f_e}}, \tag{17.18}$$

where $\mu_k(\mathbf{x}_i)$ is the membership value of the ith pattern in class C_k, z_{ik} is the weighted distance of the training pattern $\mathbf{x}_i$ from C_k, and the positive constants f_d and f_e are the denominational and exponential fuzzy generators controlling the amount of fuzziness in this class-membership set. They influence the amount of overlapping among the output classes. Note that, here we have used a (non-linguistic) definition of the output nodes which indicates the degree of belongingness of a pattern to a class. However, this definition may be suitably modified in other application areas to include linguistic definitions.

17.4.3 Knowledge encoding

Let an interval $[x_{j_1}, x_{j_2}]$ denote the range of feature x_j covered by class C_k. Then we denote the membership value of the interval as $\mu([x_{j_1}, x_{j_2}])$ $(= \mu(between\ x_{j_1}\ and\ x_{j_2}))$ and compute it as [27]

$$\mu(between\ x_{j_1}\ and\ x_{j_2}) = \{\mu(greater\ than\ x_{j_1}) * \mu(less\ than\ x_{j_2})\}^{\frac{1}{2}}, \tag{17.19}$$

where

$$\begin{aligned} \mu(greater\ than\ x_{j_1}) &= \{\mu(x_{j_1})\}^{\frac{1}{2}},\ if\ x_{j_1} \le c_{prop}, \\ &= \{\mu(x_{j_1})\}^2,\ otherwise, \end{aligned} \tag{17.20}$$

and

$$\begin{aligned} \mu(less\ than\ x_{j_2}) &= \{\mu(x_{j_2})\}^{\frac{1}{2}},\ if\ x_{j_2} \ge c_{prop}, \\ &= \{\mu(x_{j_2})\}^2,\ otherwise. \end{aligned} \tag{17.21}$$

Here c_{prop} denotes c_{j_l}, c_{j_m} and c_{j_h} for each of the corresponding three overlapping fuzzy sets *low*, *medium* and *high* as in [28]. The output membership for the corresponding class C_k is found using (17.18). Note that, for the computation of z_{ik} [25] of (17.18), x_{ij} is replaced by the mean of the interval $[x_{j_1}, x_{j_2}]$ of the jth feature.

We have also considered the intervals in which a class is *not* included. The complement of the interval $[x_{j_1}, x_{j_2}]$ of the feature x_j is the region where the class C_k does not lie and is defined as $[x_{j_1}, x_{j_2}]^c$ (where S^c denotes the complement of S). The linguistic membership values for $[x_{j_1}, x_{j_2}]^c$ is denoted by $\mu([x_{j_1}, x_{j_2}]^c)$ $(= \mu(not\ between\ x_{j_1}\ and\ x_{j_2}))$ and is calculated as

$$\mu(not\ between\ x_{j_1}\ and\ x_{j_2}) = \max\{\mu(less\ than\ x_{j_1}), \mu(greater\ than\ x_{j_2})\}, \tag{17.22}$$

since, $not\ between\ x_{j_1}\ and\ x_{j_2} \equiv less\ than\ x_{j_1}\ OR\ greater\ than\ x_{j_2}$.

Let the linguistic membership values for class C_k in the interval $[x_{j_1}, x_{j_2}]$, as calculated by (17.19)–(17.21), be $\{\mu_\ell([x_{j_1}, x_{j_2}]),\ \ell = L, M, H\}$. Similarly for the complement of the interval, using (17.22), we have

$$\{\mu_L([x_{j_1}, x_{j_2}]^c), \mu_M([x_{j_1}, x_{j_2}]^c), \mu_H([x_{j_1}, x_{j_2}]^c)\}.$$

A fuzzy MLP [28] with only one hidden layer is considered, taking two hidden nodes corresponding to $[x_{j_1}, x_{j_2}]$ and its complement respectively. Links are introduced between the input nodes and the corresponding nodes in the hidden layer

$$\text{if and only if} \quad \mu_A([x_{j_1}, x_{j_2}])\ or\ \mu_A([x_{j_1}, x_{j_2}]^c) \geq 0.5\ \forall j,$$

where $A \in \{L, M, H\}$. The weight $w^{(0)}_{k_{\alpha_p} j_m}$ between k_{α_p}th node of the hidden layer (the hidden node corresponding to the interval $[x_{j_1}, x_{j_2}]$ for class C_k) and j_mth $(m \in \{L, M, H\})$ node of the input layer corresponding to feature x_j is

$$w^{(0)}_{k_{\alpha_p} j_m} = p_k + \epsilon, \tag{17.23}$$

where p_k is the *a priori* probability of class C_k and ϵ is a small random number. This hidden node is designated as *positive* node. A second hidden node k_{α_n} is considered for the complement case and is termed as *negative* node. Its connection weights are initialized as

$$w^{(0)}_{k_{\alpha_n} j_m} = (1 - p_k) + \epsilon. \tag{17.24}$$

The small random number ϵ makes the weights asymmetric. Thus for an M-class problem, we have $2M$ nodes in the hidden layer. In our algorithm we have considered the following two cases:

- *All* connections between these $2M$ hidden nodes and all nodes in the input layer are possible. The other weights are initially set as small random numbers. We call this as model AN.
- Only those *selected* connection weights initialized by (17.19)–(17.24) are allowed. This is called model SN.

It is to be mentioned that the method described above can suitably handle convex pattern classes only. In the case of concave classes we consider multiple intervals for a feature x_j corresponding to the various convex partitions that may be generated to approximate the given concave decision region. This also holds for the complement of the region in x_j in which a particular class C_k is not included. Hence, in such cases we introduce hidden nodes, *positive* and *negative*, for each of the intervals with connections being established by (17.23) and (17.24). In this connection, it is to be noted that a concave class may also be subdivided into several convex regions as in [17].

Let there be $(k_{pos} + k_{neg})$ hidden nodes, where $k_{pos} = \sum_{\alpha_p} k_{\alpha_p}$ and $k_{neg} = \sum_{\alpha_n} k_{\alpha_n}$, generated for class C_k such that $k_{pos} \geq 1$ and $k_{neg} \geq 1$. Now connections are established between kth output node (for class C_k) and only the corresponding $(k_{pos} + k_{neg})$ hidden nodes. We assume that if any feature value (for class C_k) is outside some interval α, the total input received by the corresponding hidden node k_α is zero and therefore this produces an output $v_{k_\alpha}^{(1)} = 0.5$ due to the sigmoid nonlinearity.

The connection weight $w_{kk_\alpha}^{(1)}$ between the kth output node and the k_αth hidden node is calculated from a series of equations using activation values. For an interval α as input for class C_k, the expression for output $v_k^{(2)}$ of the kth output node is

$$v_k^{(2)} = f(v_{k_\alpha}^{(1)} w_{kk_\alpha}^{(1)} + \sum_{r \neq \alpha} 0.5 w_{kk_r}^{(1)}), \tag{17.25}$$

where $f(\cdot)$ is the sigmoid function. The hidden nodes k_r correspond to the intervals not represented by the convex partition α. Thus for a particular class C_k we have as many equations as the number of intervals (including *not*) used for approximating any concave and/or convex decision region C_k. Thereby, we can uniquely compute each of the connection weights $w_{kk_\alpha}^{(1)}$ $\forall\alpha$ (corresponding to each hidden node k_α and class C_k pair).

The network architecture, so encoded, is then refined by backpropagation training using the input pattern set under supervised learning. In case of model AN, all the link weights are trained. In case of model SN, only the selected link weights are trained, while the other connections are kept clamped at zero. If the network achieves satisfactory performance, the classifier design is complete. Otherwise, we resort to link pruning or node growing which are described in [19].

17.5 Results

The effectiveness of these methodologies along with comparisons has been demonstrated extensively on various real-life pattern recognition problems. Here we provide the results on two data sets— vowel [25] and Iris [10]. For detailed results on the other data sets, one may refer to [9, 19, 24].

The vowel data consists of a set of 871 Indian Telugu vowel sounds. These were uttered in a consonant-vowel-consonant context by three male speakers in the age group of 30 to 35 years. The data set has three features, F_1, F_2 and F_3 corresponding to the first, second and third vowel formant frequencies obtained through spectrum analysis of the speech data. Fig. 13.4 shows the overlapping nature of the six vowel classes (*viz.*, ∂, a, i, u, e, o) in the $F_1 - F_2$ plane (for ease of depiction). The details of the data and its extraction procedure are available in [25]. This vowel data is being extensively used for the past three decades in the area of pattern recognition.

Anderson's Iris data [10] set contains three classes, *i.e.*, three varieties of Iris flowers, namely, *Iris setosa* (IS), *Iris versicolor* (IV) and *Iris virginica* (IVir), consisting of 50 samples each. Each sample has four features, namely, Sepal Length (SL), Sepal Width (SW), Petal Length (PL) and Petal Width (PW). Iris data has been used in many research investigations related to pattern recognition and has become a sort of benchmark data.

17.5.1 On neuro-fuzzy feature selection

Tables 17.1 and 17.2 provide the degrees of importance (w) of individual features, corresponding to the vowel and Iris data. For the supervised method, the values of r_k in (17.5) are so chosen that the membership values of all the patterns of a class are at least 0.5 for that class. For 6-class vowel data the values of r_k were found to be 28.8, 78.5, 21.4, 74.0, 20.4 and 47.8 corresponding to its different classes. Similarly, these values are 71.7, 241.3 and 193.9 for 3-class Iris data. For the unsupervised method the value of β is set as 0.16 for vowel data and 0.3 for Iris data.

It is interesting to note from these tables that the order of importance obtained by both the supervised and unsupervised methods is the same for these two data sets. For vowel data, the order of importance of individual features is $F_2 > F_1 > F_3$ which is the same as obtained by some earlier investigations [21, 23]. ($x > y$ means feature x is more important than y.) For Iris data, PL and PW are found to be the best two features. This is also in conformity with

Table 17.1 Importance of different features of vowel data.

Feature	w-values and Rank			
	Supervised		Unsupervised	
	w	Rank	w	Rank
F_1	0.257358	2	0.590065	2
F_2	0.437536	1	0.896044	1
F_3	0.154319	3	0.120944	3

Table 17.2 Importance of different features of Iris data.

Feature	w-values and Rank			
	Supervised		Unsupervised	
	w	Rank	w	Rank
SL	0.203230	4	0.058414	4
SW	0.302529	3	0.194421	3
PL	0.422186	1	0.965575	1
PW	0.402027	2	0.603508	2

some earlier investigation [37]. In order to establish these results, we consider scatter plots and k-NN classifier. Here, results are given only on Iris data. It is again evident from Figs. 17.3-17.8 and the results of the k-NN classifier [9, 24] that $\{PL, PW\}$ is the best feature pair. Between PL and PW, it is difficult to find the edge of one over the other.

17.5.2 On neuro-fuzzy knowledge-based classification

Here we demonstrate the effectiveness of the neuro-fuzzy knowledge-based model on the vowel data. The variables f_d and f_e of (17.18) were set at 5.0 [28]. The entire data set has been divided into *training set* and *test set*.

Table 17.3 shows the results obtained with vowel data. Since all the classes in the feature space are convex, we use two hidden nodes for each of the classes. Hence we require a total of 12 hidden nodes for this data set. The results demonstrate that model AN gives acceptably good performance in just

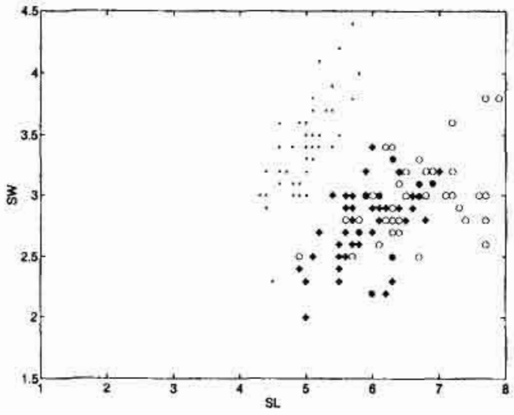


Fig. 17.3 Scatter plot $SL - SW$ of Iris data. Here '.', '+' and 'o' represent classes *Iris setosa*, *Iris versicolor* and *Iris virginica*, respectively

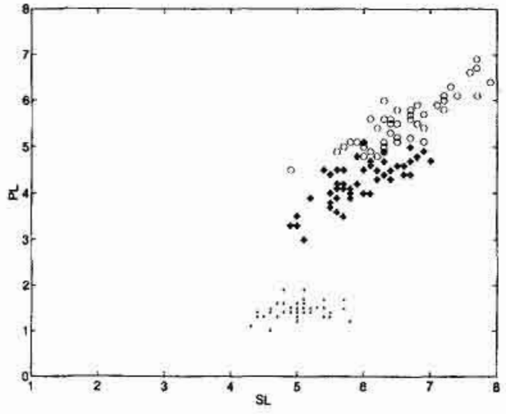


Fig. 17.4 Scatter plot $SL - PL$ of Iris data. Here '.', '+' and 'o' represent classes *Iris setosa*, *Iris versicolor* and *Iris virginica*, respectively

200 epochs whereas model SN cannot do so due to fewer links. Since the vowel classes are overlapping and need more information for their proper partitioning, model SN could not perform well as compared to model AN. For details on the comparative results, one may refer to [19].

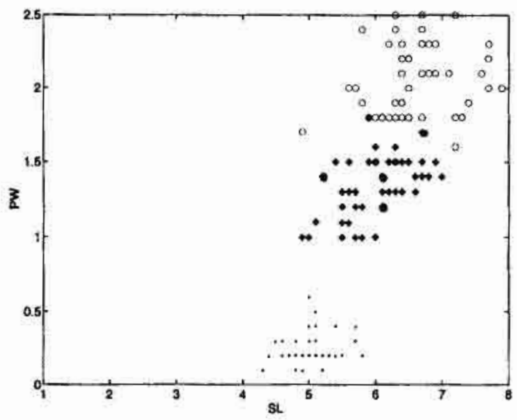


Fig. 17.5 Scatter plot $SL - PW$ of Iris data. Here '.', '+' and 'o' represent classes *Iris setosa*, *Iris versicolor* and *Iris virginica*, respectively

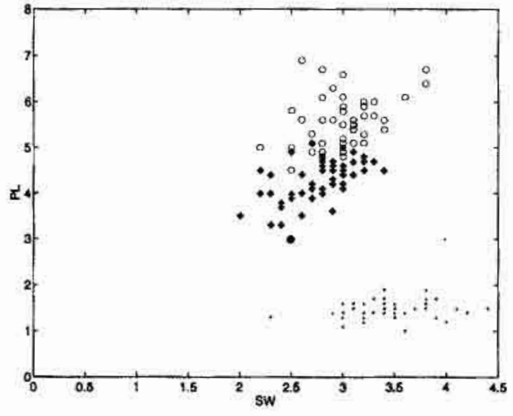


Fig. 17.6 Scatter plot $SW - PL$ of Iris data. Here '.', '+' and 'o' represent classes *Iris setosa*, *Iris versicolor* and *Iris virginica*, respectively

17.6 Conclusions and Discussion

We have shown here how neuro-fuzzy integration can be exploited for developing methodologies for feature selection and classification. Various connectionist

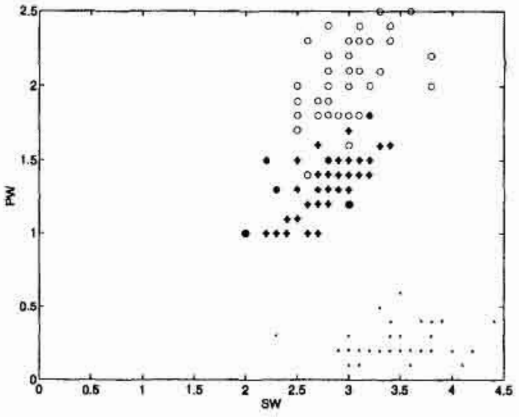


Fig. 17.7 Scatter plot $SW - PW$ of Iris data. Here '.', '+' and 'o' represent classes *Iris setosa*, *Iris versicolor* and *Iris virginica*, respectively

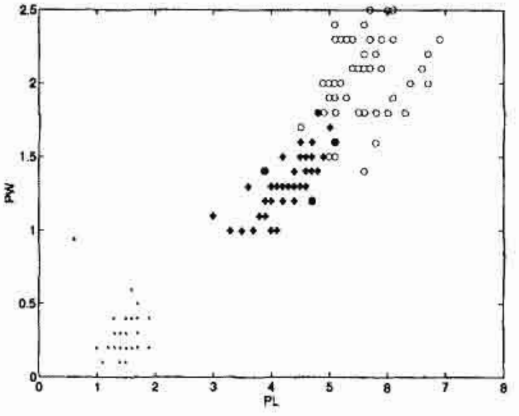


Fig. 17.8 Scatter plot $PL - PW$ of Iris data. Here '.', '+' and 'o' represent classes *Iris setosa*, *Iris versicolor* and *Iris virginica*, respectively

models have been designed. Feature selection algorithms assume interdependencies of the features. The unsupervised model does not need to know the number of clusters. The incorporation of fuzziness at various levels of the knowledge-based networks helps in modeling uncertainty in both input repre-

Table 17.3 Classification performance of knowledge-based models on vowel data

Model	Class	Score(%)	
		Training	Testing
AN	∂	42.86	27.69
	a	87.5	86.42
	i	94.12	87.74
	u	100.0	82.35
	e	90.0	69.52
	o	100.0	93.83
	Overall	90.59	78.63
SN	∂	0.0	0.0
	a	62.5	58.02
	i	94.12	87.74
	u	100.0	82.35
	e	85.0	68.45
	o	94.44	93.21
	Overall	82.35	73.79

sentation and output decision.

Individual feature ranking, as obtained by the neuro-fuzzy feature selection methods, conforms well to those obtained using other methods [21, 23, 37]. This ranking is also validated with respect to classification performance/clustering ability (using k-NN classifier and fuzzy c-means clustering algorithm), and class/cluster structures (using scatter plots) [9, 24]. The knowledge-based system learns faster than other related models, and provides superior recognition score.

References

[1] M. Banerjee, S. Mitra, and S. K. Pal, "Rough fuzzy MLP: Knowledge encoding and classification," *IEEE Trans. on Neural Networks*, vol. 9, pp. 1203–1216, 1998.

[2] J. Basak, R. K. De, and S. K. Pal, "Unsupervised feature selection

using neuro-fuzzy approach," *Pattern Recognition Letters*, vol. 19, pp. 997–1006, 1998.

[3] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Trans. on Neural Networks*, vol. 5, pp. 537–550, 1994.

[4] K. W. Bauer, Jr., S. G. Alsing, and K. A. Greene, "Feature screening using signal-to-noise ratios," *Neurocomputing*, vol. 31, pp. 29–44, 2000.

[5] L. M. Belue and K. W. Bauer, Jr., "Determining input features for multilayer perceptrons," *Neurocomputing*, vol. 7, pp. 111–121, 1995.

[6] J. C. Bezdek and S. K. Pal, eds., *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data*. New York: IEEE Press, 1992.

[7] G. Castellano and A. M. Fanelli, "Variable selection using neural-network models," *Neurocomputing*, vol. 31, pp. 1–13, 2000.

[8] R. K. De, N. R. Pal, and S. K. Pal, "Feature analysis : Neural network and fuzzy set theoretic approaches," *Pattern Recognition*, vol. 30, pp. 1579–1590, 1997.

[9] R. K. De, J. Basak, and S. K. Pal, "Neuro-fuzzy feature evaluation with theoretical analysis," *Neural Networks*, vol. 12, pp. 1429–1455, 1999.

[10] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

[11] L. M. Fu, "Knowledge-based connectionism for revising domain theories," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23, pp. 173–182, 1993.

[12] K. Hirota and W. Pedrycz, "Knowledge-based networks in classification problems," *Fuzzy Sets and Systems*, vol. 59, pp. 271–279, 1993.

[13] A. Kowalczyk and H. L. Ferra, "Developing higher-order neural networks with empirically selected units," *IEEE Trans. on Neural Networks*, vol. 5, pp. 698–711, 1994.

[14] P. van de Laar and T. Heskes, "Input selection based on an ensemble," *Neurocomputing*, vol. 34, pp. 227–238, 1995.

[15] B. F. Leao and A. F. Rocha, "Proposed methodology for knowledge acquisition: A study on congenital heart disease diagnosis," *Methods of Information in Medicine*, vol. 29, pp. 30–40, 1990.

[16] R. J. Machado and A. F. Rocha, "A hybrid architecture for connectionist expert systems," in *Intelligent Hybrid Systems* (A. Kandel and

G. Langholz, eds.), Boca Raton: CRC Press, 1992.

[17] D. P. Mandal, C. A. Murthy, and S. K. Pal, "Determining the shape of a pattern class from sampled points in $\mathcal{R}^2$," *International Journal of General Systems*, vol. 20, pp. 307–339, 1992.

[18] R. Masuoka, N. Watanabe, A. Kawamura, Y. Owada, and K. Asakawa, "Neurofuzzy system – fuzzy inference using a structured neural network," in *Proceedings of the 1990 International Conference on Fuzzy Logic and Neural Networks, Iizuka,* (Japan), pp. 173–177, 1990.

[19] S. Mitra, R. K. De, and S. K. Pal, "Knowledge-based fuzzy MLP for classification and rule generation," *IEEE Trans. on Neural Networks*, vol. 8, pp. 1338–1350, 1997.

[20] S. Mitra, P. Mitra, and S. K. Pal, "Evolutionary modular design of rough knowledge-based network using fuzzy attributes," *Neurocomputing*, vol. 36, pp. 45–66, 2001.

[21] S. K. Pal, "Fuzzy set theoretic measures for automatic feature evaluation: II," *Information Sciences*, vol. 64, pp. 165–179, 1992.

[22] S. K. Pal, J. Basak, and R. K. De, "Fuzzy feature evaluation index and connectionist realization," *Information Sciences*, vol. 105, pp. 173–188, 1998.

[23] S. K. Pal and B. Chakraborty, "Fuzzy set theoretic measures for automatic feature evaluation," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 16, pp. 754–760, 1986.

[24] S. K. Pal, R. K. De, and J. Basak, "Unsupervised feature evaluation: A neuro-fuzzy approach," *IEEE Trans. on Neural Networks*, vol. 11, pp. 366–376, 2000.

[25] S. K. Pal and D. Dutta Majumder, *Fuzzy Mathematical Approach to Pattern Recognition.* New York: John Wiley (Halsted Press), 1986.

[26] S. K. Pal, A. Ghosh, and M. K. Kundu, eds., *Soft Computing for Image Processing.* Heidelberg: Physica Verlag, 2000.

[27] S. K. Pal and D. P. Mandal, "Linguistic recognition system based on approximate reasoning," *Information Sciences*, vol. 61, pp. 135–161, 1992.

[28] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets and classification," *IEEE Trans. on Neural Network*, vol. 3, pp. 683–697, 1992.

[29] S. K. Pal and S. Mitra, *Neuro-fuzzy Pattern Recognition: Methods in Soft Computing.* New York: John Wiley, 1999.

[30] S. K. Pal and P. K. Pramanik, "Fuzzy measures in determining seed

points in clustering," *Pattern Recognition Letters*, vol. 4, pp. 159–164, 1986.

[31] S. K. Pal and A. Skowron, eds., *Rough Fuzzy Hybridization: A New Trend in Decision-Making*. Singapore: Springer Verlag, 1999.

[32] S. K. Pal and P. P. Wang, eds., *Genetic Algorithms for Pattern Recognition*. Boca Raton: CRC Press, 1996.

[33] W. Pedrycz and A. F. Rocha, "Fuzzy-set based models of neurons and knowledge-based networks," *IEEE Trans. on Fuzzy Systems*, vol. 1, pp. 254–266, 1993.

[34] K. L. Priddy, S. K. Rogers, D. W. Ruck, G. L. Tarr, and M. Kabrisky, "Bayesian selection of important features for feedforward neural networks," *Neurocomputing*, vol. 5, pp. 91–103, 1993.

[35] D. W. Ruck, S. K. Rogers, and M. Kabrisky, "Feature selection using a multilayer perceptron," *Journal of Neural Network Computing*, vol. 20, pp. 40–48, 1990.

[36] R. Setino and H. Liu, "Neural-network feature selector," *IEEE Trans. on Neural Networks*, vol. 8, pp. 654–662, 1997.

[37] J. M. Steppe and K. W. Bauer, Jr., "Improved feature screening in feedforward neural networks," *Neurocomputing*, vol. 13, pp. 47–58, 1996. pp. 40–48, Fall 1990.

[38] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, pp. 119–165, 1994.

Appendix

Operation of the Supervised Neural Network Model for Feature Selection

When the kth auxiliary node of the network is activated, input node i has an activation value

$$u_{ik} = (I_{ik})^{r_k}, \tag{17.26}$$

where I_{ik} is the total activation received by the ith input node for the pattern $\mathbf{x}$, when the auxiliary node k is active. I_{ik} is given by

$$I_{ik} = x_i - m_{ki}. \tag{17.27}$$

x_i is the external input (value of the ith feature for the pattern $\mathbf{x}$) and $-m_{ki}$ is the feedback activation from the kth auxiliary node to the ith input node. The activation value of the kth output node is given by

$$v_k = g(y_k), \tag{17.28}$$

where $g(\cdot)$, the activation function of each output node, is a π-function as given in (17.4). y_k, the total activation received by the kth output node for the pattern $\mathbf{x}$, is given by

$$y_k = \left(\sum_i u_{ik} \times (\frac{w_i}{\lambda_{ki}})^2\right)^{\frac{1}{r_k}}. \tag{17.29}$$

Note that y_k is the same as d_k (in (17.5)) for the given input pattern $\mathbf{x}$, and v_k is equal to the membership value of the input pattern $\mathbf{x}$ in class C_k.

The expression for $E(\mathrm{w})$ (from (17.1)), in terms of the output node activations, is given by

$$E(\mathrm{w}) = \sum_k \sum_{\mathbf{x}\in C_k} \frac{v_k(1-v_k)}{\sum_{k'\neq k} \frac{1}{2}\left[v_k(1-v_{k'}) + v_{k'}(1-v_k)\right]} \times \alpha_k. \tag{17.30}$$

The training phase of the network takes care of the task of minimization of $E(\mathrm{w})$ with respect to w which is performed using simple gradient-descent technique. The change in w_i (Δw_i) is computed as

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i, \tag{17.31}$$

where η is the learning rate.

For the computation of $\frac{\partial E}{\partial w_i}$, the following expressions are used.

$$\frac{\partial H_{kk'}(\mathbf{x})}{\partial w_i} = \frac{1}{2}\left[[1-2v_{k'}]\frac{\partial v_k}{\partial w_i} + [1-2v_k]\frac{\partial v_{k'}}{\partial w_i}\right], \tag{17.32}$$

$$\frac{\partial H_k(\mathbf{x})}{\partial w_i} = [1-2v_k]\frac{\partial v_k}{\partial w_i}, \tag{17.33}$$

$$\begin{aligned}\frac{\partial v_k}{\partial w_i} &= -4d_k(\mathbf{x})\frac{\partial d_k(\mathbf{x})}{\partial w_i}, & 0 \le d_k(\mathbf{x}) < \tfrac{1}{2},\\ &= -4[1-d_k(\mathbf{x})]\frac{\partial d_k(\mathbf{x})}{\partial w_i}, & \tfrac{1}{2} \le d_k(\mathbf{x}) < 1,\\ &= 0, & otherwise,\end{aligned} \tag{17.34}$$

and

$$\frac{\partial d_k(\mathbf{x})}{\partial w_i} = \left(\frac{w_i}{d_k(\mathbf{x})}\right)^{r_k-1}\left(\frac{x_i-m_{ki}}{\lambda_{ki}}\right)^{r_k}. \tag{17.35}$$

Alternately, we can also express E as a function of W_{ki}, where $W_{ki} = (\frac{w_i}{\lambda_{ki}})^{r_k}$, and then minimize E with respect to W_{ki}. In this case, during training phase, the values of W_{ki}s can be updated using the same gradient-descent technique. After training, the degree of importance of ith feature can be computed as $w_i = W_{ki}^{\frac{1}{r_k}} \times \lambda_{ki}$.

Operation of the unsupervised neural network model for feature selection

When pth and qth patterns are presented to the input layer, the activation produced by ith $(1 \le i \le 2n)$ input node is

$$v_i^{(0)} = u_i^{(0)}, \tag{17.36}$$

where

$$\begin{aligned}u_i^{(0)} &= x_{pi},\ for\ 1 \le i \le n \ \text{ and}\\ u_{i+n}^{(0)} &= x_{qi},\ for\ 1 \le i \le n\end{aligned} \tag{17.37}$$

represent the total activations received by ith and $(i+n)$th $(1 \le i \le n)$ input node, respectively. The total activation received by jth hidden node (connecting ith and $(i+n)$th, $1 \le i \le n$, input nodes) is given by

$$u_j^{(1)} = 1 \times v_i^{(0)} + (-1) \times v_{i+n}^{(0)},\quad for\ 1 \le i \le n, \tag{17.38}$$

and the activation produced by it is

$$v_j^{(1)} = (u_j^{(1)})^2. \tag{17.39}$$

The total activation received by the output node which computes μ^T-values, is

$$u_T^{(2)} = \sum_j W_j v_j^{(1)}, \tag{17.40}$$

and that received by the other output node which computes μ^O-values, is

$$u_O^{(2)} = \sum_j v_j^{(1)}. \tag{17.41}$$

Therefore, $u_T^{(2)}$ and $u_O^{(2)}$ represent d_{pq}^2 as given by (17.14) where $0 \le w_i < 1$ and $w_i = 1$, $\forall i$, respectively. The activations, $v_T^{(2)}$ and $v_O^{(2)}$, of the output nodes represent μ_{pq}^T and μ_{pq}^O for pth and qth pattern pair, respectively. Thus,

$$\begin{aligned} v_T^{(2)} &= 1 - \tfrac{(u_T^{(2)})^{\frac{1}{2}}}{D}, \ \ if\ (u_T^{(2)})^{\frac{1}{2}} \le D, \\ &= 0, \qquad\qquad otherwise, \end{aligned} \tag{17.42}$$

and

$$\begin{aligned} v_O^{(2)} &= 1 - \tfrac{(u_O^{(2)})^{\frac{1}{2}}}{D}, \ \ if\ (u_O^{(2)})^{\frac{1}{2}} \le D, \\ &= 0, \qquad\qquad otherwise. \end{aligned} \tag{17.43}$$

The evaluation index (which is computed off the network), in terms of these activations, is then written (from (17.10)) as

$$E(\mathbf{W}) = \frac{2}{s(s-1)} \sum_p \sum_{q \ne p} \frac{1}{2} [v_T^{(2)}(1 - v_O^{(2)}) + v_O^{(2)}(1 - v_T^{(2)})]. \tag{17.44}$$

The task of minimization of $E(\mathbf{W})$ with respect to $\mathbf{W}$ is performed using gradient-descent technique, where the change in W_j (ΔW_j) is computed as

$$\Delta W_j = -\eta \frac{\partial E}{\partial W_j}, \forall j. \tag{17.45}$$

Here η is the learning rate.

For computation of $\frac{\partial E}{\partial W_j}$ corresponding to a pair of patterns, the following expressions are used.

$$\frac{\partial E(\mathbf{W})}{\partial W_j} = \frac{1}{2}\left[1 - 2v_O^{(2)}\right]\frac{\partial v_T^{(2)}}{\partial W_j}, \tag{17.46}$$

$$\begin{aligned} \frac{\partial v_T^{(2)}}{\partial W_j} &= -\frac{\frac{1}{2}(u_T^{(2)})^{-\frac{1}{2}}\frac{\partial u_T^{(2)}}{\partial W_j}}{D}, \quad if\ (u_T^{(2)})^{\frac{1}{2}} \leq D, \\ &= 0, \qquad\qquad\qquad\quad otherwise, \end{aligned} \tag{17.47}$$

and

$$\frac{\partial u_T^{(2)}}{\partial W_j} = v_j^{(1)}. \tag{17.48}$$

Chapter 18

ADAPTIVE SEGMENTATION TECHNIQUES FOR HYPERSPECTRAL IMAGERY

H. Kwon, S. Z. Der and N. M. Nasrabadi

U. S. Army Research Laboratory
Adelphi, MD 20783-1197, U.S.A
e-mail: *heesung_kwon@yahoo.com*
{sder,nnasraba}@arl.army.mil

Abstract

The authors have presented a supervised and an unsupervised segmentation algorithm that adapt to the local characteristics of hyperspectral imagery. Both use an iterative method in which the images are first coarsely segmented, followed by more refined segmentation in successive iterations. Local adaptation is satisfactorily incorporated into the successive iterations using local feature extraction and the segmentation results of the previous iteration. A modified cost function, that incorporates a spatial smoothness term into a spectral distance measure, is used in both algorithms. In the supervised technique, the quadtree-based segmentation used is effective in representing the underlying spectral structure of the hyperspectral images, resulting in lower computational cost. In the unsupervised technique, the iterative use of a local spectral dissimilarity measure provides a set of values that can discriminate among different materials. The unsupervised approach proves to be superior to other unsupervised algorithms, particularly for complex hyperspectral scenes containing mixtures of a large number of different materials.

18.1 Introduction

As the demand for digital battlefield technologies grows, automatic target recognition (ATR) techniques have been increasingly used. ATR systems often consist of two main algorithmic components – target segmentation (or detection) and identification. Since target identification depends on preceding segmentation results, overall performance relies on segmentation performance. Hyperspectral remote sensing techniques have been widely applied in military applications (*e.g.*, ATR), environmental monitoring, atmospheric and space applications.

Segmentation of hyperspectral imagery is based on discrimination between materials according to the spectral signature in thermal emission [2, 14, 16] or solar reflection [5, 9]. A hyperspectral image region filled with the same material type tends to show strong spectral similarity, while different materials exhibit differences in spectral reflectivity, especially in certain regions of the spectrum [9]. The local spatial relationships of the neighboring pixels in hyperspectral images also play an important role in segmentation when they are effectively combined with the spectral information. Accordingly, hyperspectral (or multispectral) segmentation algorithms that simultaneously utilize both spectral and spatial characteristics of objects, achieved better segmentation or detection performance [1, 3, 7, 8, 9, 17].

Adaptive statistical classifiers that maximize the *a posteriori* probability density, given the distribution of classes, have been frequently used to effectively detect the spectral differences [1] and the intensity differences in single-band images [10]. The techniques iteratively estimate the parameters (mean and covariance) of the class-conditional densities to adapt to the local characteristics of the image, under the assumption that each class has a different Gaussian distribution. However, the computational cost of these methods becomes very high as the image size and spectral dimensionality increase. This limits their practical use for hyperspectral applications. There is also no guarantee that the spectral distribution of each class follows a Gaussian distribution. Therefore, there has been a strong demand for an effective technique that can provide both improved segmentation performance and less computational cost.

We introduce an adaptive segmentation technique for hyperspectral imagery based on an iterative method, in which segmentation at a given iteration depends closely on the segmentation results at the previous iteration [11, 12]. The hyperspectral images are first coarsely segmented and then the segmentation is successively refined using recursive local feature extraction based on either quadtree decomposition or a sliding window-based method [8]. During the segmentation process, each cluster center (mean vector that serves as a spectral template for a material type) is recursively updated over a local image region based on the segmentation results of the previous iteration. Local adaptation is desirable because spectral differences in illumination or atmospheric attenuation as well as material variation can cause pixels of the same material type to display different spectral signatures. The method does not require knowledge of the probability distribution of each material type, and substantially reduces the computational cost because its decision logic is much simpler. Quadtree decomposition has been frequently used to represent the underlying structure of the various forms of digital data [13]. In the quadtree decomposition, the hyperspectral images are adaptively decomposed into the multiscaled subregions in which the spectral data are approximately homogeneous. A local feature vector (*i.e.*, a material centroid) and its corresponding cost function for each material type (*e.g.*, the target and the vegetation) are recursively estimated and updated over the decomposed regions, so that increasingly accurate segmentation results are achieved as decomposition proceeds. In the sliding window-based method, for each pixel the feature vector of each material type is estimated over the sliding window that is centered on the corresponding input pixel, whose size is reduced progressively during the segmentation process. Since the sliding window-based technique is a pixel-based approach, it provides better segmentation performance than the quadtree-based method, at the expense of higher computational cost. However, the computational cost of the sliding window-based method is still much less than that of adaptive statistical classifiers. The cost function of both methods consists of a spatial constraint term and the Euclidean distance between each pixel spectrum and the material centroid. The spatial constraint term is used to penalize pixels that are classified differently from their neighbors in the previous iteration, providing spatial continuity throughout the segmentation process.

We also introduce an unsupervised segmentation technique, in which the spectral characteristics of material types do not need to be known *a priori*. The unsupervised technique is designed to apply to hyperspectral scenes with a large number of different material types.

18.2 Hyperspectral imaging system

In the hyperspectral imaging system that we deal with, the image of a tactical scene that contains military targets and background vegetation is first taken by an imaging spectrometer at a spectral range of 460 to 1000 nm with a step size of 10 or 20 nm. The imaging spectrometer then splits the light reflected from the scene according to frequency bands, and produces 55/28 spatially registered image bands, as shown in Fig. 18.1. We call this collection of bands, in order of decreasing spectral band frequencies, a hyperspectral cube (see Fig. 18.2). Two or four different polarizations are used for each spectral band, doubling or quadrupling the number of images per band. Accordingly, each image in the hyperspectral cube corresponds to one specific frequency band in the whole spectral range. Using the spectral band values along the spectral domain for each pixel in the scene, one can form a spectral band-value curve. Fig. 18.3 shows an example of the spectral band-value curve along with its band information.

18.3 Segmentation of hyperspectral imagery

In this section, we briefly describe two frequently used segmentation techniques for hyperspectral imagery, which are a template-matching technique and a statistical classifier.

18.3.1 Template-matching technique

The template-matching technique is based on a spectral distance measure between a prototype vector (*i.e.*, a template) of each material type and an input pattern. This simple method has low computational cost because each pixel spectrum is considered independently of its neighbors. The prototype vector f_i is created by calculating the mean of a set of sample patterns that belongs to the corresponding material type. It is then used as a template for the minimum-distance classification. A cost function C_i for material i is estimated for each input pattern, and the corresponding input pattern is classified into the material with the lowest cost function. The cost function C_i is defined as

$$C_i = \|x - f_i\|, \tag{18.1}$$

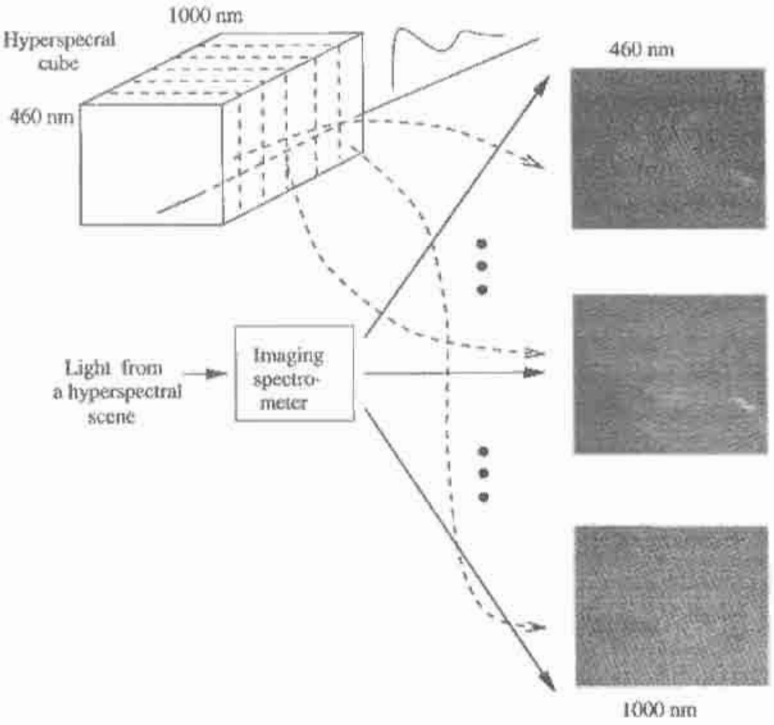


Fig. 18.1 Hyperspectral imaging system

where $\|\cdot\|$ represents the Euclidean norm, and x represents the input vector. Fig. 18.5 shows a two-class template-matching example for the hyperspectral images.

Since the mean vector uses the global mean, and thus does not adjust to spatial variation of the spectral signature of each material type, the template-matching technique generally produces unsatisfactory segmentation performance. Improvements can be made by incorporating a local adaptation process, in which the prototype vector is gradually updated over a local region whose size is reduced during the segmentation process. The adaptive technique is introduced in Section 18.4.

18.3.2 Statistical classifiers

A statistical classifier is based on a probabilistic approach, in which an input pattern is classified to a particular material type with the minimum classification errors [4]. By the Bayes theorem, the *a posteriori* probability density $p(m_i|x)$, the probability of the input pattern x relative to the material type m_i, can be expressed as

$$p(m_i|x) \propto p(x|m_i)P(m_i) \qquad i = 1, 2, \cdots, K, \tag{18.2}$$

where $p(x|m_i)$ represents the conditional probability of the input pattern given the distribution of the material type m_i, $P(m_i)$ represents the *a priori* density, and K represents the number of different material types. A statistical classifier that classifies the input pattern into the material type with the maximum $p(m_i|x)$ is called a Bayes classifier. In the Bayes classifier, both the distribution of each material type and $P(m_i)$ must be known *a priori*. In most practical applications, the conditional probability is obtained under the assumption that each material type has a different Gaussian distribution. Accordingly, the conditional probability associated with the input vector can be easily calculated by identifying the mean and covariance of the corresponding material type. The statistical approach based on the Bayes classifier is usually superior to the non-adaptive template-matching technique in terms of segmentation performance, while it requires much higher computational cost.

18.4 Adaptive segmentation based on iterative local feature extraction

18.4.1 Initial segmentation

The template-matching technique based on a minimum-distance classifier [9] is used to obtain an initial segmentation. For each material type, we first create a representative spectral band-value curve (a mean curve) by averaging the spectral band-value curves of the training pixels for that material. The spectral band-value curves are obtained from several square windows manually extracted from the images, as shown in Fig. 18.2. Several training hyperspectral cubes are used in this process. The spectral band-value curves are normalized by their means to reduce variation in the pixel curves caused by differences in illumination. For each material, we create a prototype feature vector from its representative curve by selecting and concatenating only the spectral band

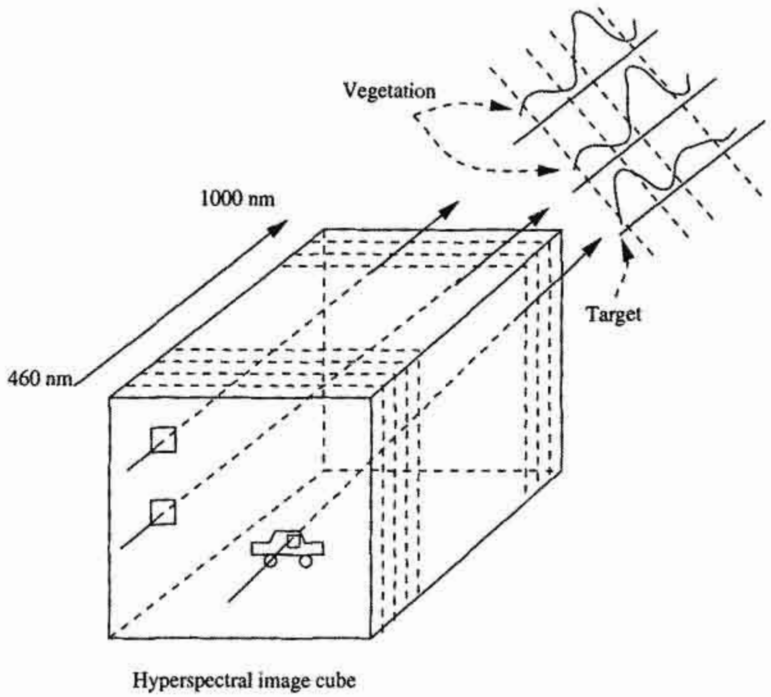


Fig. 18.2 Creation of the spectral band-value curves for the target and the vegetation regions in a hyperspectral cube

areas that give substantial discrimination among different materials. Use of the prototype feature vectors created from the selected band areas further reduces the computational cost without affecting segmentation performance. Fig. 18.4 shows the representative spectral band-value curves used for the target and the vegetation materials, and the prototype feature vectors inside the box areas. Using the prototype feature vectors, we perform segmentation for the hyperspectral cubes. For each pixel in the scene, its corresponding spectral band-value curve is tested against the prototype feature vectors of different materials. Each pixel in the scene is classified as the particular material that gives minimum distance between that material's feature vector and its corresponding test curve.

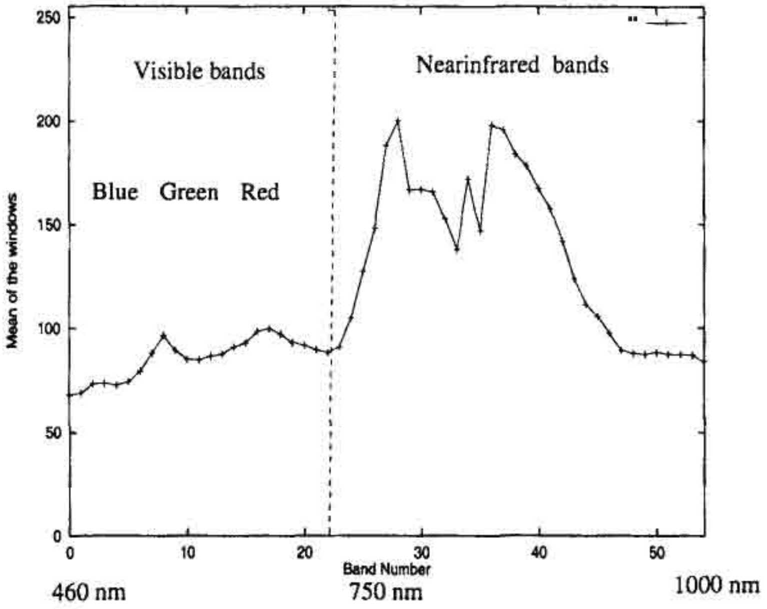


Fig. 18.3 An example of the spectral band-value curve, along with band information

18.4.2 Quadtree-based segmentation

The quadtree-based segmentation technique is based on quadtree decomposition and the use of a modified minimum-distance classifier on the nodes of a quadtree structure. The nonadaptive method described in Section 18.4.1 provides the prototype feature vectors that serve as the initial feature vectors of the materials (*i.e.*, the initial centroids of the target and vegetation regions), and the classification results serve as the initial segmentation. The algorithm recursively decomposes the hyperspectral images until they are divided into multiscaled rectangular regions in which the spectral band-value curves are similar to one another. The typical quadtree structure for a hyperspectral cube is shown in Fig. 18.6. Each time a node X^m is decomposed into four equal quadrants, X_i^{m-1}, $i = 1, 2, 3, 4$, the algorithm updates the local feature vector of each material type and re-estimates the corresponding cost function for each quadrant. The above process is repeated until the decomposition process ends. The cost function consists of (i) a Euclidean distance term, measured

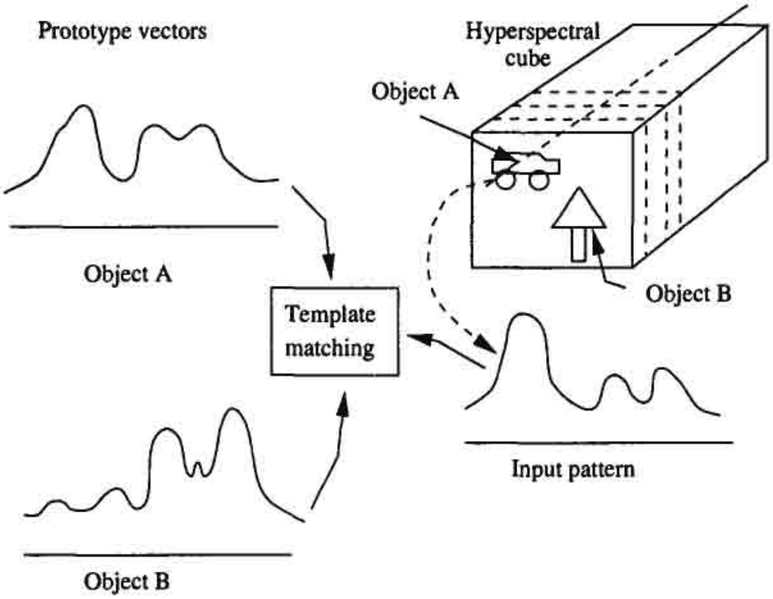


Fig. 18.4 Two-class template-matching example

between the feature vector of each material type and the input vector and (ii) a spatial similarity term. The spectral band values of the selected band areas are averaged for each material to form the feature vector. The spatial constraint term is used to impose a spatial smoothness constraint on the pixel classification. The cost function for material j at pixel location c of the node X^m is

$$C_m^j = \|x_c - f_m^j\| + \sum_N V_N(x), \tag{18.3}$$

where $\|\cdot\|$ represents the Euclidean norm, x_c and f_m^j represent the input vector at pixel location c and the feature vector of material j in the corresponding node, respectively, and $\sum_N V_N(x)$ represents the spatial constraint term. $V_N(x)$ is defined as

$$V_N(x) = \begin{cases} -\alpha & \text{if } \omega_q = j \text{ and } q \in N \\ +\alpha & \text{if } \omega_q \neq j \text{ and } q \in N, \end{cases}$$

where α represents a positive value, ω_q represents segmentation of the pixel at location q, and N represents the neighborhood of the pixel at c. If the neighboring pixel was classified as material j in the previous segmentation for the parent of the current node, the cost function decreases by α; otherwise it increases by α. The feature vector f_m^j is the result of averaging the corresponding spectral-band values of the pixels of material j within the corresponding block. For each pixel in the corresponding node, the cost function for each material type is calculated, and the corresponding pixel is classified into the material with the lowest cost function. As the decomposition process proceeds, the feature vector gradually adapts to the local spectral contents of the local region. The decomposition and segmentation process stops if the number of pixels that are classified differently from the classification in the parent node is lower than a threshold, or the predefined smallest size of the block is reached. Fig. 18.7 shows how the algorithm works. Fig. 18.8 shows the performance improvement of the quadtree-based method over the nonadaptive method [9] obtained on our data set. The target area is decomposed into small blocks because of the spectral difference from the neighboring vegetation area.

18.4.3 Sliding window-based segmentation

A sliding window-based segmentation method is an adaptive segmentation technique that iteratively extract a local feature over the local region; the algorithm is a modified version of the previous adaptive algorithm based on the statistical classifiers [1, 10]. The algorithm is designed to provide better segmentation performance. The algorithm is designed to provide better segmentation performance than that of quadtree decomposition, mainly due to its pixel-based approach, at the expense of higher computational cost. Like the quadtree decomposition method, the initial feature vectors and the classification results are obtained from the nonadaptive method described in Section 18.4.1.

For each pixel, the window-based algorithm iteratively updates the local feature vector of each material type and measures the corresponding cost function for classification. It uses the same cost function as that of quadtree decomposition. The cost function for material j at the pixel location c is

$$C_j = \|x_c - f_c^j\| + \sum_N V_N(x), \tag{18.4}$$

where x_c and f_c^j represent the input vector and the feature vector of the mate-

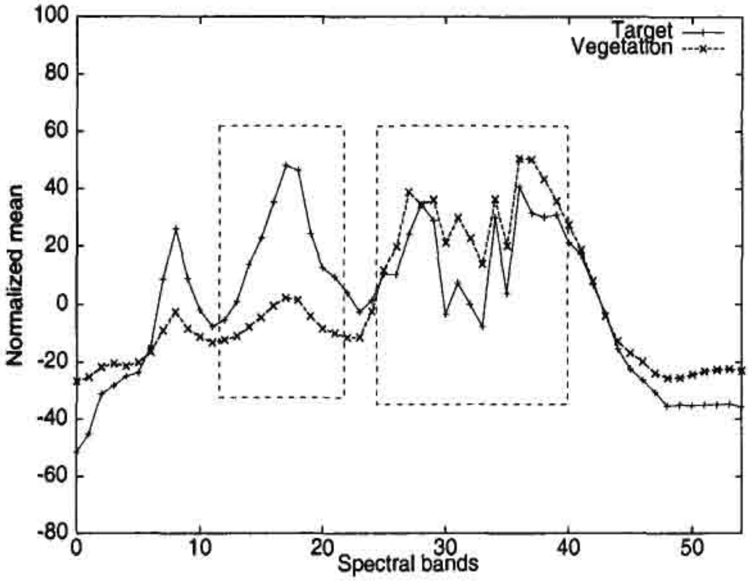


Fig. 18.5 The normalized representative spectral band-value curves for the target and vegetation regions. The prototype feature vectors inside the box areas give substantial band-value difference

rial j over the corresponding sliding window, respectively, and $\sum_N V_N(x)$ represents the same spatial constraint term as that of the quadtree-based method. For each pixel, the feature vector of each material is estimated over a *sliding* window that is centered on the corresponding input pixel, whose size is reduced progressively to adapt to the local details. The feature vector f_c^j is obtained by averaging the spectral-band values of the pixels of material j within the sliding window, as shown in Fig. 18.9. The method alternates between updating the local feature vector f^j and obtaining the segmentation $\hat{\omega}$. Fig. 18.10 shows how the window-based algorithm works. At the initial segmentation, the size of the sliding window is the same as the size of the whole image, so the material feature vectors are globally identical. The iterative classification proceeds for a fixed window size, during which the cost functions and the corresponding segmentation are repeatedly re-estimated at each iteration. The iteration stops if the number of pixels that are classified differently than in the previous iteration is lower than a threshold, or the predefined maximum number of it-

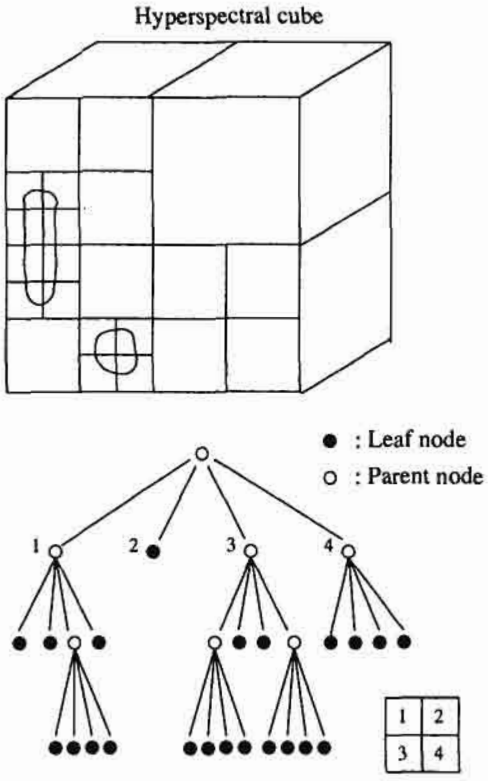


Fig. 18.6 The quadtree data structure of a hyperspectral cube

erations is reached. After the segmentation stabilizes, the size of the sliding window decreases by a factor of two and the feature vector of each type of material for the corresponding window is re-estimated for each pixel according to the local pixels in each image. Thus, each pixel has a different set of feature vectors during every cycle except the first. The same iterative procedure as that of the initial segmentation is applied. If the stop criterion is met, the sliding window size is further reduced. This adaptive process continues until the size of the sliding window reaches its smallest size. Fig. 18.11 shows the performance improvement of the window-based method over the nonadaptive method [9] obtained on our data set. The window-based method more accurately segments the target (an HMMWV) area due to its ability to exploit the

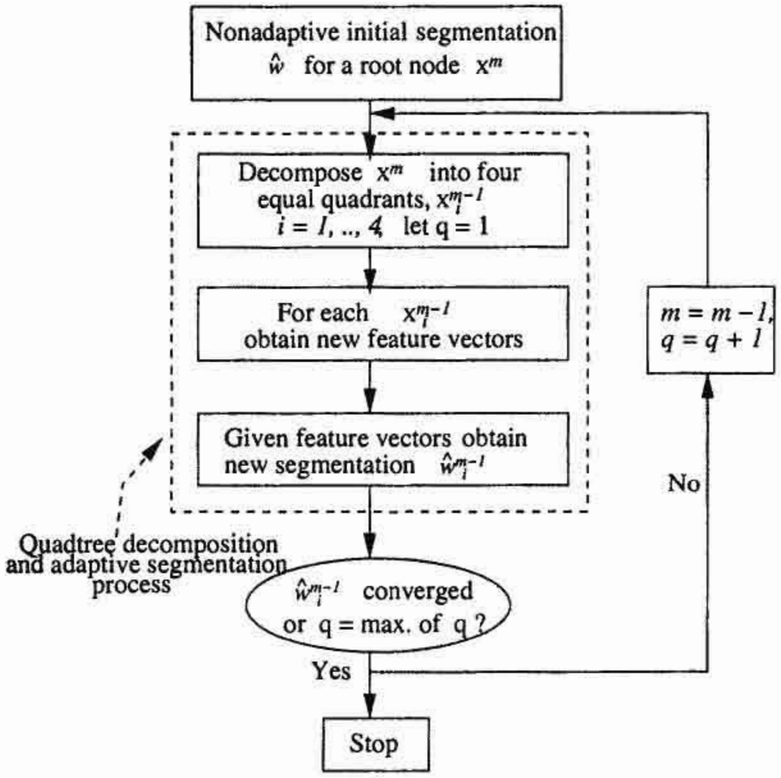

Nonadaptive initial segmentation $\hat{w}$ for a root node x^m
Decompose x^m into four equal quadrants, x_i^{m-1} $i = 1, .., 4$ let q = 1
For each x_i^{m-1} obtain new feature vectors
Given feature vectors obtain new segmentation $\hat{w}_i^{m-1}$
$m = m - 1$, $q = q + 1$
No
Quadtree decomposition and adaptive segmentation process
$\hat{w}_i^{m-1}$ converged or q = max. of q ?
Yes
Stop

Fig. 18.7 The quadtree-based adaptive segmentation algorithm

local details.

18.4.4 Simulation results

We used three tactical hyperspectral cubes as training data to obtain the initial feature vectors for the target and the vegetation in the initial segmentation process. The band areas used to create the prototype feature vectors from the spectral band-value curves were from the 11th band through the 39th band; a total of 29 bands out of 55 bands were used. The size of the tactical scene to be segmented was 640×480. The smallest block size of the quadtree decomposition used was 40×30. For the window-based method, we used the

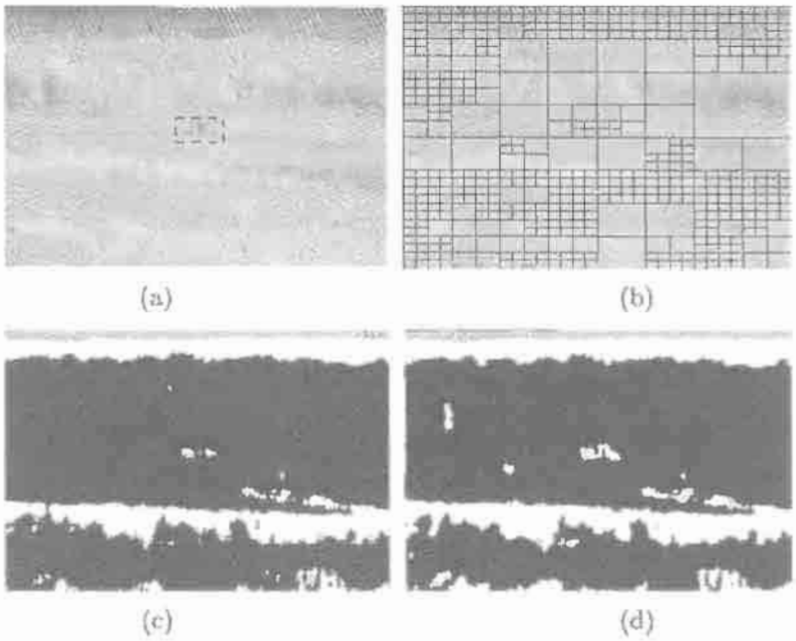


Fig. 18.8 Segmentation results: (a) 32nd band image from cube 1 (target is inside the dotted box area), (b) quadtree structure of cube 1, (c) nonadaptive method, and (d) quadtree-based method. Note that the entire target area belongs to the smallest decomposed blocks

variable-size sliding windows for estimating the local feature vectors, and the smallest sliding-window size was 40×30. The parameter α for the spatial constraint term used for both methods was 5. The four neighboring pixels were used to impose the spatial constraint. The stopping threshold was 1% of the total number of pixels in the image. The maximum number of iterations for each sliding window size was 5 in the window-based method. We satisfactorily segmented more than 10 hyperspectral cubes using both methods, and they provided much improved segmentation performance over the nonadaptive method for all the cubes tested. The quantitative performance results for both methods are presented in Table 18.1 for the four cubes that contain small targets; the two adaptive methods showed dramatic improvement over the nonadaptive method since they are suitable for the detection of small

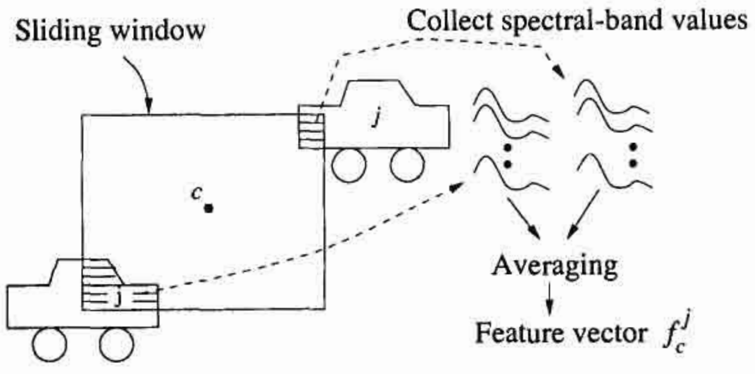


Fig. 18.9 Feature vector estimation based on variable-size sliding window

targets due to their powerful local adaptivity. The window-based method provided better segmentation performance than that of quadtree decomposition mainly due to its pixel-based approach at the expense of higher computational cost. Figs. 18.8 and 18.11 show the segmentation results of the nonadaptive method [9] and the corresponding adaptive segmentation method on one of the cubes. Both adaptive methods provide a better segmentation of the target. For cube 2 in Table 18.1, the nonadaptive method barely recognizes the target area, while the adaptive methods capture almost the entire target area. The average running time of the quadtree-based and window-based algorithm was about 10 and 30 minutes, respectively, on a Sun Sparc-20 machine, while the adaptive statistical technique took a lot more time to complete. However, the major improvements of the window-based algorithm were made in the early iterations with relatively large window sizes before the smallest window was used, as can be seen in Table 18.1. Accordingly, we can further reduce the running time of the window-based algorithm and maintain reasonable segmentation performance by increasing the smallest window size (*e.g.*, it took less than 15 minutes when the smallest window size was 160×120).

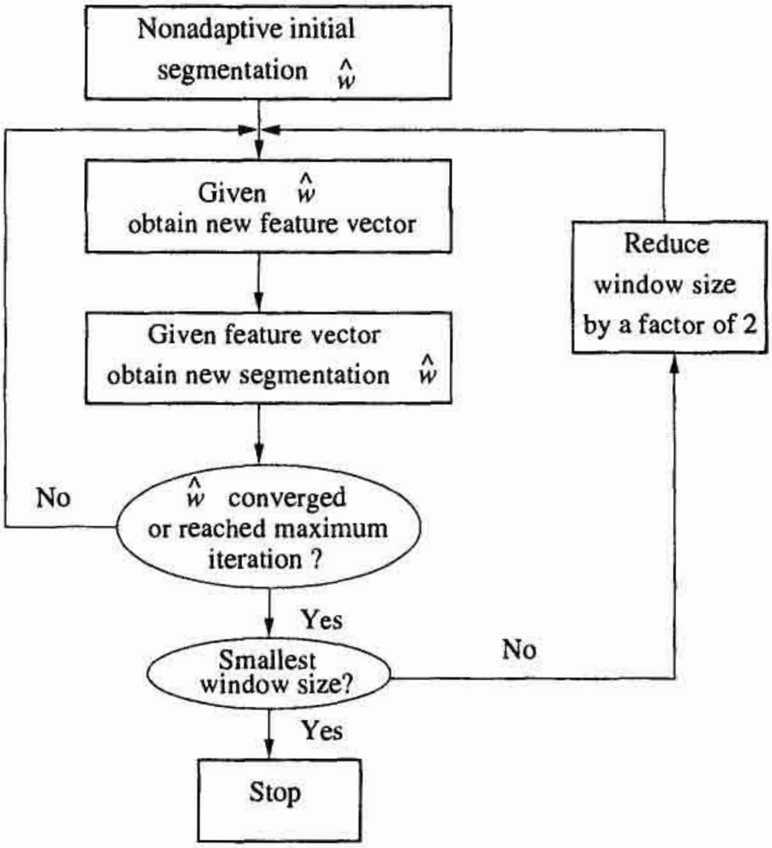
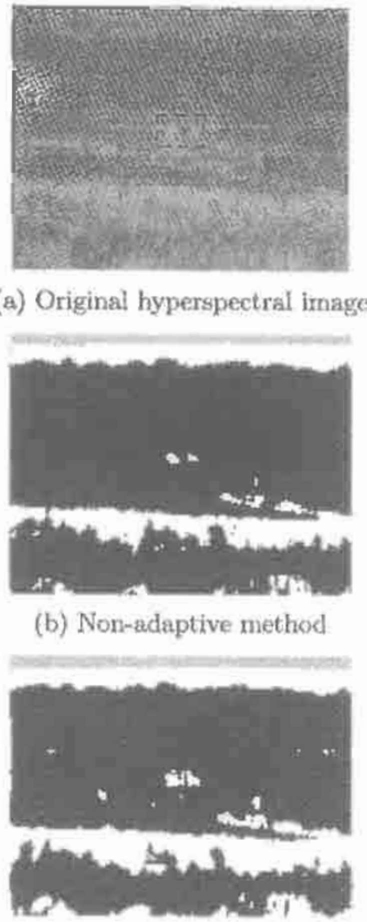

Nonadaptive initial segmentation $\hat{w}$
Given $\hat{w}$ obtain new feature vector
Given feature vector obtain new segmentation $\hat{w}$
$\hat{w}$ converged or reached maximum iteration ?
No
Yes
Smallest window size?
No
Reduce window size by a factor of 2
Yes
Stop

Fig. 18.10 The sliding window-based adaptive segmentation algorithm

18.5 Adaptive unsupervised segmentation

18.5.1 Introduction

Most previous segmentation algorithms (such as, minimum-distance classifiers based on a template-matching technique [9, 8], adaptive statistical classifiers [1, 10]) were based on supervised methods, which require the *a priori* knowledge of the spectral characteristics for each material type. As the applications of hyperspectral remote sensing techniques increase, more material types tend to be included in complex hyperspectral scenes. Accordingly, it is often difficult



(a) Original hyperspectral image

(b) Non-adaptive method

(c) Adaptive method

Fig. 18.11 Performance comparison between the nonadaptive method and the window-based adaptive method using test cube. Target is inside the dotted box area in the original image

or impractical to obtain *a priori* knowledge of the spectral characteristics for all the material types. This difficulty suggests the use of unsupervised segmentation techniques, where spectral features are obtained and processed without using the *a priori* knowledge of the spectral characteristics. Recently, an un-

Table 18.1 The segmentation performance of the nonadaptive method and the two adaptive methods for the target region. Each segmentation result for the sliding window-based method was obtained using the corresponding sliding window ranged from 640 × 480 through 40 × 30. As the sliding window size decreases, segmentation performance increases

	Cube 1	Cube 2	Cube 3	Cube 4
Nonadaptive method (%)	45.26	3.85	72.89	54.95
quadtree-based method (%)	76.84	82.88	93.11	80.01
Window based method (%)	75.00	19.23	79.11	62.61
	80.53	71.43	89.78	72.79
	81.05	92.31	95.56	77.93
	81.32	96.15	96.89	82.88
	81.53	98.35	97.78	83.78

supervised segmentation algorithm, which used VQ-based initial segmentation and the subsequent iterative maximum *a posteriori* (MAP) estimation, was developed for hyperspectral imagery with a relatively small number of spectral bands [15]. However, the computational cost of the above method becomes very high as the image size and spectral dimensionality increase. The practical use of the unsupervised segmentation algorithm based on a probabilistic approach is limited for hyperspectral applications with high spectral dimensionality.

18.5.2 Unsupervised segmentation based on a spectral dissimilarity measure

An effective unsupervised texture segmentation technique based on a dissimilarity measure using local Gabor coefficients was used in [6]. We present an adaptive unsupervised segmentation algorithm that provides a suitable dissimilarity measure for segmenting hyperspectral images, with reduced computational complexity and improved segmentation performance. The algorithm discriminates among material types using an iterative spectral dissimilarity measure of the local spectral difference between materials. The algorithm consists of (i) the initial segmentation based on a fixed spectral dissimilarity measure and the k-means algorithm, and (ii) the subsequent adaptive segmentation based on the iterative spectral dissimilarity measure over a local region whose size is reduced progressively.

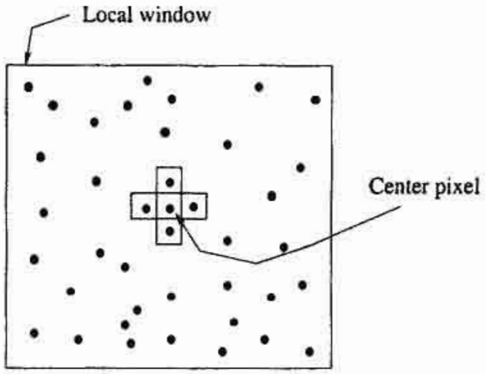


Fig. 18.12 Randomly selected pixels and neighboring pixels

In the initial segmentation, for each pixel in the hyperspectral image, a relatively large fixed size local window is placed around the pixel. Note that each pixel corresponds to a spectral band-value vector in the spectral domain. The spectral dissimilarity d_i associated with pixel location i is defined as

$$d_i = \frac{\sum_{j \in B} \|S_j - S_i\|}{N_i}, \tag{18.5}$$

where $\|\cdot\|$ represents the Euclidean norm, B represents a set of randomly selected pixels and the neighboring pixels within the local window, as shown in Fig. 18.12, S_i and S_j represent the corresponding spectral band-value vectors of the pixel at i and j, respectively, and N_i represents the number of pixels selected (the randomly selected pixels and neighboring pixels) to estimate d_i. After the spectral dissimilarity of every pixel is obtained, a spectral dissimilarity matrix D is formed. Each element of the matrix represents the degree of the local spectral difference between the corresponding pixel and its neighboring and randomly selected pixels within the local window. The spectral dissimilarity matrix provides a set of spectral-feature values quite suitable for initial clustering of the pixels, as shown in Fig. 18.13(b); the pixels within the same material type tend to have similar values, while the values differ between different material types. Initial segmentation is then performed by applying the k-means algorithm to the matrix. Fig. 18.13(c) shows the initial segmentation results.

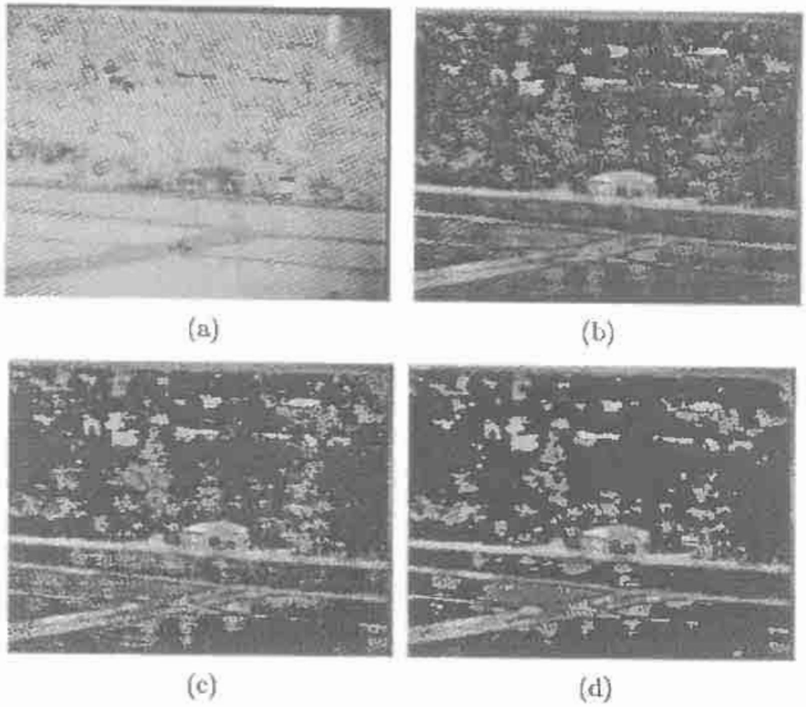


Fig. 18.13 Unsupervised segmentation results: (a) one of original hyperspectral images taken at 740 nm, (b) spectral feature (spectral dissimilarity matrix) for initial segmentation associated with the hyperspectral images to be segmented, (c) initial segmentation results using k-means on the spectral dissimilarity matrix, and (d) segmentation results corresponding to the smallest sliding window

A subsequent adaptive segmentation technique iteratively updates the local spectral dissimilarity d_i in a local window and measures the corresponding cost function for classification of each pixel. The size of the window is progressively reduced during the segmentation process; the window size starts from half of that used in the initial segmentation. As the window size is reduced, each element of the dissimilarity matrix gradually represents the corresponding local spectral characteristics; therefore, increasingly accurate segmentation results are obtained. The cost function consists of (i) the local spectral difference term relative to the current pixel spectrum, and (ii) a spectral similarity term. The spectral similarity term imposes a spatial smoothness constraint during the segmentation process. The cost function for material k at pixel location i is

$$C_i^k = \frac{\sum_j (d_j^k - d_i)}{N^k} + \sum_{\mathcal{N}} V_{\mathcal{N}}(i), \tag{18.6}$$

where d_j^k and N^k represent the corresponding spectral dissimilarity of the pixel at j and the number of pixels in the region classified as material k, respectively, and $\sum_{\mathcal{N}} V_{\mathcal{N}}(i)$ represents the spatial constraint term. $V_{\mathcal{N}}(i)$ is defined as

$$V_{\mathcal{N}}(i) = \begin{cases} -\alpha & \text{if } \omega_q = k \text{ and } q \in \mathcal{N} \\ +\alpha & \text{if } \omega_q \neq k \text{ and } q \in \mathcal{N}, \end{cases}$$

where α represents a positive value, ω_q represents segmentation of the pixel at location q, and N represents the neighborhood of the pixel at i. If the neighboring pixel was classified as material k in the segmentation of the previous iteration, the cost function decreases by α, otherwise it increases by α.

The cost function represents the average local spectral difference between the pixel to be classified and the pixels in the local region classified as material k. For each material type, the corresponding cost function is estimated, and the pixel is then classified as the material with the lowest cost function; this process assigns the input pixel into the material type with the most similar spectral characteristics. Local adaptation of the proposed algorithm is obtained by updating the cost function for each material type in the local region that is defined by a sliding window whose size is progressively reduced by a factor of two. The iterative classification proceeds for a fixed window size, during which the cost functions and the corresponding segmentation are repeatedly re-estimated at each iteration. The iteration stops if the number of pixels that are classified differently than in the previous iteration is lower than a threshold, or the predefined maximum number of iterations is reached. After the segmentation stabilizes, the size of the sliding window is decreased by a factor of two. This adaptive process continues until the size of the sliding window reaches its smallest size.

18.5.3 Simulation results

Five-level segmentation is used when applying the k-means algorithm to the initial segmentation results. 80 randomly selected pixels and four neighboring pixels of the corresponding input pixel in a local sliding window were used to estimate local spectral dissimilarity. The smallest sliding-window size was 40×30. The parameter α for the spatial constraint term used was 5. Several hyperspectral images were segmented satisfactorily in an unsupervised manner, showing superiority over existing unsupervised techniques. Fig. 18.13 shows the material segmentation performed by the algorithm; note that the small airplane is clearly distinguished from the road.

18.6 Conclusions

We have presented a supervised and an unsupervised segmentation algorithm that adapt to the local characteristics of hyperspectral imagery. Both algorithms used an iterative method in which the hyperspectral images were first coarsely segmented and then the segmentation is refined in successive iterations. Local adaptation was satisfactorily incorporated into the successive iterations using local feature extraction and the segmentation results of the previous iteration. A modified cost function, that incorporated a spatial smoothness term into a spectral distance (or difference) measure, was used in both algorithms.

In the supervised technique, the quadtree-based segmentation proved to be an effective method to represent the underlying spectral structure of the hyperspectral images, providing less computational cost. Better segmentation performance was obtained by sliding window-based segmentation, mainly because of the pixel-based local adaptation, at the expense of higher computational cost. However, the sliding window-based segmentation technique still had much lower computational cost than adaptive statistical classifiers. The choice between the two methods depends on the size and dimensionality of the imagery used.

In the unsupervised technique, the iterative use of a local spectral dissimilarity measure provided a set of values that can discriminate among different materials. The unsupervised segmentation technique proved to be superior to other unsupervised algorithms especially when a large number of different materials are mixed in complex hyperspectral scenes.

References

[1] E. L. Ashton, "Detection of subpixel anomalies in multispectral infrared imagery using an adaptive Bayesian classifier," *IEEE Transactions on Image Processing*, vol. 36, pp. 506–517, 1998.

[2] J. Cheung, D. Ferries, and L. Kurz, "On classification of multispectral infrared data," *IEEE Transactions on Image Processing*, vol. 6, pp. 1456–1460, 1997.

[3] J. O. Eklundh, H. Yamamoto, and A. Rosenfeld, "A relaxation method for multispectral pixel classification," *IEEE Transactions of Pattern Analysis and Machine Intelligence*, vol. 1, pp. 72–75, 1980.

[4] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. New York: Addison Wesley, 1993.

[5] M. Gottlieb, L. Denes, B. Kaminsky, P. Metes, and N. Gupta, "Hyperspectral and polarization imaging using an atof imager," in *Proceedings of the 3rd Annual ARL Fedlab Symposium*, (College Park, MD), pp. 23–27, Feb. 1999.

[6] T. Hofmann, J. Puzicha, and J. M. Buhmann, "An optimized approach to unsupervised hierarchical texture segmentation," in *Proceedings of IEEE International Conference on Image Processing*, (Santa Barbara, CA), 1997.

[7] R. L. Kettig and D. A. Landgrebe, "Classification of multispectral image data by extraction and classification of homogeneous objects," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 14, pp. 19–26, 1976.

[8] H. Kwon, S. Z. Der, and N. M. Nasrabadi, "An adaptive hierarchical segmentation based on quadtree decomposition for hyperspectral imagery," in *Proceedings of IEEE Conference on Image Processing*, (Vancouver, Canada), 2000.

[9] H. Kwon, S. Z. Der, N. M. Nasrabadi, and H. Moon, "Use of hyperspectral imagery in material classification in outdoor scenes," in *Proceedings of the SPIE*, vol. 3804, (Denver, CO), pp. 104–115, 1999.

[10] T. N. Pappas, "An adaptive clustering algorithm for image segmentation," *IEEE Transactions on Signal Processing*, vol. 40, pp. 901–914, 1992.

[11] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxation operation," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, pp. 420–433, 1976.

[12] A. Rosenfeld, "Iterative methods in image analysis," *Pattern Recognition*, vol. 10, pp. 181–187, 1978.

[13] H. Samet, "The quadtree and related hierarchical data structure," *ACM Computing Surveys*, vol. 16, pp. 188–260, 1984.

[14] C. R. Schwartz, M. T. Eismann, and J. N. Cederquist, "Thermal multispectral detection of military vehicles in vegetated and desert background," in *Proceedings of the SPIE*, vol. 2742, (Orlando, FL), pp. 286–297, Apr. 1996.

[15] Y. Tatsuya and D. Gingras, "Unsupervised multispectral image classification using MRF models and VQ method," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, pp. 1173–1176, 1999.

[16] B. Thai and G. Healey, "Invariant subpixel material identification in hyperspectral imagery," in *Proceedings of the DARPA Image Understanding Workshop*, pp. 809–814, 1998.

[17] T. Watanabe, H. Suzuki, and R. Yokoyama, "Improved contextual classifiers of multispectral image data," *IEICE Transactions on Fundamentals of Electrical Communication and Computer Science*, vol. 77, pp. 1445–1450, 1994.

Chapter 19

PATTERN RECOGNITION ISSUES IN SPEECH PROCESSING

B. Yegnanarayana and C. Chandra Sekhar

Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai-600036, INDIA
e-mail: *yegna@iitm.ernet.in*

Abstract

The authors have discussed the issues involved in the processing of the speech signal for performing pattern recognition tasks normally required for speech processing. They describe some of the currently available pattern recognition models used in the context of speech. Some challenging problems are also addressed.

19.1 Introduction

Speech is used primarily as a mode of communication for exchange of messages or information among human beings. Since we are endowed with both speech production and perception mechanisms, we do not realize the sophistication of the pattern generation process of the speech production mechanism and of the pattern perception mechanism of the auditory processing system. We begin to realize this sophistication, and our lack of understanding of it, when we try to replace a human being by a machine for generation of speech and/or for recognition of speech.

Speech signal carries with it information about the intended message, besides the characteristics and the state of the speaker, and also the characteristics of the language. Moreover, the signal is usually modified or corrupted by the environment, and the characteristics of the channel (telephone, microphone, cellular phone) used for transmission and recording of the signal. It appears as though the speech production process, including the language involved, is well matched with the speech perception process. Adequate redundancy is incorporated in the signal to take care of the likely degradations due to environmental conditions and channel effects.

The pattern recognition tasks that a human being performs in the context of speech are summarized in Table 19.1. The objective in this chapter is to discuss the issues involved in processing the speech signal for these tasks, and describe some of the currently available pattern recognition models used in the context of speech.

It is useful to understand why the speech tasks listed in Table 19.1 qualify to be pattern recognition tasks. Typically, in any pattern recognition task the samples used to capture the common pattern characteristics are not repeated. However, the pattern is still present in a new sample, and is recognized by our natural pattern recognition processing mechanism. In fact, the main difference between human and machine intelligence comes from the fact that human beings perceive everything as a *pattern*, whereas for a machine everything is *data* [7, 24]. Even in a routine data consisting of integer numbers (like telephone numbers, bank account numbers, car numbers), human beings tend to perceive a pattern. Recalling the data is also normally from a stored pattern. If there is no pattern, then it is difficult for a human being to remember and reproduce the data later. Thus, storage and recall operations in human beings and machines are performed by different mechanisms. The pattern nature in storage and recall automatically gives robustness and fault tolerance for the human system. Moreover, typically far fewer patterns than the expected capacity of the human memory system are stored.

Functionally also human beings and machines differ in the sense that human beings *understand* patterns, whereas machines can be said to *recognize* patterns in data. In other words, human beings can get the whole object in the data even though there is no clear identification of subpatterns in the data. For example, consider the name of a person written in handwritten cursive script. Even though the individual pattern for each letter may not be evident, the name is understood due to the visual hints provided in the written script. Likewise, speech is understood even though the patterns corresponding to the individual sounds may be distorted, sometimes to unrecognizable extents [3]. Another major characteristic of human pattern processing mechanism is its ability to continuously *learn* from examples, which is not understood well enough to implement it in an algorithmic fashion in a machine.

Table 19.1 Pattern recognition tasks in the context of speech. The objectives, types and methods for each task are given in the table

Task	Type	Objective	Issues	Method
Speech recognition	Isolated word recognition	Match templates of different lengths to determine the word	Medium size vocabulary; All words are important; Begin-end detection	Dynamic Time Warping
	Connected word recognition	Determine the number of words and the actual words in a sequence of words	Small size vocabulary; All words are important; Absence of word boundaries	Two-level dynamic programming
	Keyword spotting	Determine if a particular keyword occurs in the given speech	Detection of unknown words; Rejection of out-of-set utterances	Hidden Markov Models (HMM)
	Continuous speech recognition	Determine the text corresponding to the given speech	Choice of subword units; Contextual effects; Pronunciation dictionary	HMMs; Neural networks; Hybrid models
	Dialogue speech recognition	Understand ill-formed sentences	Dialogue modeling; Machine learning of dialogue	Stochastic language model adaptation

Human beings are capable of making mental patterns in their biological neural network from an input data, given in the form of numbers, text, picture, sounds, *etc.*, using their sensory mechanisms of vision, sound, touch, smell and taste. The patterns are also formed from a temporal sequence of data as in the case of speech and video. Human beings have the ability to recall the stored patterns even when the information is noisy or partial or mixed with information pertaining to other patterns.

Methods for solving pattern recognition tasks generally assume a sequential model for pattern recognition process, consisting of pattern environment, sensors to collect data from the environment, feature extraction from data and pattern association/storage/classification/grouping using the features [10, 11, 14]. The simplest solution to a pattern recognition problem is to use template matching, where the data of the test pattern is matched point by point with the corresponding data in the reference pattern. Obviously, this can work only for simple and highly restricted pattern recognition tasks. At the next level of complexity, one can assume a deterministic model for the pattern generation process, and derive the parameters of the model from a given pattern in order to represent the information in the pattern. Matching the test and reference patterns is done at the parametric level. This works well when the model of the pattern generation process is known with reasonable accuracy. One could also assume a stochastic model for the pattern generation process, and derive the parameters of the model from a large set of training patterns. Matching the test and reference patterns can be performed by several statistical methods such as likelihood ratio, variance weighted distance and Bayesian classification. Other approaches for pattern recognition tasks depend on extracting features from parameters or data. These features may be specific for the task. A pattern is described in terms of features, and pattern matching is done using descriptions in terms of these features. Another method based on descriptions is called syntactic pattern recognition, in which a pattern is expressed in terms of primitives suitable for the classes of patterns under study. Pattern matching is performed by matching the descriptions of the patterns in terms of the primitives. Methods based on the knowledge of the pattern generating source have also been explored for pattern recognition tasks. These knowledge-based systems express the knowledge in the form of rules for generating and perceiving the patterns. Different approaches commonly used for general pattern recognition tasks are summarized in Table 19.2.

The main difficulty in each of the aforesaid pattern recognition techniques is that of choosing an appropriate model for the pattern generating process, and then estimating the parameters of the model in the case of a model-based approach, *or* extraction of features from the data/parameters in the case of feature-based methods, *or* selecting appropriate primitives in the case of syntactic pattern recognition, *or* deriving rules in the case of a knowledge-based approach. The pattern recognition is all the more difficult when the test patterns are noisy or distorted versions of the patterns used in the training process. The ultimate goal is to impart to a machine the pattern recognition capabilities similar to those of human beings. This goal is difficult to achieve using many of the conventional methods, because, as mentioned earlier, these methods assume a sequential model for the pattern recognition process [1, 5, 20]. On the other hand, the human pattern recognition process is an integrated process involving the use of biological neural processing even from the stage of sensing the environment. Thus the neural processing takes place directly on the data for feature extraction, selective attention and pattern matching. Moreover, the large size (in terms of number of neurons and interconnections) of the biological neural network and the inherently different mechanism of processing may be contributing to our abilities of pattern recognition in spite of variability and noise, and also to our abilities to deal with the temporal patterns as well as with the so called stability-plasticity dilemma [24].

19.2 Nature of speech signal

Speech is a result of excitation of a time-varying vocal tract system by a time-varying excitation source [4, 19]. The vocal tract system, including the coupling of the nasal tract, can be accurately described in terms of positions of the articulators such as tongue, lips, jaw and velum. Generally, the vocal tract system is approximately described in terms of the acoustic features such as the frequencies of the resonances (formants) and anti-resonances (anti-formants) of the system. These features are easier to extract from the signal than the articulatory parameters. The excitation of the vocal tract system consists of broadly three categories: (1) voiced source (the quasi-periodic excitation due to the vibrating vocal cords), (2) unvoiced source (the turbulent flow of air at a narrow constriction created in the vocal tract during production), and (3) plosive source (the abrupt release of the pressure built up behind a closure in the vocal tract system). The voiced source is characterized by the periodicity (pitch period) and the change of the pitch period with time (intonation). In general, the short-time characteristics of the speech signal are represented by the short-time (10-20 ms) spectral features of the vocal tract system as well as the nature of excitation in the short-time segment. These are called segmental features. Suprasegmental features of speech are represented by intonation, the durations of different sound units, and the coarticulation reflecting the dependence of characteristics of one sound unit on the neighboring sound units during speech production. Different features commonly used in speech tasks are given in Table 19.3.

Fig. 19.1 shows the speech signal waveform, the short-time (ST) spectrum and the linear prediction (LP) spectrum for voiced and unvoiced segments. The envelope of the short-time spectrum indicated by the LP spectrum shows peaks corresponding to the resonances of the vocal tract system for the segment. The resonance characteristics are determined by the shape of the vocal tract, which in turn depends on the sound unit being produced.

The time variation of the vocal tract system is indicated by the dark bands in the wideband spectrogram of the utterance of a sentence shown in Fig. 19.2. This figure also shows the amplitude variation of the speech signal and the pitch contour. The pitch contour gives an indication of the intonation pattern of the speech. Thus several suprasegmental features can be seen in the plots of Fig. 19.2.

Speech is a sequence of sound units corresponding to a linguistic message. The meaningful sound units *or* phonetic units are different for different languages. It is generally difficult to relate the basic sound units to the text symbols used for a language. In some languages there may be good correspondence of the phonetic units to the written text as in most Indian languages. In fact the alphabet in these languages can be approximately related to the basic sound units. Typical sound units for an Indian language are shown in Fig. 19.3. In fact, phonemes, the sound units in English (see Fig. 19.4) are difficult to associate with the string of 26 text symbols of the English alphabet. The objective in speech recognition is to determine the sound units by processing speech.

The main problem is in processing the speech signal in a manner similar to the human auditory mechanism, in order to extract features relevant to a particular task. The problem is further complicated by the fact that the message is conveyed not only through the segmental features but also by the suprasegmental features. It is our lack of understanding of these segmental and suprasegmental features and the methods of extraction of these features that makes speech recognition tasks extremely difficult for implementation by

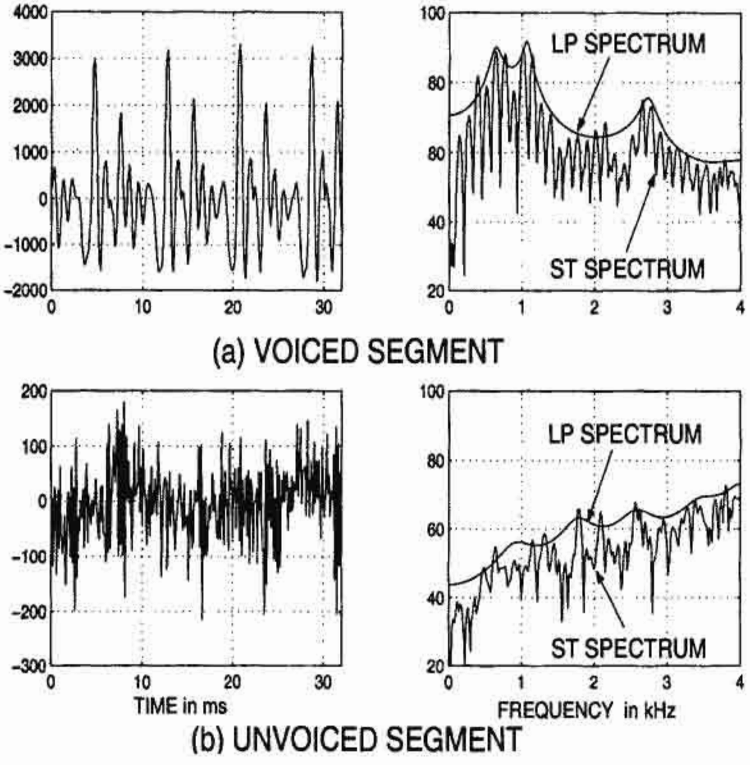


Fig. 19.1 Speech signal waveform, short-time (ST) spectrum and linear prediction (LP) spectrum for (a) voiced speech segment and (b) unvoiced speech segment

machines [6, 19].

Speech processing is a pattern recognition problem, since the signal waveform and the parameters/features derived from it are different in values each time. However, the same pattern information is conveyed, resulting in our identification of different sound units and the message contained in the sequence of these units. It is difficult to articulate the pattern information precisely. Moreover, the features relevant to a pattern are usually hidden at deep levels and hence are difficult to extract from the signal. There is also an in-built robustness in these features, and hence pattern recognition from speech is possible even when the signal is degraded by additive noise and channel effects such as in telephone speech.

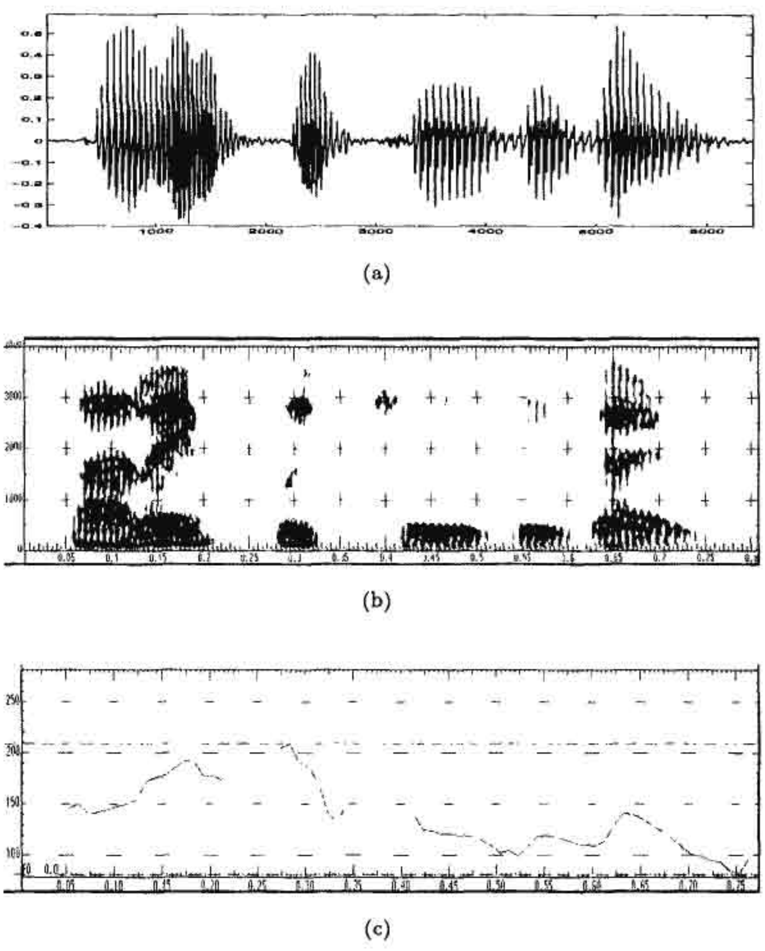


Fig. 19.2 Suprasegmental features for the phrase /talaippu cheydihal/. (a) speech signal waveform, (b) wideband spectrogram, (c) energy contour (top) and pitch contour

Another major issue related to pattern recognition in speech is the selective processing of human auditory mechanism. All the speech signal or the parameters/features extracted from it are not equally important. In fact, a few acoustic hints are captured, and the message is inferred from the sequence of these hints. This is analogous to our reading the message from a cursive script.

Thus the main issue is to determine suitable hints for each pattern recognition task, and extract the features relevant for those hints from the speech signal. Obviously, any uniform representation of information in speech will not be adequate, although that is the first step in all the existing speech pattern recognition applications.

19.3 Feature extraction in speech

Parameter and feature extraction is the first step in any pattern recognition task involving speech. However, in most cases, the features to be extracted are dictated by the production and perception of speech, rather than by the requirements of a task. For example, in speech recognition it may be necessary to distinguish even closely related sound units such as /t/ and /th/, or /p/ and /b/. In speaker recognition, one needs to identify the characteristics unique for a given speaker ignoring the speech-related features. In voice disorder identification, it is necessary to determine the peculiarities of the new voice source, ignoring the characteristics of the normal voice source. If the issues relevant to the given pattern recognition tasks are not addressed at the feature extraction stage, then one may end up in the familiar curse of dimensionality problem. That is, additional irrelevant data/features may seriously limit the performance of the pattern recognition task.

Another major problem in speech processing is that, even in cases where we know what the relevant features are, processing of speech is dictated by the available methods for feature extraction, and also by the methods for measures of similarity or dissimilarity. For example, we know that articulatory parameters are very useful for robust speech recognition. However, since they are not unique, and are also difficult to determine from the speech signal, usually parameters such as formants and magnitude of the spectral envelope are used for many pattern recognition tasks. Different methods are used for extraction of segmental and suprasegmental features. There are several methods such as weighted cepstral distance and Itakura distance for measuring the similarity of the feature values [19, 23].

The most difficult part of speech processing is to determine the long term features and extract them for pattern recognition. It is also difficult to match the long term pattern features, such as the features in a sequence of instants of excitation or the features in formant contours.

Most speech systems use parameters or features derived from short-time spectrum of speech. A segment of 20 ms is normally considered as an analysis frame, and such frames are considered once every 10 ms. The envelope of the short-time spectrum (see Fig. 19.1) represents the frequency response of the shape of the vocal tract system in that analysis frame. The information in the envelope is represented using log energy in different frequency bands of the spectrum. The bands are chosen according to nonlinear frequency scale, called mel-scale, which is based on perceptual criterion. The short-time spectral information is also represented using mel-scale cepstral coefficients, which are discrete inverse Fourier Transform (FT) coefficients of the log spectral band energies. Normally about 15 to 20 mel-scale cepstral coefficients are found to be adequate to represent useful spectral envelope information [19].

Linear Prediction (LP) technique is also used for analysis of a frame to determine the parameters of an all-pole model of the vocal tract system for that frame [13]. The parameters are called Linear Prediction Coefficients (LPCs). Typically 10 to 14 LPCs are used for each frame of the speech signal sampled at 10 kHz. One can derive the short-time spectral envelope from the LPCs which approximates the peaks of the envelope better than the valleys. However, usually the LP spectrum is approximated by the cepstral coefficients derived using the inverse FT of the log magnitude LP spectrum. About 15 to 20 LP cepstral coefficients are used for a frame of data.

Typically, each frame of speech data is represented by a feature vector consisting of mel-scale cepstral coefficients or LP cepstral coefficients. Thus only the spectral envelope information is extracted and represented in the feature vector. Most of the source information available in the LP residual and also the suprasegmental information available in the prosodic features are ignored when deriving the information about the sound units in the speech signal. The short-time analysis produces a feature vector for every 10 ms for further processing. It is these feature vectors that are used in almost all the pattern recognition tasks in speech including speech recognition, speaker recognition, voice disorder identification and word spotting.

We will focus our discussion here on pattern recognition tasks in speech recognition. A sequence of feature vectors corresponds to a meaningful sound unit of a language. It is not easy to establish the relation between the sequence of feature vectors and the sound unit uniquely, due to the partial nature of the speech information in the vector and also due to variability in the derived feature vectors for the same sound unit in different contexts at different times for different speakers. In fact, if the sound units are close to the linguistic information, then it is easy to express a given text as a sequence of these sound units. But it is difficult to determine these units by processing the speech signal through the feature vectors. On the other hand, if the sound units are identified with the distinct groups/clusters (derived using Vector Quantization (VQ) techniques [19]) of feature vectors, then it is difficult to express the language text in terms of these units. In other words, it is difficult to create a pronunciation dictionary for words or sequence of words in a text in terms of the sound units. Thus the major challenges for pattern recognition in speech are signal-to-symbol transformation and symbol-to-text conversion.

Another important issue is similarity/distortion measure. Even if one were to associate a sound unit with a feature vector or a cluster of feature vectors or a sequence of feature vectors, one has to still match the feature vectors derived from a test utterance with the typical vector(s) or typical sequence of vectors stored as reference. A standard approach for matching two feature vectors is using Euclidean distance, although better measures based on perceptual criteria are more useful [19]. Currently, the most commonly used distance measure is the weighted cepstral distance [23].

In the next section we will discuss some of the pattern recognition models used in the context of speech recognition.

19.4 Pattern recognition models for speech recognition

The speech recognition problem we want to consider is the following: Given a speech signal, determine the text (sequence of symbols) corresponding to the signal. As mentioned earlier, the choice of symbols determines the complexity at the signal-to-symbol transformation level or at the symbol-to-text conversion level. If W is the sequence of words corresponding to the text of a speech utterance, represented as a sequence Y of observation symbols, then the following two formulations are available for the speech recognition problem: *Maximum a posteriori probability (MAP) estimate formulation:* Determine the word sequence that maximizes the probability $P(W|Y)$, which is given according to Bayes rule, as

$$P(W|Y) = \frac{P(Y|W)P(W)}{P(Y)}$$

where $P(Y)$ is the probability of the observation symbol sequence, and $P(W)$ is the probability of the word sequence for the given language. The term

$P(W)$ also represents the language model. The term $P(Y|W)$ denotes the probability of the observation sequence Y produced by the model of the given word sequence W, and it is also called the likelihood function.

Maximum likelihood formulation: If the language model is not available or not used, then the speech recognition problem can be posed as a maximum likelihood estimation problem, where the objective is to determine the word sequence (W) whose model is most likely to produce the observed symbol sequence (Y).

The observation symbol sequence is derived from the speech signal. The sequence of feature vectors can be considered as an observation symbol sequence. In such a case the size of the symbol set is infinity due to continuous nature of the component values of the feature vector. On the other hand, if the feature vectors are identified with one of the distinct groups or clusters obtained by vector quantization of the feature vectors, then the observation symbols are discrete and finite. In the former case the probabilities of the observation symbols are described in terms of a continuous probability density function.

The pattern recognition models for speech recognition tasks vary depending on the nature and complexity of the tasks. We will discuss some of the common models in this section.

(a) *Template Matching*

For isolated and connected word recognition, the pattern recognition model uses Dynamic Time Warping (DTW) algorithm, which in turn is based on dynamic programming approach. In these cases the speech utterance for each word is stored as a sequence of feature vectors. The sequence of feature vectors derived from a test utterance is matched with each of the reference word templates using the DTW algorithm. The DTW algorithm takes into account the variability due to compression and expansion of different segments of speech. The reference word that gives the least distance is marked as the recognized word.

Isolated word recognition is limited to medium size vocabulary of about 30-300 words. The main issues in this task are determination of begin and end detection of speech utterance of a word, the constraints on the warping path, and the speaker dependency of the reference templates of the words. Since all words are equally important, the performance of the task will be poor if the words are confusable.

For connected word recognition a two-level dynamic programming algorithm is used to obtain a match for the number of words in the utterance and also for the actual words. Since there are no constraints on the sequence of words produced (for example, the connected digit recognition task), all the words in the sequence are important. To limit the complexity of search, the vocabulary is limited to the range of 10 to 30 words. The main issue in this task is the variability of the words at the boundaries due to coarticulation, besides the issue of begin-end detection and speaker variability.

(b) *Statistical Methods*

Statistical distributions of the speech feature vectors are more useful for other pattern recognition tasks such as speaker recognition than for speech recognition. The reason is that the knowledge of the sequence of feature vectors is not available in these distributions. The probability distribution of spectral feature vectors for each speaker represents characteristics unique for that speaker. A distribution model of the feature vectors is derived from each speaker's data. These models are used in speaker recognition task by identifying the model that best describes the set of feature vectors derived from the test data. Gaussian Mixture Models (GMM) have been successfully used in these tasks [18]. More recently, alternatives to GMM in the form of models have also been shown to capture the speaker-specific distributions well [12, 25].

The main problem in these approaches for speaker recognition is that the same feature vectors, which are useful for representing speech information, are used for representing speaker information also. There is no effort in determining features that are specific to speaker characteristics.

(c) *Hidden Markov Models*

Statistical methods for speech recognition involve determining the probability distributions of the feature vectors for each class of the sound units, and determining the sound unit class for each frame based on the maximum *a posteriori* probability of the observed feature vector for the frame. In the training data, the feature vectors associated with each class of sound units are expected to have distinct probability distribution, but due to significant overlap of these distributions among different classes, the speech recognition accuracy based purely on the statistical distributions is poor.

On the other hand, if it is possible to take into account the knowledge of the sequence of feature vectors, then the stochastic model is expected to perform better. Markov models are stochastic models consisting of states and state transitions, and are intended to capture the sequence information. In these models each feature vector is associated with a state, and the transition from one state to another is captured in the transitional probabilities of the state transition diagram of the Markov model. A separate model is derived for each word, and the probabilities of the states and state transitions are derived using the feature vectors of the training data.

In general, the feature vector is the observation symbol, as it is derived from the short-time analysis of the speech signal. It may be difficult to associate a unique state with each feature vector. It is likely that the feature vector belonging to a state may have a probability distribution. In such a case, a given feature vector may belong to more than one state with nonzero probability. The non-uniqueness in the association of states to observation symbols led to the concept of Hidden Markov Models (HMM), where the observed sequence of feature vectors may correspond to more than one state sequence with nonzero probability. Since the actual state sequence is not explicit, but hidden in the observation sequence of symbols, the resulting models are called Hidden Markov Models. These models are also called doubly stochastic models, as opposed to the stochastic models of the simple Markov Models.

For isolated word speech recognition, one HMM is built for each word using large amount of training data corresponding to several utterances of each word. The recognition is based on the maximum likelihood estimate of the HMM model generating the given observation sequence of the test word. In order to implement this method, the structure of the HMM in terms of states and observation symbols have to be decided a priori, based on the knowledge of the speech production and some preliminary studies.

For task-specific large vocabulary continuous speech recognition, it is necessary to define a set of subword units, and express each word in the vocabulary as a sequence of these subword units. During training, the speech data and the corresponding sequence of subword units are used to derive the HMM for each unit. The sentences of the task are represented using a network model consisting of permitted sequences of words for the task. Each word in turn has a pronunciation dictionary in terms of the subword units. Thus any legal sentence corresponds to a sequence of subword units constrained by the grammar in the network. In the recognition stage the sequence of feature vectors (symbols) is derived from the input speech data. The sequence of subword units and the corresponding text that gives maximum likelihood estimation for the observed symbol sequence is marked as the recognized utterance. There are several design choices in the HMM-based large vocabulary speech recognition. The most important one is the choice of the subword units. If the units are closer to the text, then it is easier to derive the pronunciation dictionary for the words, but it is difficult to identify the units in the speech signal. On the other hand, if the units are derived based on the characteristics of the feature vectors derived from speech signal, then it is difficult to build the pronunciation dictionary for the words. Moreover, if a large number of subword units are used, then it may be difficult to build HMM models for all the units, due to inadequate training data for each unit. If only a small number of subword units are used, then there will be significant overlap of the feature vectors among the subword units. The second major issue is the design choices for the HMM, namely the number of states, constraints on the state transitions and the distribution of feature vectors for each state. Typically phoneme-like units are chosen as subword units, which are about 50 for English language. Each unit is represented as a 3-state left-right HMM, and each state is described in terms of a Gaussian Mixture Model (GMM) of the feature vectors. Most HMM-based speech recognition systems are speaker-dependent, as the feature vectors and their distributions are derived from the speech data of an individual speaker. One of the main research issues is to adapt the systems for a new speaker with minimum additional training. Another major issue is the vulnerability of the feature vectors for degradations in the input speech. In fact, most speech recognition systems are not robust against noise and other degradation in the input speech. A third issue is the ill-formed nature of the speech utterances. The input speech normally may not exactly correspond to a legal sentence of the language due to casual nature of speaking in a dialogue mode. The language model built into the recognition system corresponds to syntactically driven utterances. Finally, it is not clear how to use the significant additional knowledge available in the input speech in the form of prosody (intonation and duration) within the

HMM frame work to improve the performance of the speech recognition system, and also how to make the system robust.

(d) *Neural Network Models for Recognition of Subword Units*

When the number of subword units is large, the available speech data for training each class is small, and the feature space for each class is complex, then one may consider exploiting the possibility of using the properties of neural network models for classification of these units. A Consonant-Vowel (CV) utterance typically forms a production unit (syllable) in speech, and hence attempts have been reported for recognition of CV utterances [8]. Since these are dynamic sounds, the spectral pattern changes with time. Each utterance of a CV unit is represented as a temporal sequence of spectral feature vectors. Each spectral vector corresponding to a fixed 10 ms segment may be represented by, say, 16 log spectral coefficients on a mel-frequency scale, or using the corresponding mel-scale cepstral coefficients. The number of spectral vectors per CV utterance generally varies. A fixed duration in the range 50-200 ms segment of CV, enclosing the vowel onset, the transition to vowel and some steady part of the vowel, can be used to represent a CV unit. The CV units are thus temporal sequence patterns, and hence static pattern recognition networks like MultiLayer Feed Forward Neural Networks (MLFFNN) are not suitable for recognition of these units. Moreover the discrimination among these CV units is low due to domination of the vowel part.

An obvious method to perform the sequence recognition is to view the temporal sequence of the spectral vectors as a two-dimensional spectral input pattern for a MLFFNN. The conventional backpropagation learning can then be used to train the network. A better approach to CV recognition is through Time Delay Neural Networks (TDNN) [21]. TDNN is a MLFFNN with its input consisting of the delayed input frames of data. The input to the intermediate hidden layers also consists of the delayed outputs of the preceding layer. Multiple copies of the TDNN are aligned with adjacent spectral vectors. For each TDNN, each unit in a layer is connected to all the units in the layer below it. Multiple copies of the TDNN enable the entire history of the network activity to be present at the same time. This allows the use of the backpropagation learning algorithm to train the network. The TDNN was able to discriminate three classes /b/, /d/, /g/ with an accuracy of 98.5%. Considering the fact that the data for each class has significant variation due to contextual effect, this result is impressive.

Extension of this network model for large number of CV classes requires modular approach, where it is necessary to distinguish the group of the CV classes first before the individual classes can be identified [21]. Moreover, because of the large size of the network, training of the network will be slow. It will also be difficult to collect sufficient training data to obtain good generalization performance from such a large network.

For the development of a recognition system for large number of CV units, we consider as an illustration, the recognition of the Stop-Consonant-Vowel (SCV) utterances in Indian languages [2]. In particular, we consider the SCV units of the Indian language, Hindi. These are highly confusable set of sound units due to close similarity in the production of some of these sound units, such as /ta/ and /tha/. The 80 SCV units can be organized into different set of subgroups using criteria guided by the phonetic description of these classes. Accordingly one set, called the Manner Of Articulation (MOA) set, is based on the four different manners of articulation. The second set, called the Place Of Articulation (POA) set, is based on the four different places of articulation and the third set, called the vowel set, is based on the five different vowels. Separate Feed Forward Neural Networks (FFNNs), called subnets, can be trained to discriminate the units within each subgroup of each set. A modular network is developed for each set by simply collecting the best output from all the subnets within the set for a given test input. It is found that the POA set gives a better performance (35.1%) than the other two sets of grouping (29.1% for the MOA set and 30.1% for the vowel set).

It is possible to improve the classification accuracy significantly by properly combining the evidence available at the output of the subnets. Confusability among the classes can be resolved to some extent by using the acoustic-phonetic knowledge of the classes. This knowledge can be incorporated as constraints to be met by the classes. For the Constraint Satisfaction Model (CSM), a feedback neural network is used with one unit for each of the 80 SCV classes. There are three different feedback networks, one for each of the three grouping criteria. Since the SCV classes within a subgroup compete among themselves during training of a subnet, excitatory connections are provided between the units of the classes in a subgroup. The connections between units across the subgroups are made inhibitory.

The weight on the connection between a pair of units is determined based on the similarity between the classes represented by the units. The similarity between two SCV classes is determined from the knowledge of speech production, and also from the confusability between them as indicated from the outputs of the subnets.

The feedback networks for different grouping criteria interact with each other through a pool of units, called instance pool [15]. There are as many units in the instance pool as the number of SCV classes. Each unit in the instance pool has a bidirectional excitatory connection with the corresponding unit in the feedback networks. Units within the instance pool compete with one another and hence are connected by a fixed negative weight.

The three feedback networks along with the instance pool constitute the CSM reflecting the known speech production knowledge of the SCVs as well as the knowledge derived from the subnets. The constraint satisfaction model is initialized suitably, and then allowed to relax until a stable state is reached for a given input, using a deterministic relaxation method [9]. The outputs of the units in the instance pool can be interpreted to determine the class of the input pattern. The overall performance of the CSM for all 80 SCV classes is 65.6% as compared to the best performance of 35.1% for the modular network. Thus a properly designed neural network architecture can significantly improve the classification performance of such difficult pattern recognition tasks.

(e) *Hybrid HMM and Neural Network Models for Speech Recognition*
Hybrid models are proposed [16] to take the advantage of the sequence modeling capability of the HMM and the pattern classification (a posteriori probability estimation) capability of the neural networks. In one such system a *MLFFNN* is trained for classification of the HMM states, and the outputs are interpreted as *a posteriori* probabilities for a contextual window of the sequence of feature vectors. Typically the input to the MLFFNN consists of feature vectors for nine consecutive frames, *i.e.*, four left and four right contexts for each frame. The network thus will have several thousands of parameters or weights to be trained. About 60-70 context-dependent acoustic phones or sound units are used for the classification task.

The HMM in this case is only a single state HMM for each phone, and the density per phone is estimated from the trained MLFFNN. The HMM state classification thus reduces to phonetic classification. Each phonetic HMM state is represented by a single probability distribution function, *i.e.*, a single output of the MLFFNN. It was found that the performance of the recognition system could be improved significantly by the use of emission probabilities generated by MLFFNN, rather than by a standard discrete or Gaussian mixture system [17].

19.5 Challenges in pattern recognition tasks in speech

In this chapter we have discussed the pattern recognition tasks in speech and some of the currently available approaches for these tasks. In order to appreciate the issues in these pattern recognition tasks better, we have discussed the nature of the speech signal with reference to speech production mechanism. The parameters and/or features are extracted from the speech signal using currently available signal processing techniques. Most of the techniques attempt to extract and represent the short-time spectral envelope of a speech segment. The pattern recognition models used for various tasks show the predominance of statistical approaches. In particular, HMM has been the most successful model for speech recognition tasks.

There are many challenging issues that need to be addressed in order to develop practical speech systems, such as for speaker recognition, speech recognition, word spotting and dialogue systems. Some of these issues are briefly discussed in this concluding section.

(1) *Temporal nature of speech features*
Most speech and speaker information is conveyed in the temporal features such as , , , *etc.* It is not clear how to develop pattern matching techniques for such temporal features. This is probably the biggest challenge in speech pattern recognition tasks.

(2) *Selective nature of features for specific tasks*
Generally speech and speaker characteristics are available in specific segmental and suprasegmental features. It is difficult to isolate these characteristics in the speech signal. Uniform representation of entire speech information is likely to produce the same kind of problem as in the other pattern recognition tasks, namely the curse of dimensional-

ity of pattern space. Too much of information is detrimental to the performance of a pattern recognition task.

(3) *Choice of subword units*
For most speech recognition tasks the choice of the basic sound units is difficult because the characteristics of the units vary significantly, especially due to context. In fact human beings seem to perceive the units mostly by context dictated by production, perception and language constraints. Identification of these units in the speech signal is a major challenge in speech recognition. This can easily be appreciated when we realize the difficulty in identifying individual characters in a casually written cursive script.

(4) *Robustness*
Since the performance of a speech system depends on matching the features in the test utterance with those collected during training, the performance of the system degrades if the feature vectors are significantly different during testing and training. Feature vectors extracted from the speech signal are affected by the environmental conditions, such as noise and channel characteristics. It is important to determine features which are robust against such degradations. Suprasegmental features such as intonation and duration are robust, but we do not know how to use them in speech and speaker recognition systems. Even temporal features such as epoch sequence and formant contours are also robust, but at present there are no good pattern recognition models that take advantage of these features.

(5) *Speech recognition in natural dialogue*
Most speech production in a dialogue mode consists of many ill-formed sentences and also several non-speech utterances. Human beings have a remarkable ability of filtering out the irrelevant portions, and interpret the ill-formed utterances as normal meaningful sentences. It appears impossible to model this dialogue situation of human communication. Until we succeed in this modeling, it becomes very difficult to use speech systems in practical environments.

(6) *Speech translation from one language to another*
It is interesting to note that translation by human beings is easier at the speech level than at the text level. The reason is that speech contains much more information in the form of segmental and suprasegmental information, whereas most of the current systems seem to rely only on the spectral information. Also human beings use their perceptual

mechanism and contextual knowledge to interpret the speech signal directly, and thus overcome the deficiencies due to ill-formedness of the utterances and degradation in the speech signal.

(7) *Limitations of current pattern recognition approaches*
Most of the current approaches assume a sequential model of pattern recognition task, namely, sensing, preprocessing, feature extraction, matching and decision making. On the other hand, the biological neural network seems to act from the signal level itself, thus enabling the human pattern recognition process to be robust and graceful against degradations. The biological system also seems to use selective attention for isolating the relevant features, and a delayed decision approach using soft decisions at each stage.

Thus more of soft-computing tools such as neural networks, fuzzy logic, and evolutionary computation, besides the signal processing and statistical models, are needed to develop the sophisticated pattern recognition models for speech systems to make the systems useful in practice.

References

[1] J.C.Bezdek, "A review of probabilistic, fuzzy and neural models for pattern recognition", in *Fuzzy Logic and Neural Network Handbook* (C.H.Chen, Ed.), McGraw-Hill, pp.2.1-2.33, 1996.

[2] C.Chandra Sekhar, *Neural Network Models for Recognition of Stop Consonant-Vowel (SCV) Segments in Continuous Speech*, Ph.D. thesis, I.I.T., Madras, April 1996.

[3] F.S.Cooper, "Acoustics in human communication: Evolving ideas about the nature of speech", *J. Acoust. Soc. Amer.*, vol.68, pp.18-21, 1980.

[4] J.R.Deller, J.G.Proakis and J.H.L.Hansen, *Discrete Time Processing of Speech Signals*, New York: Macmillan, 1993.

[5] P.A.Devijver and J.Kittler, *Pattern Recognition - A Statistical Approach*, New Jersey: Prentice Hall Inc., 1982.

[6] J.L.Flanagan, *Speech Analysis, Synthesis and Perception*, 2nd ed., New York: Springer-Verlag, 1972.

[7] M.Greenberger, *Computers and the World of the Future*, Cambridge: MIT Press, 1962.

[8] J.Harrington, "Acoustic cues for automatic recognition of English con-

sonants", in *Aspects of Speech Technology (M.A.Jack and J.Laver, eds.)*, Edinburgh: Edinburgh University Press, pp.69-143, 1988.

[9] S.Haykin, *Neural Networks: A Comprehensive Foundation*, New Jersey: Prentice-Hall International, 1999.

[10] A.K.Jain, R.P.W.Duin and J.Mao, "Statistical pattern recognition: A review", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.22, pp.4-37, Jan. 2000.

[11] L.Kanal, "Patterns in pattern recognition: 1968-1974", *IEEE Trans. on Inform. Theory*, vol.20, pp.697-722, 1974.

[12] S.P.Kishore and B.Yegnanarayana, "Speaker verification: Minimizing channel effects using auto-associative neural network models", in *Proceedings of International Conference on Acoustics, Speech and Signal Processing, Turkey*, pp.1101-1104, June 6-9, 2000.

[13] J.Makhoul, "Linear prediction: A tutorial review", in *Proceedings of the IEEE*, vol.63, pp.561-580, 1975.

[14] J.Mantas, "Methodologies in pattern recognition and image analysis: A brief survey", *Pattern Recognition*, vol.20, pp.1-6, 1987.

[15] J.L.McClelland and D.E.Rumelhart, *Explorations in Parallel Distributed Processing*, Cambridge MA: MIT press, 1988.

[16] N.Morgan and H.Bourlard, "Continuous speech recognition - An introduction to the hybrid HMM/connectionist approach", *IEEE Signal Processing Magazine*, vol.12, pp.24-42, 1995.

[17] N.Morgan and H.Bourlard, "Hybrid connectionist models for continuous speech recognition", in *Automatic Speech and Speaker Recognition*, C.H.Lee, F.K.Soong and K.K.Paliwal (Eds.), Kluwer Academic Publishers, pp.259-284, 1996.

[18] NIST, "Speaker recognition workshop notebook", in *Proceedings of NIST 2000 Speaker Recognition Workshop, University of Maryland, USA*, June 26-27, 2000.

[19] L.Rabiner and B.H.Juang, *Fundamentals of Speech Recognition*, New Jersey: Prentice-Hall, 1993.

[20] R.Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches*, New York: John Wiley and Sons, 1992.

[21] A.Waibel, "Modular construction of time-delay neural networks for speech recognition", *Neural Computation*, vol.1, pp.39-46, 1989.

[22] A.Waibel, T.Hanazawa, G.Hinton, K.Shikano and K.J.Lang, "Phoneme recognition using time-delay neural networks", *IEEE Transactions on*

Acoustics, Speech and Signal Processing, vol.37, pp.328-339, 1989.

[23] B.Yegnanarayana and D.Raj Reddy, "A distance measure based on the derivative of linear prediction phase spectrum", in *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp.744-747, 1979.

[24] B.Yegnanarayana, *Artificial Neural Networks*, New Delhi: Prentice-Hall of India, 1999.

[25] B.Yegnanarayana, S.P.Kishore and A.V.N.S.Anjani, "Neural network models for capturing probability distribution of training data", in *Proceedings of Fourth International Conference on Cognitive and Neural Systems, Boston*, p.6(A), May 25-27, 2000.

Table 19.1(contd.) Pattern recognition tasks in the context of speech. The objectives, types and methods for each task are given in the table

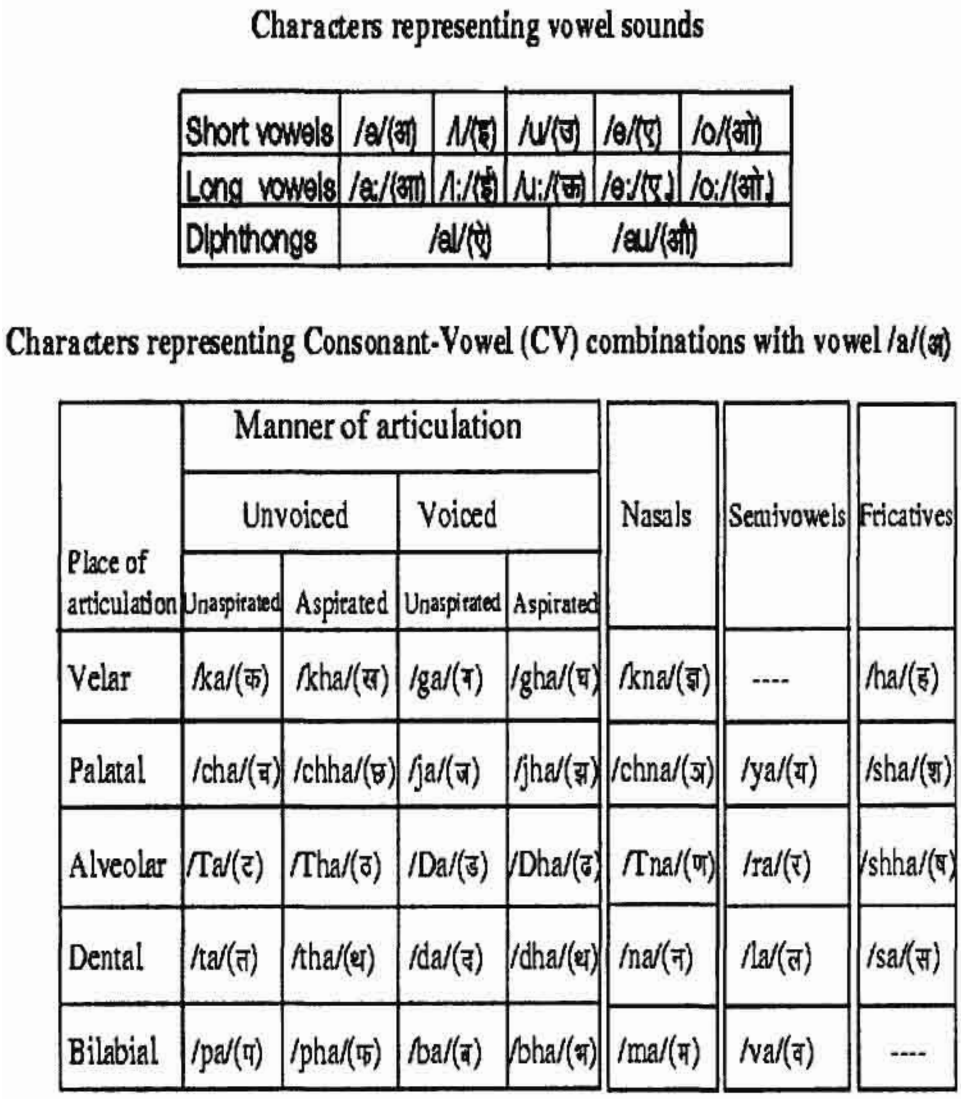
Task	Type	Objective	Issues	Method
Speaker recognition	Speaker identification	Identify a person by matching the speech data with models of different speakers	Extraction of speaker specific features; Score normalization; Trade-off between false acceptance and false rejection	Statistical methods; Neural network methods
	Speaker verification	Verify a claim by matching the speech data with the model of a speaker		
	Speaker tracking	Mark the regions where the data of a speaker is present in the the given speech		
Spoken language identification	Language identification	Determine the language of the given speech by matching with models of different languages	Capturing the statistical distributions of basic sound units of a language	Statistical methods; Neural network methods
Voice disorder identification		Determine the type of voice disorder from speech	Acoustic analysis of source parameters	Signal processing methods
Audio-video interaction		Person authentication using biometrics that involves inputs in the form of speech and video	Determination of interactive multimedia features; Combining the evidence	Multimedia signal processing methods

Table 19.2 Methods/models for solving general pattern recognition tasks

Approach	Representation	Method	Comments
Template matching	Data level	Match the data of a test pattern point by point with the data in the reference pattern	Suitable for simple and restricted tasks
	Feature level	Match the descriptions of patterns in terms of features	Works well when features are dictated by the task
	Deterministic model	Match the parameters of the models with the test pattern	Works well when the source of pattern generation process is known
Statistical method	Probability distribution of features	Match the patterns by statistical methods	A large set of training patterns is required to derive parameters of models
Syntactic method	Primitives suitable for classes	Match the descriptions of patterns in terms of primitives	It may be difficult to extract primitives from the data
Knowledge-based method	Rules	Verify validity of rules for a the test pattern	It may be difficult to derive rules and thresholds from data
Neural network method	Weights of connections among nodes	Match the test pattern with the features of classes captured in the weights	Suitable for fixed length patterns
Soft computing method	Knowledge in the form of probabilistic, fuzzy and rough uncertainties	Uncertainty-based pattern matching	Identifying the types of uncertainties relevant for a task is important

Table 19.3 Features commonly used for speech tasks

Segmental features	System features	Vocal tract parameters, Nasal tract parameters, Articulatory shape
	Source features	Voicing, Pitch period, Instants of significant excitation, Glottal shape
	Spectral features	Short-time spectral envelope, Formants, Anti-formants, Mel-frequency coefficients
	Temporal features	Pitch period, Instants of significant excitation, Glottal pulse shape
	Static features	Spectral and temporal features
	Dynamic features	Formant changes, Spectral changes, Time-frequency representations
Suprasegmental features	Prosodic features	Intonation, Duration, Intensity
	Parameter contour features	Spectrum contour, Formant contour, Instants with strengths of excitation
	Long-term features	Modulation spectrum



Characters representing vowel sounds

Short vowels	/a/(अ)	/i/(इ)	/u/(उ)	/e/(ए)	/o/(ओ)
Long vowels	/a:/(आ)	/i:/(ई)	/u:/(ऊ)	/e:/(ए)	/o:/(ओ)
Diphthongs	/ai/(ऐ)			/au/(औ)	

Characters representing Consonant-Vowel (CV) combinations with vowel /a/(अ)

	Manner of articulation						
	Unvoiced		Voiced		Nasals	Semivowels	Fricatives
Place of articulation	Unaspirated	Aspirated	Unaspirated	Aspirated			
Velar	/ka/(क)	/kha/(ख)	/ga/(ग)	/gha/(घ)	/kna/(ङ)	----	/ha/(ह)
Palatal	/cha/(च)	/chha/(छ)	/ja/(ज)	/jha/(झ)	/chna/(ञ)	/ya/(य)	/sha/(श)
Alveolar	/Ta/(ट)	/Tha/(ठ)	/Da/(ड)	/Dha/(ढ)	/Tna/(ण)	/ra/(र)	/shha/(ष)
Dental	/ta/(त)	/tha/(थ)	/da/(द)	/dha/(ध)	/na/(न)	/la/(ल)	/sa/(स)
Bilabial	/pa/(प)	/pha/(फ)	/ba/(ब)	/bha/(भ)	/ma/(म)	/va/(व)	----

Fig. 19.3 Typical sound units in an Indian language

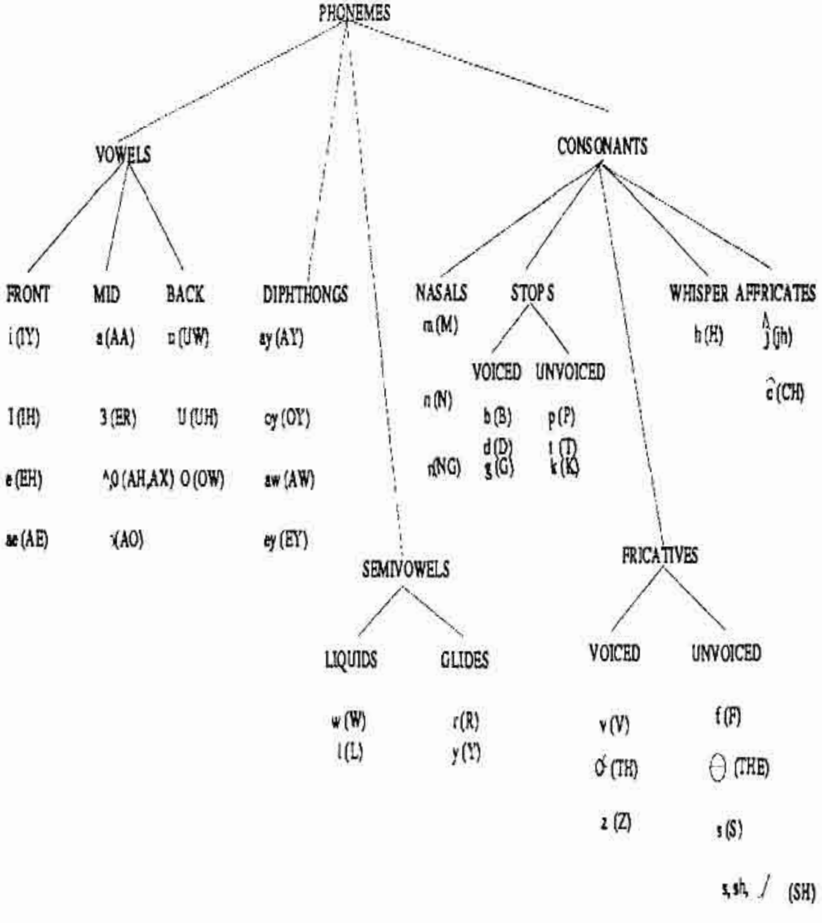


Fig. 19.4 Sound units (phonemes) in English [19]

Chapter 20

WRITING SPEED AND WRITING SEQUENCE INVARIANT ON-LINE HANDWRITING RECOGNITION

S.-H. Cha and S. N. Srihari

Centre of Excellence for Document Analysis and Recognition
State University of New York at Buffalo
Amherst, New York 14228, U.S.A
e-mail: {*scha,srihari*}*@cedar.buffalo.edu*

Abstract

On-line handwriting recognition is a useful application of pattern recognition. Despite its success, most existing systems would not recognize a character if it were written in a different writing sequence from the conventional writing system taught in school. We identify some cases where existing recognizers fail and present preprocessing techniques to overcome these cases. Due to the preprocessing, we make the recognizer invariant to writing speed and writing sequence. We utilize the stroke direction sequence string as a stepping stone and various string manipulation operators pave a way for a writing speed and writing sequence invariant on-line handwritten character recognition system. The presented techniques allow more robustness to noise to the recognizer as well as more freedom in writing to writers.

20.1 Introduction

The on-line handwritten character recognition problem is a character classification problem where spatio-temporal patterns are produced by a digitizer or an instrumented stylus that captures information about the pen-tip, generally, its position, velocity, or acceleration as a function of time. The inputs are usually the two-dimensional coordinates of successive points of the writing as a function of time that are stored in order, *i.e.*, the order of strokes made by the writer is readily available [11]. Not surprisingly, it has received a great deal of attention because of its convenience to users compared to keyboards and has found many commercial applications such as PDAs (Personal Digital Assistants). (See [7, 11, 13] for a comprehensive and detailed survey on successful techniques and applications of on-line handwriting recognition).

Characters were originally created and have been developed to be visual symbols that serve as a means of communication among humans. Unlike speech, characters are not temporal patterns. On-line handwritten character recognition, however, utilizes temporal signals such as stroke sequence to recognize the character. This causes some drawbacks. Spatially same characters can be written very differently in terms of temporal data. Moreover, most on-line handwriting recognizers require the writers to use the same writing system taught in school. For this reason, we present preprocessing techniques to allow recognizers to be invariant to writing speed and writing sequence.

One writing system taught today is *Modern Vertical* introduced by A. N. Palmer in 1923 [1]. There are many other different writing systems, such as, *Round Hand, Spencerian, Modern American and Commercial.* People of different ages, ethnicity, education backgrounds and handedness may use different writing systems. An on-line handwritten character recognizer trained with a particular writing system would not recognize characters written in other writing systems or in different order of stroke sequences although they look exactly the same in terms of shape. The different writing systems can be handled by collecting a substantially large number of specimens provided by many writers [14]. This chapter deals with handwritten characters with various styles allowing breaks and different writing sequences, and suggests a solution.

Another important motivation for this study is the unnatural feel of the electronic on-line input devices. The input patterns for the on-line handwritten character recognition are generated by a mouse on a mouse pad or a special electronic pen on an electronic surface such as a digitizer combined with a liquid crystal display. Users or writers may not produce their handwriting using the electronic devices as naturally as using the pen and paper. The unnatural and unfamiliar input devices may result in unnatural handwriting. This fact necessitates the recognizer to be more robust to unnaturally written text, *i.e.*, a recognizer with various invariant properties.

20.1.1 Desirable invariance properties

In seeking to achieve an optimal representation for a particular pattern classification, we confront the problem of *invariance* such as orientation, size, rate, with respect to characteristics deformation and discrete symmetry in the classification problems in scene analysis [6]. The desirable invariance properties in the context of on-line handwritten character recognition are with respect to noise [7], distortions [9], baseline drift [2], slant [3], script size [7], break [15], etc. Nouboud and Plamondon used data smoothing, signal filtering, dehooking and break corrections [7, 10].

One of the most desirable invariance properties in on-line handwriting recognition is writing speed invariant property [10, 12]. It is very essential because spatially same characters can be written temporally in a different way. Thus, a recognizer trained with temporal information fails to recognize handwriting if it is written at different speeds unless it is invariant to writing speed. This problem of writing speed invariant on-line handwriting recognition is usually tackled using chain-code [10] or stroke direction sequence strings [4, 5]. We present a procedure that converts non-uniform writing speed and acceleration data into uniform speed and acceleration data.

We assume that the on-line handwritten character input signals are normalized using conventional techniques, such as, position and size normalizations, deskewing, and base-line drift correction. In this chapter, we focus on writing speed and writing sequence invariance where most of the previously developed on-line handwritten character recognition systems fail. First, the *stroke sequence string* representation of given input signals bestows writing speed invariance to the recognizer. Detailed stroke sequence string extraction algorithms can be found in our earlier works [4, 5]. Next, this *stroke sequence string* representation paves a way for the writing sequence invariance property by using several string manipulation operators, *e.g.*, *concatenation, reverse* and *ring* operations. The idea of concatenation was also considered by Plamondon and Nouboud for the case of no constraint on the writing, the system being able to recognize any character defined by the user [10]. We introduce more string manipulation techniques to reduce further the constraints in writing.

20.1.2 Organization

The forthcoming sections are constructed as follows. In Section 20.2, we first illustrate how one can write spatially same characters temporally differently because of writing speed and acceleration. Next, we introduce a *stroke direction sequence string* [4, 5] representation of on-line input patterns and finally show that it is a writing speed normalized representation. Section 20.3 deals with writing sequence invariance. A number of string manipulation operators such as concatenation, reverse, ring, and sub-string removal are introduced. Section 20.4 discusses the character recognizer after preprocessing, and Section 20.5 concludes this work.

20.2 Writing speed invariance

In this section, we give an example of how spatially same characters can be written differently in terms of writing speed and acceleration. Next, we introduce a *stroke direction sequence string* [4, 5] representation of on-line input patterns and finally show that it is a writing speed normalized representation. Several subjects were asked to copy a digit *"2"* image as shown in Fig. 20.1 (a)

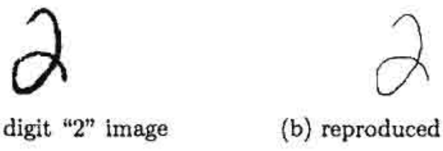
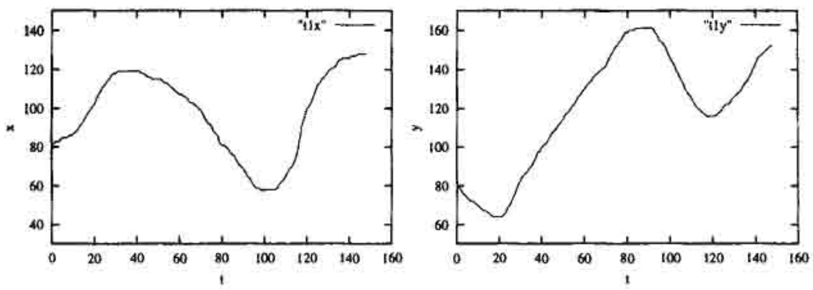
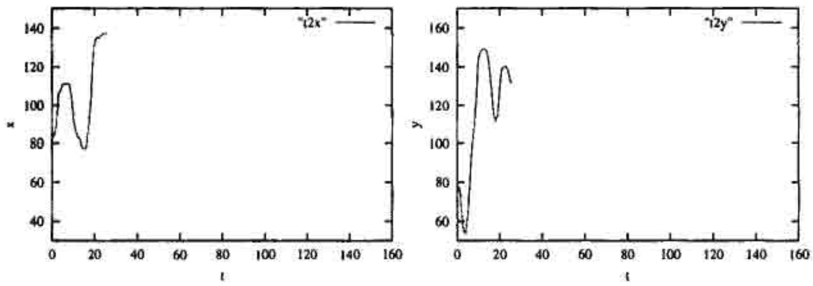

(a) original digit "2" image (b) reproduced on-line "2"

Fig. 20.1 Sample digit image "2"

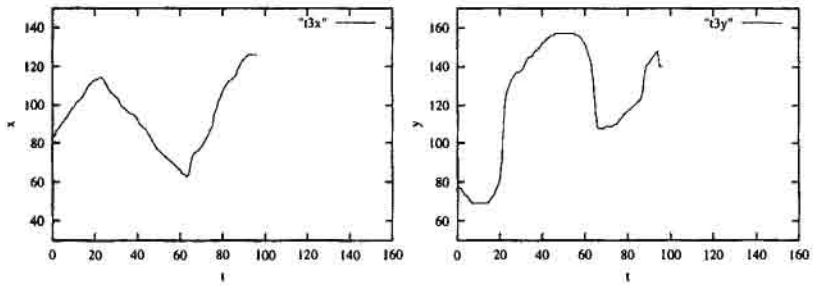
to produce the on-line data as in Fig. 20.1 (b) and everyone writes differently in terms of writing speed and time. Fig. 20.2 illustrates the case of various temporal writing sequence inputs for the spatially same character shape. Some write fast and some write slowly as in Fig. 20.2 (a) and (b), respectively. Fig. 20.2 (c) presents $X-Y$ graphs where a subject writes *"2"* in non-uniform writing speed and non-uniform acceleration. Although their spatial patterns are exactly the same, their temporal data are different due to different velocities, $v(t) = [(\frac{dx(t)}{dt})^2 + (\frac{dy(t)}{dt})^2]^{1/2}$, and different accelerations, $a(t) = \frac{dv(t)}{dt}$. See Fig. 20.3 for velocity and acceleration graphs corresponding to XY-graphs in Fig. 20.2. In this section, we present a method for normalizing the different temporal data into uniform writing speed data using the *stroke direction sequence string*.



(a) slow writing X and Y position graphs



(b) fast writing X and Y position graphs



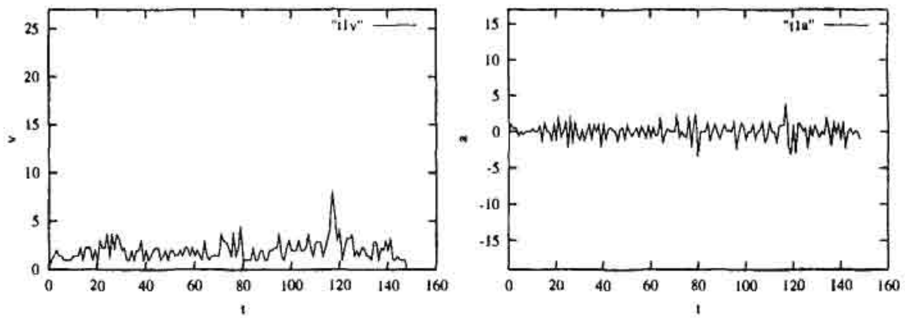
(c) non-uniform writing speed X and Y position graphs

Fig. 20.2 Various on-line XY-graphs for spatially same character "2"

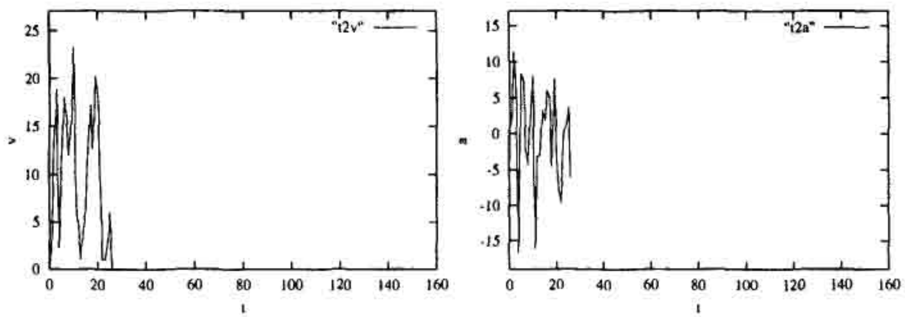
20.2.1 Stroke sequence string

A string is a sequence of symbols drawn from the alphabet Σ. We will use the following notations and symbols throughout the rest of this chapter.

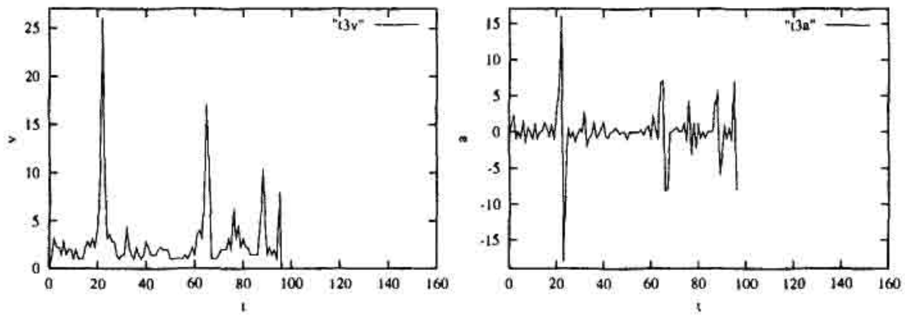
$$S_1 = (s_{1,1}, s_{1,2}, \cdots, s_{1,n_1})$$
$$S_2 = (s_{2,1}, s_{2,2}, \cdots, s_{2,n_2})$$



(a) $v(t)$ and $a(t)$ for Fig. 20.2 (a)



(b) $v(t)$ and $a(t)$ for Fig. 20.2 (b)



(c) $v(t)$ and $a(t)$ for Fig. 20.2 (c)

Fig. 20.3 Velocity and acceleration graphs for graphs in Fig. 20.2

S_1 and S_2 are strings, and n_1 and n_2 denote the length of each string. Each symbol in the string, $s_{x,y}$ has two index labels where the first index, x indicates the string to which it belongs and the second index, y indicates the location of the symbol in the string ($1 \leq y \leq n_x$).

In the stroke direction sequence string, symbol values are angular degrees. Stroke directions are quantized into r directional values as shown in Fig. 20.4, *e.g.*, (*Freeman style chain-code scheme* has $r = 8$ directional values). We

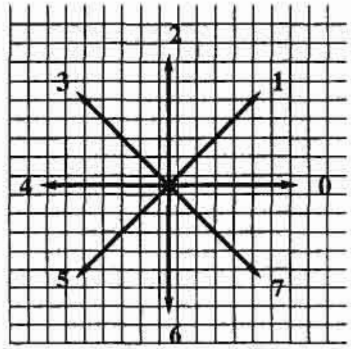


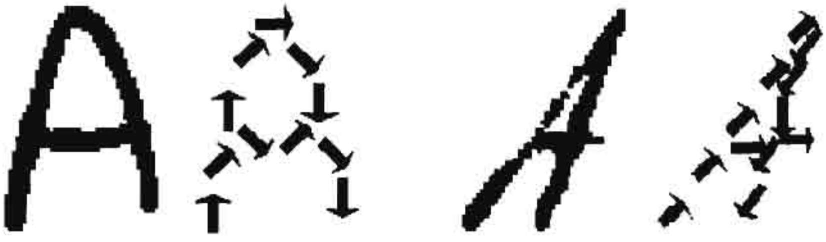
Fig. 20.4 Strokes with 8-directions and 7 pixel length

chose $r = 8$ because the approximate length of 7 for diagonal arrows can be achieved by moving 5 pixels to the right and 5 to the north; the exact distance is 7.071 as shown in Fig. 20.4. An alternative choice is $r = 12$. This gives the approximate length for the 1 o'clock direction by moving 4 pixels to the right and 7 pixels to the north; the exact distance is 8.062. Although the length error is smaller than the previous case, it has the angle error. The exact angle between the 0 stroke and 1 stroke is $29.74°$ whereas the desired one is $30°$.

As shown in Fig. 20.5, a character image can be represented by a piecewise linear stroke sequence string. The string of arrows is called a *Stroke Direction Sequence String*, or an *SDSS*, in short, and an *SDSS* is an angular type string. We define the abstract data type of a *SDSS* as follows:

Definition 1: Stroke Direction Sequence String

```
.       struct SDSS {
.           int start_x, start_y; // coordinate of start point
.           int end_x, end_y; // coordinate of end point
.           int *sds; // Stroke direction sequence list
.           int num_of_chain; // total stroke number
.       };
```



(a) X (b) Y

$$
\begin{aligned}
\mathrm{SDSS}(X) &= (\ \mathrm{SDSS}(X_1) = [\uparrow\ \nearrow\ \uparrow\ \nearrow\ \rightarrow\ \searrow\ \downarrow\ \searrow\ \downarrow]) &&+(\ \mathrm{SDSS}(X_2) = [\searrow\ \nearrow])\\
\mathrm{SDSS}(Y) &= (\ \mathrm{SDSS}(Y_1) = [\nearrow\ \nearrow\ \nearrow\ \nearrow\ \nearrow\ \swarrow\ \swarrow\ \downarrow\ \swarrow\ \swarrow]) &&+(\ \mathrm{SDSS}(Y_2) = [\rightarrow\ \rightarrow])
\end{aligned}
$$

Fig. 20.5 Sample stroke direction sequence strings

As a character may have more than one contiguous stroke, contiguous strokes are represented in a pair of parentheses, *e.g.*, a capital letter *"A"*, $X = (X_1) + (X_2)$; a given character (letter) consists of a sequence of one or more SDSS's.

We obtain stroke directional sequence strings from the movements of a mouse or a pen-based device. To do so requires that we convert the normalized and deskewed mouse position vector that is a non-uniform length direction string into the SDSS that is a string of uniform length (7-pixel long) directions.

$$
\begin{aligned}
\text{input vector} &= \{(0,0),(0,14),(3,15),(5,19),(12,19)\}\\
\overline{(0,0),(0,14)} &= \rightarrow\rightarrow \qquad (20.1)\\
\overline{(0,14),(3,15)} + \overline{(3,15),(5,19)} &= \searrow \qquad (20.2)\\
\overline{(5,19),(12,19)} &= \downarrow\\
\text{SDSS} &= \rightarrow\rightarrow\searrow\downarrow
\end{aligned}
$$

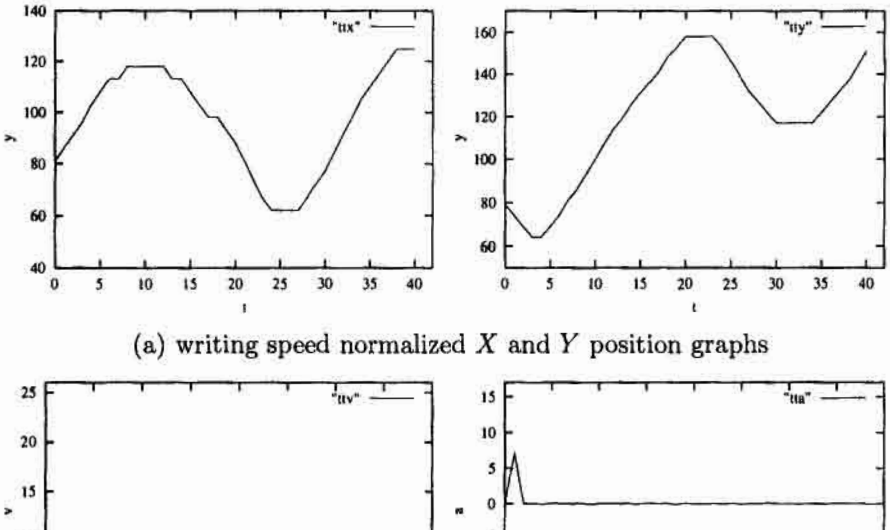
When the mouse moves fast, it creates only a few mouse positions and more strokes are filled in between these positions. Such a case is depicted in (20.1). When the mouse moves slowly, on the other hands, it creates many tiny strokes. These tiny strokes are merged into fewer 7-pixel long strokes as in (20.2). In all, geometrically best-fitting strokes are selected to create an SDSS from the size-normalized and deskewed mouse movement vector.

SDSS is generated on the basis of not temporal data but sequential and

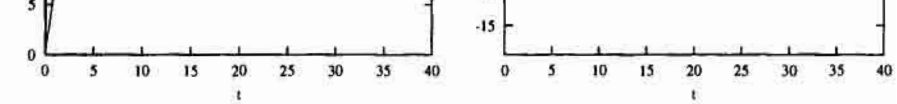
spatial data. Thus all three cases in Fig. 20.2 (a), (b) and (c) have the same or similar *SDSS* as follows:
{↗↗↗→↘↘↓↘↓↓↓↓↙↓↙↙↙↓↙↙←←←↖↑↑↑↗↗↗→→→→↘↘↘↘↓↓}.

20.2.2 String to temporal X-Y signals

An SDSS can be plotted back into the $X - Y$ position graphs. The length of the string is the time spent to draw the character. All three cases in Fig. 20.2 have the same normalized graphs as given in Fig. 20.6 (a). The velocity and acceleration are uniform 7 and 0, respectively.



(a) writing speed normalized X and Y position graphs



(b) writing speed normalized velocity and acceleration graphs

Fig. 20.6 Normalized temporal writing sequences for Fig. 20.2 character "2"

20.3 Writing sequence invariance

Once a character is represented in (a) string(s), several string manipulation operators, *e.g.*, *concatenation*, *reverse* and *ring* operations allow us to handle the problem of writing sequence invariance. Although a break correction for

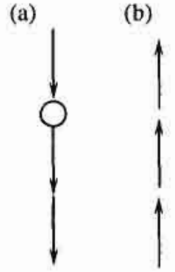


Fig. 20.7 Sample Characters "1" (a) a break in the middle (b) written backward

Chinese characters by a stroke merging technique exists in the literature [15], many on-line handwritten character recognizers would not recognize a character if it were written in a different order of drawing or if there exists a break in the middle as shown in Fig. 20.7.

20.3.1 String concatenate and reverse manipulation

This problem of awkward handwriting can be diminished by string operations: *concat* and *reverse*. *Concat* operation will concatenate two strings whose end points are near. *Reverse* operation will reverse the order of the elements as well as complement each string element. One or several breaks may occur in drawing a single line, curve or circle. This must be considered as a single string rather than multiple strings. To obviate this inadequacy, the *concat* operation is necessary. The *reverse* operation is also needed because people may write a letter in the reverse order.

This section demonstrates the step-by-step procedure with an example in Fig. 20.8. Fig. 20.8 (a) shows a handwritten character "X" in a very unnatural way of $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$. This is an example where most previous on-line character recognizers fail. We wish to manipulate the strings to make them as in Fig. 20.8 (b). As mentioned earlier, we assume that the sample image is size-, position- and slant-corrected, smoothed and deskewed.

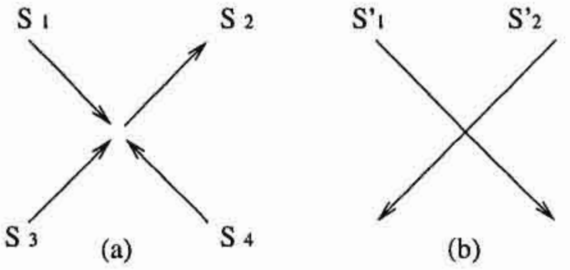


Fig. 20.8 Various writing sequences (a) Unnatural writing sequence for "X" (b) Normal writing sequence for "X"

Let $\bar{s_x}$ be a reversed writing sequence string of s and (s_x, s_y) be a concatenated string of s_x and s_y strings. First, the step $\xrightarrow{1}$ generates all possible

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} \xrightarrow{1} \begin{pmatrix} (s_1, s_2)(s_3, \bar{s_4}),\ (s_1, s_2)(s_4, \bar{s_3}) \\ (s_1, \bar{s_3})(\bar{s_2}, \bar{s_4}),\ (s_1, \bar{s_3})(s_4, s_2) \\ (s_1, \bar{s_4})(s_3, s_2),\ (s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) \\ (\bar{s_2}, \bar{s_1})(s_3, \bar{s_4}),\ (\bar{s_2}, \bar{s_1})(s_4, \bar{s_3}) \\ (\bar{s_2}, \bar{s_3})(s_4, \bar{s_1}),\ (\bar{s_2}, \bar{s_4})(s_3, \bar{s_1}) \\ (s_3, \bar{s_1})(s_4, s_2),\ (s_3, s_2)(s_4, \bar{s_1}) \end{pmatrix} \xrightarrow{2} \begin{pmatrix} (s_1, s_2)(s_3, \bar{s_4}) \\ (s_1, \bar{s_3})(\bar{s_2}, \bar{s_4}) \\ (s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) \end{pmatrix} \xrightarrow{3}$$

$$\xrightarrow{3} \begin{pmatrix} (s_1, s_2)(s_3, \bar{s_4}) \\ (s_3, \bar{s_4})(s_1, s_2) \\ (s_1, \bar{s_3})(\bar{s_2}, \bar{s_4}) \\ (\bar{s_2}, \bar{s_4})(s_1, \bar{s_3}) \\ (s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) \\ (\bar{s_2}, \bar{s_3})(s_1, \bar{s_4}) \end{pmatrix} \xrightarrow{4} \begin{pmatrix} (s_1, \bar{s_4})(\bar{s_2}, \bar{s_3})\ 95\% \\ (s_1, \bar{s_3})(\bar{s_2}, \bar{s_4})\ 87\% \\ (\bar{s_2}, \bar{s_4})(s_1, \bar{s_3})\ 30\% \\ (\bar{s_2}, \bar{s_3})(s_1, \bar{s_4})\ 20\% \\ (s_1, s_2)(s_3, \bar{s_4})\ 10\% \\ (s_3, \bar{s_4})(s_1, s_2)\ \ 5\% \end{pmatrix} \xrightarrow{5} \begin{matrix} (s_1, \bar{s_4})(\bar{s_2}, \bar{s_3}) \\ \text{class "X"} \end{matrix}$$

Fig. 20.9 Overview of character recognizer with string concat and reverse capability

strings by the concat and reverse operations. s_1 and s_2 are concatenated because the position of the terminal element of s_1 is very close to that of the starting element of s_2. $(s_3, \bar{s_4})$ becomes one string by first reversing s_4 and then concatenating s_3 and $\bar{s_4}$. There are 12 possible ways to generate a new set of strings without considering the order.

Second, in the step $\xrightarrow{2}$, we eliminate all elements in the new set of strings which violate the top-down and left-to-right sequence rule. For example, all elements containing a string $(s_4, \bar{s_3})$ are eliminated because $(s_4, \bar{s_3})$ is right-to-left sequence string. To deduce whether the string violates the rule, scan the string and add the corresponding values in (20.3).

$$\begin{pmatrix} \nwarrow & \uparrow & \nearrow \\ \leftarrow & & \rightarrow \\ \swarrow & \downarrow & \searrow \end{pmatrix} = \begin{pmatrix} -2 & -3 & -1 \\ -2 & & 2 \\ 1 & 3 & 2 \end{pmatrix} \tag{20.3}$$

If the summed value is greater or equal to 0, we accept the string. Otherwise, the string violates the rule and we reject the string. After this filtration, only three strings are left.

Since the order of strings was not considered, we generate all possible order of strings for each element. The resulting elements are produced by the step $\xrightarrow{3}$. All sequence of strings are candidates for the recognizer. In step $\xrightarrow{4}$, a recognizer takes each candidate and returns its confidence with its class. The resulting set is the ordered list of elements with confidence.

Finally, in step $\xrightarrow{5}$, the element with the highest confidence is chosen. The output sequence, $(s_1, \bar{s_4}) \rightarrow (\bar{s_2}, \bar{s_3})$ is the writing sequence of Fig. 20.8 (b).

20.3.2 Ring operator

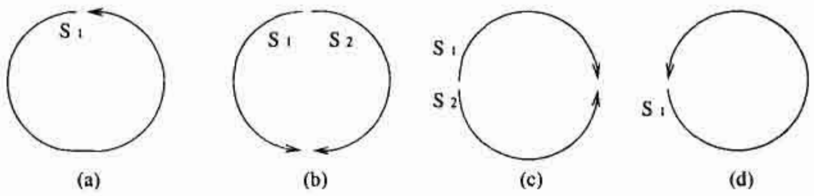


Fig. 20.10 Various ways of drawing "O"

Concatenate and reverse string manipulation operators are insufficient to solve the writing sequence invariance problem. One exceptional case is a stroke sequence that forms a ring as depicted in Fig. 20.10. We wish Figs. 20.10 (b), (c) and (d) to have the same writing sequence as Fig. 20.10 (a). This requires a special treatment called, a *ring operator*. When (a) stroke(s) form(s) a ring, *i.e*, when the start and end points coincide, a new string starts from the topmost point of the string.

Algorithm 1 shows the procedure to achieve the normalized ring that is drawn counter-clockwise from the top.

Algorithm 1 Ring Normalization

```
1        top = 0;
2        for $s_{o,1}$ to $s_{o,n}$
3            if($s_{o,i} \in \{\uparrow, \nearrow, \nwarrow\}$)
4                top = i + 1;
5        get $s_o'$ starting from $s_{o,\mathrm{top}}$
6        if (average of First quarter of $s_o \simeq \searrow$)
7            $s_o' = \bar{s}_o'$.
```

For the example of Fig. 20.10 (c), let $s_1 = (\nearrow, \rightarrow, \searrow, \downarrow)$ and $s_2 = (\downarrow, \searrow, \rightarrow, \nearrow)$. Then a new ring string is formed, $s_o = (s_1, \bar{s_2}) = (\nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \uparrow)$. Now scan the s_o to find the top (= 2nd position of the string). Next, a new ring string, s_o' is generated starting from the top: $s_o' = (\rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \uparrow, \nearrow)$. Since the string is a clockwise sequence ring, reverse s_o' to get the final string. As a result, we have a normalized ring $s_o' = (\swarrow, \downarrow, \searrow, \rightarrow, \nearrow, \uparrow, \nwarrow, \leftarrow)$. Other writing sequences in Figs. 20.10 (b) and (d) are handled similarly.

20.3.3 Sub-string removal

Another example that most on-line character recognizers fail at according to our tests is a double or multiple overwritten strokes case. As illustrated in

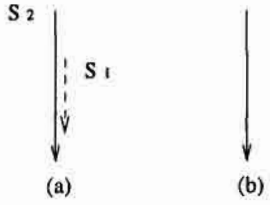


Fig. 20.11 Double stroke

Fig. 20.11, one first writes s_1 and then overwrites s_1 with s_2 expecting that the readers recognize it as *"1"*. Clearly, the spatial information is *"1"* but the temporal information is very messy. The aforementioned concatenation technique cannot handle this problem. In this section, we present a *sub-string removal* technique to overcome this issue.

First, check whether both start and end points of a string are laid on another string. If so, they are subjected to the sub-string removal procedure. We use the approximate string matching algorithm that computes the edit distance to identify the sub-string occurrence in the longer string with small errors. Since string element type is angular, we use the edit distance defined in [5]. If the edit distance is within a small threshold, we remove the smaller string.

20.4 Recognizer

Once on-line handwritten character patterns are well pre-processed by the techniques described in the previous sections, we are ready to classify the pattern into its class. Numerous methods are available for on-line handwritten character recognition and enumerated in a few exhaustive survey papers [11, 7, 13].

The problem of *on-line handwritten character recognition* can be formalized by defining a distance between characters and finding the nearest neighbor in the reference set. To recognize an unknown on-line handwritten character, one measures the edit distances between the input string and reference strings [10, 8, 4, 5]. Next, the class of an input string is determined by votes on k-nearest neighbors. As a stroke sequence signifies the shape of the individual letters, a letter *"a"* is distinguished from a letter *"b"* by its different stroke sequences.

20.5 Conclusions

In this chapter, we considered the on-line handwritten character recognition problem. We identified several cases where previously developed recognition systems fail. The problem occurs when a character is written in a different writing sequence from the conventional writing system taught in school. The stroke direction sequence string representation is a means to an end in an attempt to solve this problem. Once on-line handwritten character signal data are converted into an SDSS, various string manipulation operations are adapted to cater to a writing speed and writing sequence invariant on-line handwritten character recognition problem. They are concatenation, reverse, ring, and sub-string removal.

The proposed preprocessing is not yet a blanket approach for all writing sequences of a handwritten character. It is indeed quite easy to make up a writing sequence that would confuse a recognizer with high degree of writing sequence invariance. Examples are given in Fig. 20.12. The objective of writing

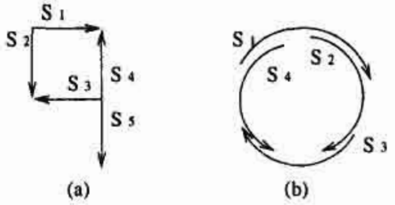


Fig. 20.12 Another hard case: (a) a digit "9" (b) "O"

sequence invariance is not to recognize a character that is intentionally written awkwardly. In all, the presented techniques allow more robustness to noise to the recognizer as well as more freedom in writing to writers.

Acknowledgments

We would like to thank all graduate students in the pattern recognition class of the fall semester of 1999. Students provided their on-line handwriting samples and most students successfully implemented the online character recognition system with the *concatenation* and *reverse* string manipulation capability as a part of homework assignment during the semester.

References

[1] R. R. Bradford and R. B. Bradford. *Introduction to Handwriting Examination and Identification.* Nelson-Hall Publishers: Chicago, 1992.

[2] M K. Brown and S. Ganapathy. Preprocessing techniques for cursive script word recognition. *Pattern Recognition*, 16(5):447–458, 1983.

[3] D J. Burr. Designing a handwriting reader. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:554–559, 1983.

[4] S-H Cha, Y-C Shin, and S. N. Srihari. Approximate character string matching algorithm. In *Proceedings of Fifth International Conference on Document Analysis and Recognition*, pages 53–56. IEEE Computer

Society, September 1999.

[5] S-H Cha, Y-C Shin, and S. N. Srihari. Approximate string matching for stroke direction and pressure sequences. In *Proceedings of SPIE, Document Recognition and Retrieval VII*, volume 3967, pages 2–10, January 2000.

[6] R. O. Duda, D. G. Stork, and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, Inc., New York, 2nd edition, 2000.

[7] F. Nouboud and R. Plamondon. On-line recognition of handprinted characters. survey and beta tests. *Pattern Recognition*, 23(9):1031–1044, 1990.

[8] M. Parizeau, N. Ghazzali and Hebert. Optimizing the cost matrix for approximating string matching using genetic algorithms. *Pattern Recognition*, 31(4):431–440, 1998.

[9] I. Pavlidis, R. Singh, and N. Papanikolopoulos. On-line handwritten note recognition method using shape metamorphosis. In *International Conference on Document Analysis and Recognition*, pages 914–918. IEEE, 1997.

[10] R. Plamondon and F. Nouboud. On-line character recognition system using string comparison processor. In *Proceedings of International Conference on Pattern Recognition*, pages 460–463. IEEE, June 1990.

[11] R. Plamondon and S. N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[12] C. Y. Suen, M. Berthod, and S. Mori. Automatic recognition of handprinted characters - the state of the art. In *Proceedings of the IEEE*, volume 68, pages 469–487, April 1980.

[13] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[14] J. R. Ward and T. Kuklinski. A model for variability effects in handprinting with implications for the design of handwriting character recognition systems. *IEEE Transactions on Systems, Man & Cybernetics*, 18:438–451, 1988.

[15] P. J. Ye, H. Hugli, and F. Pellandini. Techniques for on-line Chinese character recognition with reduced writing constraints. In *Proceedings of 7th ICPR*, pages 1043–1045. IEEE CS Press, 1984.

Chapter 21

TONGUE DIAGNOSIS BASED ON BIOMETRIC PATTERN RECOGNITION TECHNOLOGY

K. Wang[*], D. Zhang[*+], N. Li[*] and B. Pang[*]

[*]*Biometrics Research Center*
Department of Computer Science and Engineering
Harbin Institute of Technology
Harbin, CHINA
e-mail: {*wkq,pb*}*@biometrics.hit.edu.cn*

[+]*Biometrics Research Center*
Department of Computing
Hong Kong Polytechnic University
Kowloon, HONG KONG
e-mail: *csdzhang@comp.polyu.edu.hk*

Abstract

In this chapter, we present, as an aid to medical diagnosis, an automated system based on biometric pattern recognition technology, that performs the valuable diagnostic functions of the tongue diagnosis technique of Traditional Chinese Medicine (TCM). Details of tongue image capturing and database design are discussed, followed by a description of the segmentation approach we have adopted for tongue images. The feature extraction methods used by us for tongue diagnosis, as well as the classification procedure for tongue images, are also highlighted.

21.1 Introduction

Tongue diagnosis is one of the important diagnostic methods in Traditional Chinese Medicine (TCM), which has been practised in China for thousands of years. As in the ancient times, even now, many doctors can determine the characteristics of clinical manifestation by observing the tongue and hence suggest treatment for these patients. For a greater understanding of tongue diagnosis, which is a special diagnostic technique in TCM, let us take a look at the principles of diagnosis in TCM.

21.1.1 Salient features of TCM

The TCM diagnostics include examining the patient and collecting data related to health. Based on the theories of TCM, these data are sorted out, analyzed, and synthesized to determine the characteristics of clinical manifestation. As a result, we can identify the diseases and syndromes, and provide a basis for the treatment and prevention of diseases [1, 2, 7, 8, 9, 15, 16, 17, 18, 36, 37, 40, 41, ?].

During their medical practice, Chinese physicians from ancient times accumulated rich experience in diagnosis, which formed a comprehensive diagnostic system for TCM based on, *e.g.*, four diagnostic techniques (observation, auscultation and olfaction, interrogation and palpation) and syndrome differentiation. Recently, with the development of biometric technology, modern scientific approaches have been applied to conducting the four diagnostic techniques and syndrome differentiation, preserving the distinctive features of the original approaches.

According to the theories of TCM, the human body is an organic whole. The local disorder on the outside of a body can influence systems within it. Also, an internal disease can manifest itself on the exterior of the body. This means that the external manifestations can provide clues to the nature of the internal disease. Therefore, when making a diagnosis by TCM, internal pathological changes of a patient are detected mainly by means of the patient's self-sensation and external manifestations, which are known by physicians through their sensory organs. In *Grand Discussion on the Correspondence of Yin and Yang*, a chapter of *Plain Questions* [38], it is said: "To know the interior according to the exterior would ensure the correct diagnosis." This means that external changes can reflect the internal disease. It is pointed out more clearly in *Speculation from Exterior* (a chapter of *Spirit Pivot* [42]): "The obvious manifestation cannot be hidden because it does not leave *yin* and *yang*. The correct diagnosis can be made by observing and pulse feeling comprehensively. So the interior corresponding to the exterior is just like the shade always accompanying the figure. Physician can speculate the interior according to the external changes, as well as the exterior according to internal changes." Such an approach to clinical diagnosis, based on "knowing the interior by the exterior" still plays an important role in TCM clinical practice.

The diagnostic approach of "knowing the interior by the exterior" includes four diagnostic techniques: observation, auscultation and olfaction, interrogation and palpation. Each method has unique clinical functions and cannot be replaced by another. They must be simultaneously applied to ensure correct diagnosis.

The history of TCM dates back to thousands of years ago. *Internal Classic* is an important classical treatise on TCM. It summarizes the medical theories and medical experiences of Chinese people from the Spring and Autumn Periods, and the Warring States Period to the *Qin* and *Han* Dynasties. In the context of diagnostic methods, there are a lot of records about the four diagnostic methods including observation, listening and smelling, questioning and palpation, in the book. *Classic on Medical Problems*, written in the Three States Period, specially stressed the importance of pulse-reading in the diagnostic methods. From the *Tang* Dynasty to the *Jing* and *Yuan* Dynasties, syndrome differentiation developed even further. In the *Ming* and *Qing* Dynasties, the development of diagnostics was mainly in four aspects, *i.e.*, tongue inspection, questioning, palpation and the syndrome differentiation.

In recent decades, much research on the modernization of the four examining methods has been conducted comprehensively by means of acoustics, optics, magnetics, electricity, chemistry, physics and biomedical engineering, and some progress has been made.

Tongue observation is also termed tongue inspection. It is the method of observing the changes in the tongue body and tongue coating to analyze diseases. It is also a main component of observation. There are many records of tongue observation and the theories in *Internal Classic*. After that, a lot of medical works like *Treatise on Cold-Attack, Classic of Viscera, A Thousand Gold Worthy Prescriptions, An Official's Secret Prescriptions* recorded this method. *Records of Golden Mirror* was followed by a monograph of tongue examination during the *Yuan* Dynasty. Since then, many monographs have been published on the subject. Based on the experiences accumulated in medical practice for a long time, this unique method of clinical examination is effective even today.

21.1.2 Why tongue inspection?

Changes in the appearance of the tongue reflect inner visceral changes. Actually, the tongue is the orifice of the heart. Its stretching and retracting are the action of tendons, and reflect the function of the liver. The red small particles on the tip of the tongue are projections made up of heart *qi* and genuine-fire of life-gate. The white soft prickle-like hairs on the tongue surface are produced by lung *qi* with genuine-fire. The tongue fur (coating) is made up of steaming stomach *qi*. So, we can know the visceral conditions by observing the tongue [4, 12, 13, 14].

The tongue body and coating have their unique significance in diagnosis. The tongue body exhibits the condition of five *zang-viscera*, while the tongue coating shows that of six *fu-viscera*. By observing the tongue body, we can ascertain the deficiency or excess of genuine-*qi* and the severity of disease. In addition, we can judge the cold or heat of evils and the location of disease by inspecting tongue coating. Both tongue body and coating can reflect disease in the different aspects.

Among the four examination techniques, tongue observation is thought to be more reliable than others. There are three reasons:

(1) The tongue can be directly observed by the examiner's eyes, unlike the pulse which is covered by skin and hence can only be palpated, that is, felt indirectly and hence, not too distinctly
(2) The tongue is connected to the viscera internally and to the meridians externally, so both normal and morbid conditions show on tongue
(3) When evil enters the inner body, every change it produces will be embodied in tongue. Changes in inner conditions will be exhibited clearly by the changes in moisture and in the thickness of the coating

Thus changes in the tongue can reflect the conditions of genuine *qi* and evils, and the course of the disease. The significance of observing the tongue can be summarized as follows:

A : **To judge the exuberance or decline of the genuine *qi*:** The exuberance or decline of visceral *qi* and blood shows itself in the tongue. For example, a red and moist tongue signifies exuberance of *qi* and blood while a pale tongue is a sign of deficiency of both *qi* and blood. A white, thin and moist coating indicates an exuberant stomach *qi*, while no coating is due to the decline of stomach *qi*, or impairment of stomach *yin*.

B : **To distinguish the nature of disease:** Evils of different nature will produce different types of change in tongue. An absence of prickle on tongue surface together with a white and moist coating, or a bluish-black tongue without prickle, is due to cold evil. On the other hand, a red and dry tongue with yellow coating, or a red prickle tongue with yellow, thick and greasy coating, is due to warm or heat evil. A greasy or putrid coating indicates food retention. Blue maculae or spots on the tongue suggest blood stasis.

C : **To detect the location of disease:** In exogenous diseases, a thicker coating signifies a deeper location of disease. As an example, a thin coating suggests that the disease is in its initial stage, and it is an exterior syndrome; while a thick coating suggests that the evils have entered the interior of the body, and it is an interior syndrome. A crimson tongue means that disease is very deep and is critical.

D : **To infer the course of disease:** Changes in the tongue usually follow changes in genuine *qi* and evils, and disease location. We can infer the course of the disease by observing the tongue, especially in exogenous febrile diseases. For instance, the turning of coating from white to yellow, and from yellow to black is usually due to the transferring of evils from exterior to interior, or from cold to heat. It shows the deterioration of disease. If a moist coating turns dry, it is usually due to loss of body fluid resulting from heat. A change from dry to moist implies recovery of body fluid. The change of coating from thick to thin is a sign of improvement or cure.

However, it should be pointed out that sometimes the tongue is only slightly changed in some severe cases, or abnormal changes in the tongue are seen in normal people. So, tongue observation should only be used in combination with other examination techniques.

21.1.3 Relationship between tongue and viscera

The tongue is believed to be the sprout of the heart, which is the supreme monarch of all internal organs. So disease of viscera can influence not only the heart, but also the tongue.

The tongue is also called the out-show of the spleen, which dominates transportation and transformation. Therefore, the tongue is closely related to splenic function. Tongue coating has special relation with stomach *qi*. Zhang Xugu said: "In healthy body, there is a little thin coating like grass roots. It is the embodiment of stomach *qi* activity."

Meridians such as the three *yang* and three *yin* meridians of the foot, and the *Taiyang* and *Shaoyang* Meridians of the hand, are connected to the tongue. Shen Douyuan said: "All *qi* of meridians flow up to the tongue. So we can know the deficiency or excess, cold or heat of viscera and meridians by observing tongue."

Viscera have their representative areas on tongue surface because of their close relationship with the tongue. Although the ancient statements on this were different, the most popular one was from the Bihua's Medical Mirror: "The tip of tongue belongs to the heart, the middle to the spleen and stomach, the bilateral margins to the liver and gallbladder, the root to the kidney". Another way of expressing this is that tip corresponds to upper-*jiao*, the middle to the middle-*jiao* and the root to the lower-*jiao*, as shown in Fig. 21.1.

21.1.4 Main features of tongue diagnosis

Tongue diagnosis includes observing tongue body (texture) and tongue coating. Tongue texture is the main body of tongue made up of muscles and blood vessels. Tongue coating is the fur-like material on its upper surface. The normal tongue is characterized by a medium size, soft texture (neither tough nor tender), free movement, pink color, a thin and even white coating with moderate moistness. It is usually called "pink tongue with white and thin coating."

Observations on tongue coating and tongue texture have their own application areas, respectively. The observation of tongue texture is more important. Generally speaking, observation of tongue texture mainly probes the visceral conditions, while observation of tongue coating inspects the nature and location of disease and the clarity or turbidity of stomach *qi*. If the changes are only in tongue coating, the illness is mild. When changes develop from tongue coating to tongue texture, it indicates a worsening of disease. Changes in tongue texture are embodied in abnormal changes of the vitality, color, shape and movement of tongue body. On the basis of tongue analysis, we can apply biometric pattern recognition technology to TCM diagnosis. In fact, we have been developing an automated tongue diagnosis System, where a tongue image is captured by camera and its features are automatically extracted and analyzed for TCM diagnosis (see Fig. 21.2).

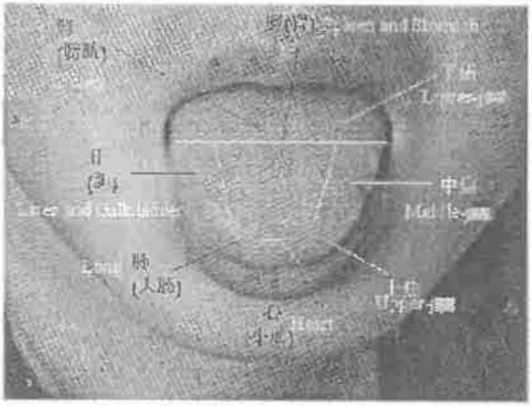


Fig. 21.1 Division of tongue surface

Our basic idea is to use image processing and pattern recognition technology for dealing with tongue images. Firstly, we use our capturing tool to get tongue images corresponding to different diseases and set up a large-scale database sorted by disease. Then, a tongue image is segmented by using an appropriate processing technology, and features are extracted from the corresponding disease. After analyzing the features, the disease can be diagnosed by biometric pattern recognltion technology proposed by us.

Although tongue diagnosis is an ancient and much-used procedure, there are no quantitative measures and standards for tongue features. Physicians have been determining disease by only his or her experience alone. Obviously this traditional method is not scientific. The significance of our work lies in the use of modern approaches to emulate age-old Chinese diagnostic techniques. We

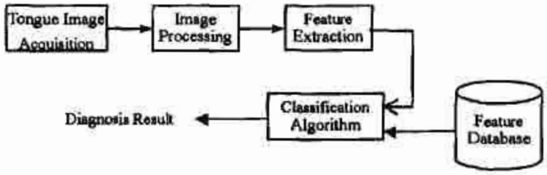


Fig. 21.2 The flowchart of a biometric tongue diagnosis system

have been developing a biometric biometric pattern recognition tongue diagnosis system which can capture tongue images, extract features and diagnose the disease (see Fig. 21.2). It is important to use such a system in early diagnosis and health care. Two key issues, namely, tongue image acquisition and tongue classification, are discussed in the following sections. The latter determines whether a tongue is healthy, and if not, which disease it signifies. It involves two operations: feature extraction and feature matching. To implement such a system, these problems should be solved beforehand.

21.2 Tongue image capturing

From traditional Chinese medicine theory, color and luster are important features of the tongue. Therefore, undistorted color acquisition is crucial. There are two ways of obtaining high-quality tongue images. One is via a closed box, as shown in Figure 21.3, which can avoid the influence of uneven illumination but is less user-acceptable. Now we have designed such a device and manufactured a prototype to be applied to our further research (see Fig. 21.3). The other is to use a high intensity white light, which can overwhelm the natural light to guarantee comparatively steady illumination. In both cases, a high quality camera is used to grab the tongue image. Then, the image is digitized and passed to the computer for further processing.

It is necessary to build a tongue image database with enough labeled samples. The recognition results of tongue diagnosis system will ultimately depend on these samples. The following steps are recommended:

(1) Set up a database with over 10,000 samples, including raw images of lingual surface and hypoglottis, and the extracted features of tongue. These samples should be obtained from persons of different sex, age, and disease in terms of predefined proportion. A tongue image database on such a scale should be good enough to build an automated tongue diagnosis system.
(2) Build classified index of the tongue database in terms of sex, age, disease, symptoms and features.
(3) Select different types of tongue images in terms of the category of disease and evaluate their input qualities and representative samples.

Thousands of tongue images have been obtained by us from the hospitals, which are classified by different diseases. Table 21.1 provides a summary of partial tongue images that we have obtained.

21.3 Segmentation of tongue images

Before tongue diagnosis is done, the exact portion of the image corresponding to the tongue must be extracted from the true-color tongue image captured by a camera with a resolution of 640×480. However, general pixel-based edge detectors fail to segment the tongue because of the complexity of the image and discontinuity of the tongue's edge.

As an efficient image segmentation technique, active contours (snakes) were originally proposed by Kass [10]. They advocate that the presence of an edge depends not only on the gradient at a specific point but also on the spatial distribution. Snakes incorporate this global view of edge detection by assessing continuity and curvature, combined with the local image gradient. The principal advantage over other edge-detecting techniques is the integration of image data, an initial estimated location, desired contour properties and knowledge-based constraints. Our technique uses an en-

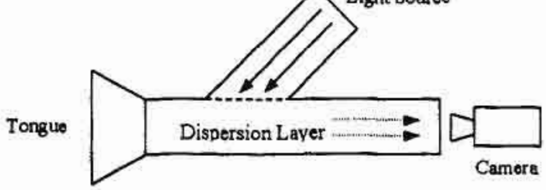


Fig. 21.3 The tongue capturing system

ergy minimization framework. Snakes are curves defined within an image domain, which can move under the influence of internal forces coming from within the curve itself and external forces computed from the image data. However, due to the shrinking nature near strong features (that generates large external strength) and the sensitivity to initial positions, many efforts on initialization, modification and combination of active contour with different techniques are conducted to improve the behaviors of snakes [3, 5, 11, 39].

In the following section, we present a new active contour, which we call *Time-adaptive Snakes*, by introducing temporal variables into the parameters of internal forces. The coarse edge can be obtained quickly in the beginning, and refined according to a pre-designed template near the true edge, which is implemented by dynamically changing the trade-off between internal and external forces. It is clear that this method can guarantee an accurate contour which is crucial to TCM.

21.3.1 Time adaptive snakes (TAS)

A traditional snake is a curve $v(s) = [x(s), y(s)], s \in [0, 1]$, that moves through the spatial domain of an image to minimize the energy functional

$$E_{snake} = \int_0^1 \frac{1}{2}(\alpha(s)|v'(s)|^2 + \beta(s)|v''(s)|^2) + E_{ext}(v(s))ds, \qquad (21.1)$$

where $\alpha(s)$ and $\beta(s)$ are weighting parameters that control the snake's tension and rigidity, respectively, and $v'(s)$ and $v''(s)$ denote the first and second derivatives of $v(s)$ with respect to s. The first-order term makes the snake act like a membrane and the second-order term makes it act like a thin plate. The external energy function E_{ext} is derived from the image so that it takes on its smaller values at the features of interest, such as boundaries. The final solution is a snake that minimizes (21.1). Thus, a snake can be regarded as a mathematical energy-minimizing curve or as a physical chain structure driven by forces that come from a user interface.

Apparently, the two regularization parameters, which are often set constant

Table 21.1 A summary of partial tongue images in our database

Disease	Number	Disease	Number
Health	2500	Ileus	135
Appendicitis	150	Gall-stone	165
Bladder tumor	120	Cholecyst polyps	34
Intestines perforation	86	Cholecystitis	65
Peritonitis	150	Liver cancer	46
Hypertension	245	Marrow cancer	35
Gastricism	145	Pancreas disease	122
Lung cancer	87	Emphysema	79
Leucocythemia	56	Lung-heart disease	86
Hepatocirrhosis	54	Coronary heart disease	76
Hepatitis	78	Thyroiditis	66

(that is position invariable), can significantly influence the solution to (21.1). Setting $\alpha >> \beta$ emphasizes regularization, yielding strongly model-driven solutions which are robust to noise, but imprecise. In contrast, setting $\beta >> \alpha$ enables the snakes to effectively capture boundary discontinuity, but it also makes them sensitive to noise. Therefore, constant values of α and β will not work.

To solve this problem, we introduce a temporal variable into α, β and the control parameter of external energy γ to produce a new class of snakes which we call *time-adaptive snakes.* Time-adaptive snakes are series of parameterized curves that minimize the following energy functional:

$$E_{snake} = \int_0^1 \frac{1}{2}(\alpha(s,t)|v'(s)|^2 + \beta(s,t)|v''(s)|^2) + \gamma(s,t) + E_{ext}(v(s))ds, \tag{21.2}$$

Thus, we can impose a dynamic control upon the curves according to different requirements, by changing the associated weights of the three terms. For example, when trying to find the boundary of an object within the image domain we can use a typical external energy, $E_{ext}(x,y) = -|\nabla I(x,y)|^2$, designed to lead an active contour toward step edges, where $I(x,y)$ is a gray-level image. At the beginning, we usually want the snake to quickly approach the desired features (a step edge) with little regard to shape properties – continuity and smoothness. In order to do it well, $\gamma(s,t) >> \alpha(s,t), \beta(s,t)$ can be set. On the contrary, we are able to refine the boundary, according to *a priori* shape information, by setting internal weighting parameters to large values after approximate location of edge.

21.3.2 Combined models for tongue segmentation

There are three genres [6] of contour detection:

- Classic *pixel orientated* image processing, such as threshold, gradient-based segmentation and edge filters
- *Active contours* like snakes which rely on local image information
- *Deformable templates* which use global image information

Our approach is to combine all three methods together. First, we construct a deformable template of the tongue; then, we let a snake approach the tongue's boundary freely; finally, we refine the contour by forcing the snake to deform according to the template.

Deformable templates

We propose a double-parabola deformable template of the tongue. The two parabolas describe the top and the bottom curves of the tongue, respectively (see Fig. 21.4). They are formulated by the equations: $y = ax^2 + bx + c$ or $y = Ax$ where A is the vector of coefficients and can be calculated using three input points that are not on the same line. The deformable template

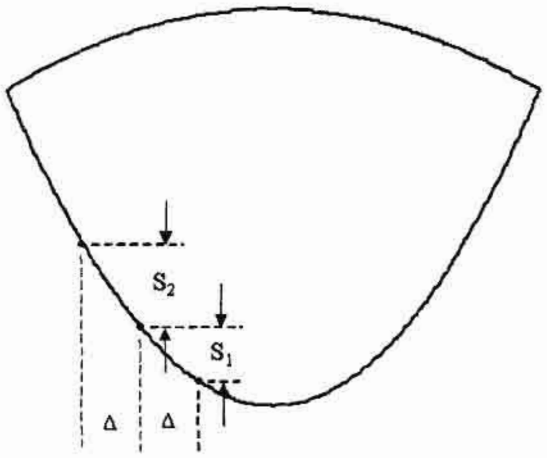


Fig. 21.4 A template for the tongue

tries to minimize its energy, called *template energy*. The definition of the energies and the order in which the parameters of the deformable template are changed is crucial for the final result of the energy reduction. We apply the deformable template of the tongue to initialize, in order to improve the shape of the tongue snake model, incorporating the global shape knowledge embodied in the deformable template.

TAS for tongue segmentation

Time-adaptive snakes (TAS) can be regarded as a physically-based energy minimizing splines that move under the influence of internal forces, predefined template forces and image forces computed from the image data.

The traditional internal energy is a weighted sum of first-order and second-order derivatives of the curve vector. The first-order term makes the snake act like a membrane and the second-order term makes it act like a thin plate. These cause a shrinking closed and open curves to a point without the support of external forces. Modification of the continuity and curvature constraints can remove this unwanted contraction force and take the template constraints into account. To implement these properties we divide the internal energy term into two parts: horizontal forces and vertical forces.

Using a discrete contour which is a set of points $\vec{v}(n) = [\vec{x}(n), \vec{y}(n)]$, $n \in \{1, 2 \cdots, N\}$, where N is the number of points, the horizontal forces are represented by the continuity term

$$E_{h\_int}(v(n)) = \alpha(n,t)|x(n-1) + x(n+1) - 2x(n)|. \tag{21.3}$$

The horizontal internal forces ensure that contour points are evenly spaced horizontally.

The vertical forces are developed from the deformable template defined earlier (Fig. 21.4). Unlike the original smoothing forces [10] which have a potential linearizing effect, the template-based energy combined with the horizontal internal forces, causes the contour to take a specific shape. In the case of a parabola formalized by $[\tilde{x}(n), \tilde{y}(n)]$ for $n = 1, 2, \cdots, N$,

$$(\tilde{y}(n+1) - \tilde{y}(n)) - (\tilde{y}(n) - \tilde{y}(n-1)) = 2\alpha\Delta^2, \tag{21.4}$$

where α is the coefficient of the quadratic term in the template equation, and Δ is the increment along x-axis. Thus, the vertical forces can be defined as

$$E_{v\_int}(v(n)) = \beta(n,t)|\Gamma y(n) - 2\alpha\Delta^2|, \tag{21.5}$$

where $\Gamma y(n) = y(n+1) + y(n-1) - 2y(n)$. Then the internal forces can be calculated by

$$E_{int} = \sigma_{n=1}^{N} E_{h\_int}(v(n)) + E_{v\_int}(v(n)), \tag{21.6}$$

that pulls each node on the contour towards its estimated position. The image energy is computed from the image intensity gradient magnitudes, G_n, of all points and normalized by

$$E_{image,n} = (G_{min,n-G_n})/(G_{max,n} - G_{min,n}), \tag{21.7}$$

where $G_{min,n}$ and $G_{max,n}$ are the minimum and maximum in the neighborhood of point n.

Thus, a combined model is obtained by incorporating all three forces into the energy formulation. A combined model has remarkable advantages – the deformable template acts like an interactive user who, having global information about possible shapes, controls and corrects the shape of the snake while it moves toward local minimum. However, in using constant-driven techniques, there is inevitable difficulty in parameter determination. When the driving force is too high the contour will be driven over the feature of interest, while when too low, it may cause the contour to become entrapped in a weak local minimum. In order to avoid this, we advocate the use of time-adaptive weights that keeps changing during the processing. For example, we can set the parameter of template force $\beta(s,t) = \sin(wt + \theta)$, where w and θ can be evaluated differently according to specific applications.

21.3.3 Segmentation results

To demonstrate the performance, the new technique is compared with a conventional single Kass snake [10] using preprocessed tongue images. We use the red channel of the original image as the tested image which can be regarded as gray-level image, since red is the dominating color in tongue domain.

The image functional used in the tests was the edge-based $E_{image}(v) = |\nabla I(v)|$. In order to increase the capture range of snake at the beginning and locate precise edges, we introduce a time-varying parameter into the image functional formula as follows:

$$E_{image,t}(v) = -|\nabla(G_{\sigma(t)}(v) * I(v))|, \tag{21.8}$$

where $G_{\sigma(t)}$ is a two-dimensional Gaussian function with time-varying standard deviation that takes relative large values to blur the image so as to enlarge the capture range of snake and becomes smaller as the snake approaches the true edge, to sharpen the image. $\sigma(t)$ can be any linear or nonlinear monotone incremental function. Another advantage of using $G_{\sigma(t)}$ is to make the algorithm more robust in the presence of noise by preventing the snake from getting trapped at local minima.

The parameter $\beta(n,t)$ in (21.5) is the weight of template forces. In the test, we make it position-independent and a Gaussian function with respect to the temporal variable: $\beta(t) = G_{\sigma,\mu}(t)$.

Fig. 21.5 (a) shows the Kass snake result with $\beta = 0.2$, $\gamma = 1.0$ and $\alpha >> \beta, \gamma$; Fig. 21.5 (b) shows the Kass snake result with $\beta = 0.8$, other parameters remaining the same. Fig. 21.5 (c) shows the TAS result using

$\beta(t) = G_{\sigma,\mu}(t)$ with $\sigma = 0.1, \mu = 0.85$. (Template and initial position of snake in green color; final result of snake in red color). It can be seen from

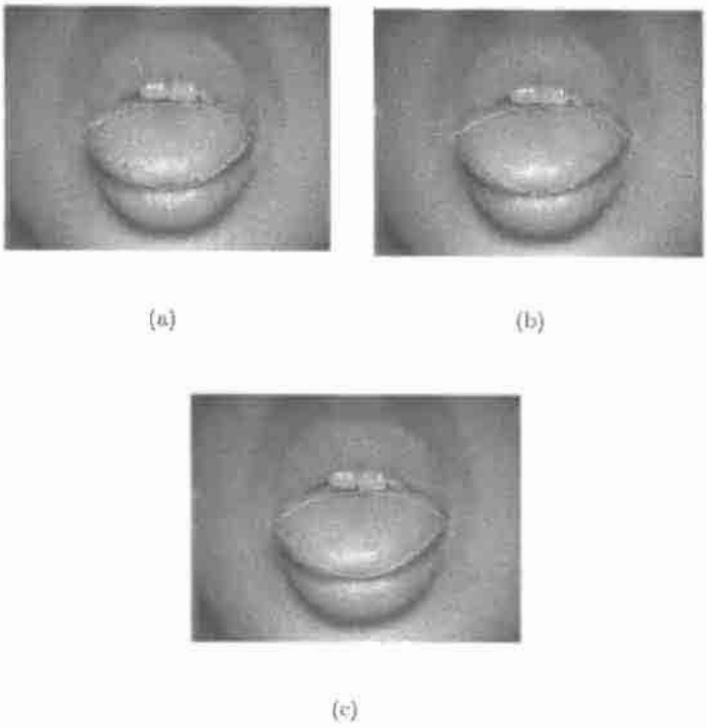


Fig. 21.5 (a) (b) Kass snakes with different parameter values, and (c) TAS snake

Fig. 21.5 that the performance of the TAS snake is superior to the traditional Kass approaches because the former can more precisely describe the edge of tongue which is crucial to tongue diagnosis in TCM.

Note that many other image preprocessing techniques can be used to improve the performance of the snake algorithm further, such as median filtering, morphological filtering and color space transformation.

21.4 Tongue feature extraction

We have two feature extraction approaches, namely, expert knowledge-based feature extraction, and statistical method. Various features, with the guidance of Traditional Chinese Medicine experts, can be extracted from the tongue, including geometrical features, color and lustre features, tongue coating, and lingual vena. Efficient and accurate extraction of these features is the key to the accuracy of tongue diagnosis.

According to famous tongue diagnosis expert, Prof. Li Naiming, there are many features, as listed below:

- **Tongue Color**: pale, pink, red, crimson, purple, bluish purple, dark purple, blue, and so on
- **Tongue coating**: thinness and white, white, thickness and white, thinness and yellow, yellow, gray, black, exfoliation and nothing
- **Tongue visible material**: speckle and spot, streak, knurl, excrescence, ulcer, tooth print, tumor, blue print, tongue body fat or emaciated, and so on
- **Tongue vena**: diameter, circuity, exaggeration, and branch
- **Tongue vena color**: pink, dark red, red, purple, crimson.
- **Changed part**: tip, margins, root, middle

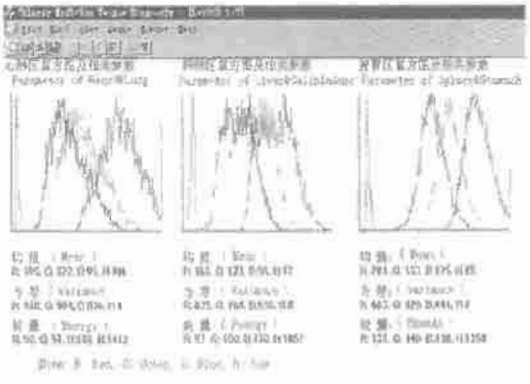


Fig. 21.6 Examples of statistical features

The statistical approach does not utilize experts' experience but is promising. For example, we can get statistical features such as mean, variance, energy and so on, as shown in Fig. 21.6.

According to our experiment, there are some features which are good for tongue diagnosis. The following section summarizes the detection and application of these features.

21.4.1 The detection of glistening points

The moistness of tongue body is an important pathological feature, which is represented by percentage of glistening points in the tongue image. The glistening points are much brighter in the image, and their gray value is more than a threshold. For example, the gray value of glistening points is more than 200 in many cases. The black points are glistening points in the image, shown as Fig. 21.7. We define

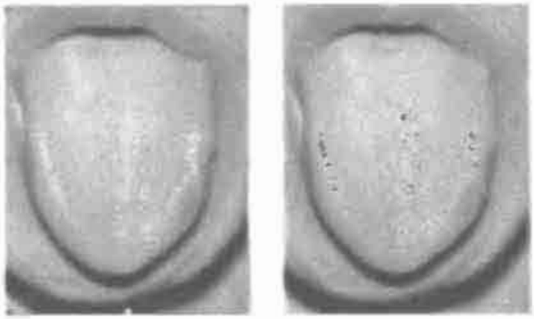


Fig. 21.7 (a) original image (b) The black points are glistening points

$$RflxPcnt = \frac{nRflxPnt}{nTtlPxl}, \tag{21.9}$$

where, $RflxPcnt$ is the ratio of glistening points to the total tongue area in image, $nRflxPnt$ is the number of glistening points, $nTtlPxl$ is the total number of pixels in the tongue body.

21.4.2 Mean of chroma

The color feature of tongue body and coating is crucial for diagnosis. For example, healthy people have a pink tongue body with a thin, white coating, while a person suffering from pancreatitis has a blue or light blue or purple tongue body. As a result of illumination and other environmental factors, it is not quite robust to use RGB color space to represent the color of tongue body and coating. Therefore HIS color space is employed and especially chroma (hue) is important. Fig. 21.8 shows the hue distribution curve of tongue body of health people, and of people suffering from pancreatitis.

Aver1 is the statistical mean of (tongue body) hue of healthy people while Aver2 is one of pancreatitis patients in Fig. 21.8. According to our experiment Aver1 is 11 and Aver2 is 150.

$$HueAver = \frac{1}{N}\sum_{k=1}^{N} HueVal_k. \tag{21.10}$$

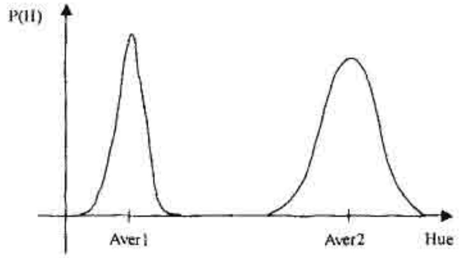


Fig. 21.8 The distribution of mean of chroma

21.4.3 Chroma variance

Mean of chroma can correctly reflect the distribution of some simple cases. For example, if there is little coating and petechia on the surface of tongue, then one color is dominant. When there is a thick coating or obvious petechia on the tongue surface, then the mean of chroma cannot reflect the color distribution of tongue surface correctly, as can be seen from Fig. 21.9. According to Fig. 21.9, when there is coating on the tongue surface, the chroma distribution is obviously bimodal. Here chroma variance is a better indicator of the distribution. Therefore, we can infer that there is coating and obvious petechia on

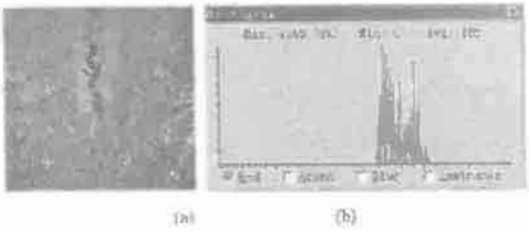


Fig. 21.9 (a) Original image and (b) The Histogram of red color

the tongue surface when chroma variance exceeds a threshold.

$$HueVari = \frac{1}{N}\sum_{k=1}^{N}(HueVal_k - HueAver)^2. \qquad (21.11)$$

Chroma based clustering analysis

When chroma variance is much higher (more than $ThresHV$), there may be several cases:

(1) Presence of tongue coating
(2) Presence of a large area of petechia, the color being close to that of tongue body
(3) Presence of a small area of petechia, the color being much different from that of the tongue body

Therefore the correct answer cannot be obtained by just the chroma variance of the tongue surface. For example, the first and third cases above are indistinguishable. For solving this problem, chroma-based clustering analysis is employed. Considering the difference of mean and variance of chroma before and after clustering, the following analysis is done:

(1) Calculate the chroma Variance $HueVari$
(2) If $HueVari < ThresHV$ then there is no obvious disease on surface of tongue
(3) Else use c-means clustering algorithm to classify the pixels of tongue surface into two classes and calculate their mean and variance respectively, $HueAver1$, $HueAver2$, $HueVari1$ and $HueVari2$

(4) If $|HueAver1 - HueAver2| < ThresHAD$ and $(HueVari1 + HueVari2) > \alpha HueVari$ then there is a little petechia; else, there is coating on surface of tongue.

Experimental results obtained on applying chroma based c-means clustering algorithm to tongue images are shown in Fig. 21.10, where (a) original image; (b) tongue coating area; (c) tongue body area and (d) chroma distribution are shown.

21.5 Tongue classification

Classification involves assigning a tongue pattern to a certain category according to its formation. According to the Traditional Chinese Medicine theory, the entire tongue can be artificially divided into several non-overlapping regions which correspond to different organs respectively. Therefore, the feature matching process should be carried out region by region. In our system, some common diseases can be diagnosed correctly, as shown in Fig. 21.11. Feature

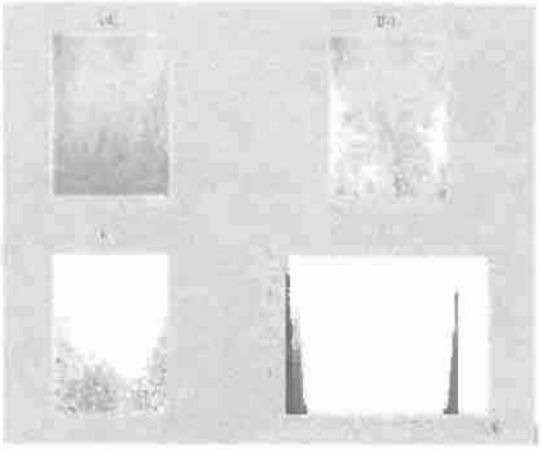


Fig. 21.10 Experimental results of clustering

matching is actually a classification procedure, so the various classification approaches based on statistical pattern classification, neural networks, belief networks and fuzzy theory can be applied to tongue image matching. The issue of feature matching of tongue image can be further described as follows:

(1) Development of efficient approaches for disease feature matching based on statistical pattern classification, neural networks, belief networks and fuzzy theory
(2) Comparison of performances of various approaches for disease matching
(3) Investigation of the effect of the different identification functions on identification results, based on the different approaches for tongue feature matching

Fig. 21.11 An example of diagnosis by biometric TCM system

21.6 Conclusions

Tongue diagnosis is one of the most widely used diagnostic procedures in TCM, and has significant clinical applications. Tongue diagnosis can not only detect pathological changes and the positions of the changes, but also find what modern medical equipment cannot do, namely, the degrees of the pathological changes and the functions of the viscera. It can also propose the treatment in time. In the modern world, there is an ever-growing need to develop such an automated tongue diagnosis system based on pattern recognition, that fully exploits the utility of tongue diagnosis as a clinical diagnostic aid. As a first attempt in medical diagnosis, we apply biometric technology to tongue diagnosis.

The effectiveness of our proposed methods is demonstrated in the chapter.

References

[1] D. Bensky and R. Barolet. *Chinese Herbal Medicine: Formulas and Strategies.* Eastland Press, Seattle, Washington, 1993.

[2] D. Bensky and R. Barolet. *Chinese Herbal Medicine: Materia Medica (Revised Edition).* Eastland Press, Seattle, Washington, 1993.

[3] L.D. Cohen. On active contour models and balloons. *CVGIP: Image Understanding*, vol. 53, pp. 211–218, 1991.

[4] T.T. Den and Z. Q. Guo. *Diagnostics of Traditional Chinese Medicine.* Shanghai Science and Technology Press, Shanghai, 1984.

[5] S.R. Gunn and M.S. Nixon. A robust snake implementation: A dual active contour. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 63–68, 1997.

[6] S. Horbelt and J.L. Dugelay. Active contours for lipreading-combining snakes with templates. *15th GRETSI Symposium on Signal and Image Processing*, pages 18–22, September 1995.

[7] S.L. Huang. *Research on Pulse of TCM.* People's Health Press, 1995.

[8] Isselbacher. *Harrison's Principles of Internal Medicine (13th Ed.).* McGraw-Hill, New York, 1994.

[9] Jones and Robert. *Acupuncture Techniques.* North Atlantic Books, Berkeley, 1996.

[10] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *Procedings of the First International Conference on Computer Vision*, pages 259–269, 1987.

[11] K.F. Lai and R.T. Chin. *On regularization, formulation and initialization of the active contour models (snakes)*, pp.542–545. Asian Conf. on Computer Vision, 1993.

[12] N.M. Li. *Illustration of Tongue Pictures of Yu-syndrome.* Heilongjiang Sceince and Technology Press, Heilongjiang, 1990.

[13] N.M. Li. *The Great Integrated of Chinese Tongue Diagnosis.* Academic Press, Beijing, 1994.

[14] N.M. Li and Y. F. Wang. *Diagnosis by Observing Tongue.* Heilongjiang Sceince and Technology Press, Heilongjiang, 1987. Paradigm Publisher.

[15] Z.X. Long. *Diagnostics of Traditional Chinese Medicine.* Academic

Press, Beijing, 1998.

[16] Macciocoa and Giovanni. *Foundations of Chinese Medicine.* Churchill Livingston, New York, 1989.

[17] Macciocoa and Giovanni. *The Practice of Chinese Medicine.* Churchill Livingston, New York, 1989.

[18] Macciocoa. *Tongue Diagnosis in Chinese Medicine.* Eastland Press, 1995.

[19] The study of qualitative and quantitative analysis for Chinese medical tongue diagnosis using high resolution color ccd. www.ccmp.gov.tw/ 4e/ 86-052e.htm.

[20] Traditional Chinese medicine. http://www.healthy.net/ CLINIC/ therapy/ Chinmed/ Index.asp.

[21] The Journal of Chinese Medicine. http://www.pavilion.co.uk/ jcm.

[22] Chinese medicine - the new world of holistic health. http://detox.j12.net/ chronicho/ chinese-medi-cine.phtm.

[23] Medicine and acupuncture in Canada. http://www.medicinechinese.com/.

[24] Birmingham Center for Chinese medicine. http://www.bccm.freeserve.co.uk/.

[25] Chinese traditional medicine. http://www.libraries.wayne.edu/ shiffman/ altmed/ china/ chin...

[26] Traditional Chinese medicine. http://loki.stockton.edu/ g̃ilmorew/ consorti/ 2peasia.htm.

[27] Traditional Chinese medicine correspondence programs. http://www.bta.net.cn/ tcm/ tcm/ chinatcm.-htm.

[28] Traditional Chinese medicine. http://reference.cd-rom-directory.com/ cdrom-2.cdprod1/ 010.

[29] Yahoo! Health Traditional Medicine Chinese. http://asia.yahoo.com/ Health/ Traditional-Medicine/ Chinese.

[30] Traditional Chinese medicine. http://www.iospress.nl/ html/ tcm.html.

[31] Institute of traditional Chinese medicine. http://www.mitcm.org/.

[32] Traditional Chinese medicine. http://www.tradedaily.com/ tcm/ big.htm.

[33] Traditional Chinese medicine. http://www.scn.org/ fremont/ acu/ chinese-med.html.

[34] Shanghai university of traditional Chinese medicine. http://www.csc.edu.cn/ foreign-stu/ shang-hai/ shzyy1/ shzyy.htm.

[35] Chinese traditional medicine. http://www.chalmers.com.au/ China-House/ Pc/ book3.html.

[36] D. Molony. *The American Association of Oriental Medicine's Complete Guide to Chinese Herbal Medicine.* 1998.

[37] Rui and Chun Ji. *Acupuncture Case Histories from China.* Eastland Press, Seattle, 1988.

[38] B. Wang. *Plain Question in Internal Classic.* People Health Press, Beijing, 1979.

[39] C.Y. Xu and J.L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Trans. on Image Processing*, vol. 7, pp. 359–369, 1998.

[40] X.Yang and S.G. Cheng. *Hand Diagnosis with Shape and Color.* Tian Jin Science and Technology Press, Tian Jin, 1998.

[41] H. H. Yin and B. N. Zhang. *Elementary Theory of Traditional Chinese Medicine.* Shanghai Science and Technology Press, Shanghai, 1984.

[42] Y.G. Zhang. *Spirit Pivo in Internal Classic.* Science and Technology Health Press, Shanghai, 1958.

Index

About the Editors

Sankar K. Pal is a *Professor* and *Distinguished Scientist* at the Indian Statistical Institute, Calcutta. He is also the *Founding Head* of Machine Intelligence Unit. He received the M. Tech. and Ph.D. degrees in Radio physics and Electronics in 1974 and 1979 respectively, from the University of Calcutta. In 1982 he received another Ph.D. in Electrical Engineering along with DIC from Imperial College, University of London. He worked at the University of California, Berkeley and the University of Maryland, College Park during 1986-87 as a *Fulbright Post-doctoral Visiting Fellow*; at the NASA Johnson Space Center, Houston, Texas during 1990-92 and 1994 as a *Guest Investigator* under the *NRC-NASA Senior Research Associateship program*; and at the Hong Kong Polytechnic University, Hong Kong in 1999 as a *Visiting Professor*. He served as a *Distinguished Visitor of IEEE Computer Society (USA)* for the *Asia-Pacific Region* during 1997-99.

Prof. Pal is a *Fellow* of the IEEE, USA, Third World Academy of Sciences, Italy, and all the four National Academies for Science/Engineering in India. His research interests includes Pattern Recognition, Image Processing, Data Mining, Soft Computing, Neural Nets, Genetic Algorithms, and Fuzzy Systems. He is a co-author/co-editor of seven books including *Fuzzy Mathematical Approach* to *Pattern Recognition*, John Wiley (Halsted), N.Y., 1986, *Neuro-Fuzzy Pattern Recognition : Methods in Soft Computing*, John Wiley, N.Y. 1999; and has about three hundred research publications.

He has received the *1990 S. S. Bhatnagar Prize* (which is the most coveted award for a scientist in India), *1993 Jawaharlal Nehru Fellowship*, *1993 Vikram Sarabhai Research Award*, *1993 NASA Tech Brief Award*, *1994 IEEE Trans. Neural Networks Outstanding Paper Award*, *1995 NASA Patent Application Award*, *1997 IETE - Ram Lal Wadhwa Gold Medal*, *1998 Om Bhasin Foundation Award*, *1999 G. D. Birla Award for Scientific Research*, the *2000 Khwarizmi International Award (1st winner) from the Islamic Republic of Iran*, and the *2001 Syed Husain Zaheer Medal* from Indian National Science Academy.

Prof. Pal is an *Associate Editor*, IEEE Trans. Neural Networks (1994-98), Pattern Recognition Letters, Int. J. Pattern Recognition and Artificial Intelligence, Neurocomputing, Applied Intelligence, Information Sciences, Fuzzy Sets and Systems, and Fundamenta Informaticae; a *Member, Executive Advisory Editorial Board*, IEEE Trans. Fuzzy Systems, Int. Journal on Image and Graphics, and Int. Journal of Approximate Reasoning; and a *Guest Editor* of many journals including the IEEE Computer.

Amita Pal (Pathak) obtained her Master of Science degree in Statistics from the University of Calcutta in 1981. She was awarded a Ph.D. degree by the Indian Statistical Institute in 1993. In 1994, she joined as a lecturer in the Applied Statistics Unit of the Indian Statistical Institute, where she is currently an Associate Professor. She visited the Department of Mathematics of the Imperial College of Science, Technology and Medicine, London as a UNDP fellow in 1994. Her research interests are mainly in the areas of Pattern Recognition, Machine Learning and Bayesian Image Analysis.