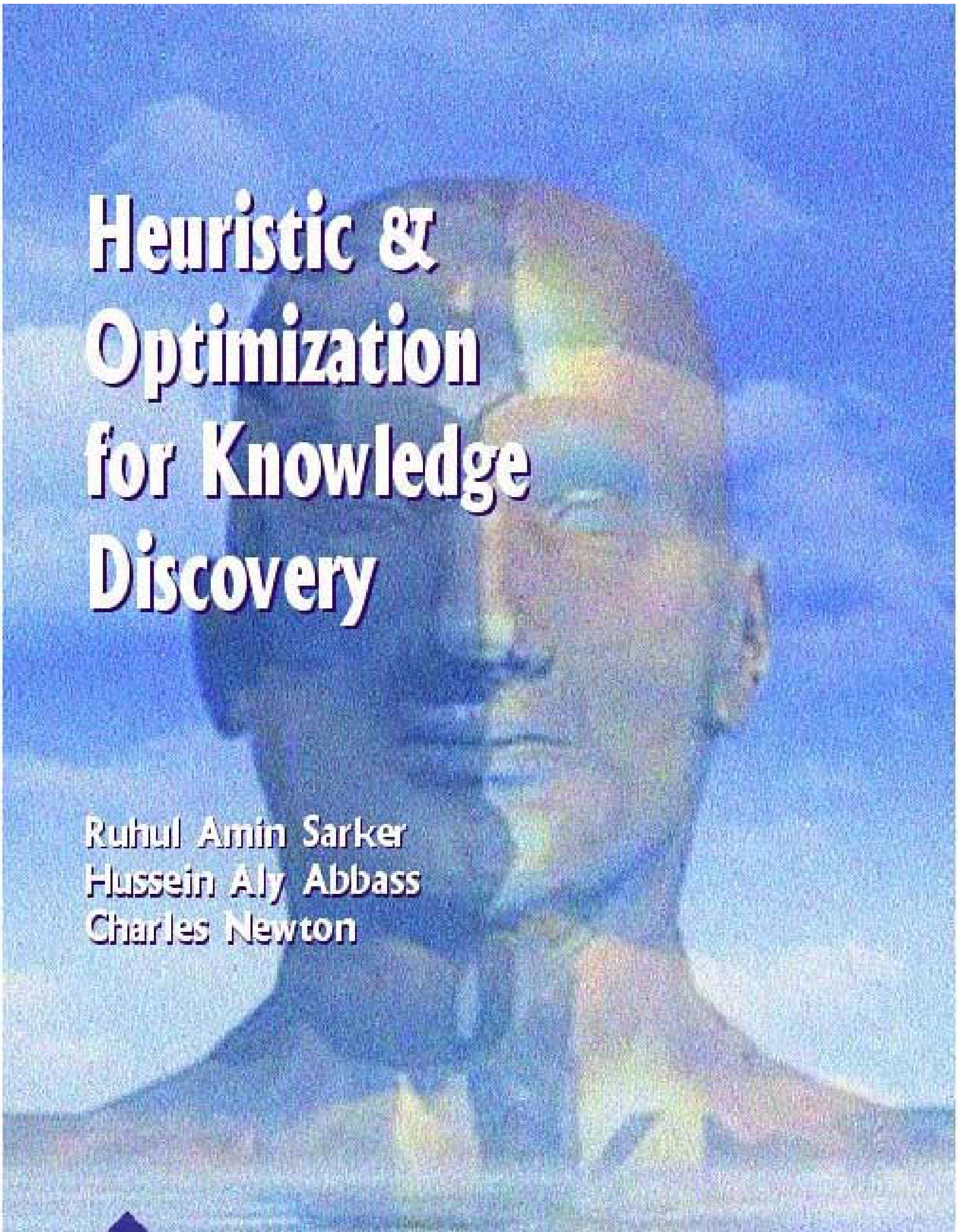


Heuristic & Optimization for Knowledge Discovery

Ruhul Amin Sarker
Hussein Aly Abbass
Charles Newton

IDEA GROUP PUBLISHING

Heuristics and Optimization for Knowledge Discovery

Ruhul A. Sarker

Hussein A. Abbass

Charles S. Newton

University of New South Wales, Australia

Idea Group Publishing

Information Science Publishing

Hershey • London • Melbourne • Singapore • Beijing

Acquisitions Editor:	Mehdi Khosrowpour
Managing Editor:	Jan Travers
Development Editor:	Michele Rossi
Copy Editor:	Elizabeth Arneson
Typesetter:	LeAnn Whitcomb
Cover Design:	Tedi Wingard
Printed at:	Integrated Book Technology

Published in the United States of America by
Idea Group Publishing
1331 E. Chocolate Avenue
Hershey PA 17033-1117
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Library of Congress Cataloging-in-Publication Data

Heuristics and optimization for knowledge discovery / [edited by] Ruhul Sarker, Hussein Abbass, Charles Newton.
p. cm.

Includes bibliographical references and index.
ISBN 1-930708-26-2 (cloth)
1. Heuristic programming. 2. Combinatorial optimization. I. Sarker, Ruhul. II. Abbass, Hussein. III. Newton, Charles, 1942-

T57.84 .H48 2001
006.3--dc21 2001039720

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

Heuristic and Optimization for Knowledge Discovery

Table of Contents

Preface

The advancement in data collection, storage, and distribution technologies has far outpaced computational advances in techniques for analyzing and understanding data. This encourages researchers and practitioners to develop a new generation of tools and techniques for data mining (DM) and for knowledge discovery in databases (KDD). KDD is a broad area that integrates concepts and methods from several disciplines including the fields of statistics, databases, artificial intelligence, machine learning, pattern recognition, machine discovery, uncertainty modeling, data visualization, high performance computing, optimization, management information systems, and knowledge-based systems.

KDD is a multistep iterative process. The preparatory steps of KDD include data selection and/or sampling, preprocessing and transformation of data for the subsequent steps of the process. Data mining is the next step in the KDD process. Data mining algorithms are used to discover patterns, clusters and models from data. The outcomes of the data mining algorithms are then rendered into operational forms that are easy for people to visualize and understand.

The data mining part of KDD usually uses a model and search based algorithm to find patterns and models of interests. The commonly used techniques are decision trees, genetic programming, neural networks, inductive logic programming, rough sets, Bayesian statistics, optimisation and other approaches. That means, heuristic and optimisation have a major role to play in data mining and knowledge discovery. However, most data mining work resulting from the application of heuristic and optimisation techniques has been reported in a scattered fashion in a wide variety of different journals and conference proceedings. As such, different journal and conference publications tend to focus on a very special and narrow topic. It is high time that an archival book series publishes a special volume which provides critical reviews of the state-of-art applications of heuristic and optimisation techniques associated with data mining and KDD problems. This volume aims at filling in the gap in the current literature.

This special volume consists of open-solicited and invited chapters written by leading researchers in the field. All papers were peer reviewed by at least two recognised reviewers. The book covers the foundation as well as the practical side of data mining and knowledge discovery.

This book contains 15 chapters, which can be categorized into the following five sections:

- Section 1: Introduction
- Section 2: Search and Optimization
- Section 3: Statistics and Data Mining
- Section 4: Neural Networks and Data Mining
- Section 5: Applications

In the first chapter, an introduction to data mining and KDD, and the steps of KDD are briefly presented. The DM tasks and tools are also provided in this chapter. The role of heuristic and optimisation techniques in KDD are also discussed.

Section 2 contains Chapters 2 to 6. Chapter 2 presents an algorithm for feature selection, which is based on a conventional optimization technique. The effectiveness of the proposed algorithm is tested by applying it to a number of publicly available real-world databases. Chapter 3 reports the results obtained from a series of studies on cost-sensitive classification using decision trees, boosting algorithms, and MetaCost which is a recently proposed procedure that converts an error-based algorithm into a cost-sensitive algorithm. The studies give rise to new variants of algorithms designed for cost-sensitive classification, and provide insight into the strengths and weaknesses of the algorithms. Chapter 4 presents an optimization problem that addresses the selection of a combination of several classifiers such as boosting, bagging and stacking. This is followed by the discussion of heuristic search techniques, in particular, genetic algorithms applied to automatically obtain the ideal combination of learning methods for the stacking system. Chapter 5 examines the use of the Component Object Model (COM) in the design of search engines for knowledge discovery and data mining using modern heuristic techniques and how adopting this approach benefits the design of a commercial toolkit. The chapter also describes how search engines have been implemented as COM objects and how the representation and problem components have been created to solve rule induction problems in data mining. Chapter 6 discusses the possibility of applying the logical combinatorial pattern recognition (LCPR) tools to the clustering of large and very large mixed incomplete data (MID) sets. This research is directed towards the application of methods, techniques and in general, the philosophy of the LCPR to the solution of supervised and unsupervised classification problems. In this chapter, the clustering algorithms GLC, DGLC, and GLC+ are introduced.

Chapters 7 to 9 comprise Section 3. Chapter 7 introduces the Bayes' Theorem and discusses the applicability of the Bayesian framework to three traditional statistical and/or machine learning examples: a simple probability experiment involving coin-tossing, Bayesian linear regression and Bayesian neural network learning. Some of the problems associated with the practical aspects of the implementation of Bayesian learning are then detailed, followed by the introduction of various software that is freely available on the Internet. The advantages of the Bayesian approach to learning and inference, its impact on diverse scientific fields and its present applications are subsequently identified. Chapter 8 addresses the question of how to decide how large a sample is necessary in order to apply a particular data mining procedure to a given data set. A brief review of the main results of basic sampling theory is followed by a detailed consideration and comparison of the impact of simple random sample size on two well-known data mining procedures: naïve Bayes classifiers and decision tree induction. The chapter also introduces a more sophisticated form of sampling, disproportionate stratification, and shows how it may be used to make much more effective use of limited processing resources. Chapter 9 shows how the Gamma test can be used in the construction of predictive models and classifiers for numerical data. In doing so, the chapter also demonstrates the application of this technique to feature selection and to the selection of the embedding dimension when dealing with a time series.

Section 4 consists of Chapters 10 and 11. Neural networks are commonly used for prediction and classification when data sets are large. They have a major advantage over conventional statistical tools in that it is not necessary to assume any mathematical form for the functional relationship between the variables. However, they also have a few associated problems, like the risk of over-parametrization in the absence of P-values, the lack of appropriate diagnostic tools and the difficulties associated with model interpretation. These problems are particularly pertinent in the case of small data sets.

Chapter 10 investigates these problems from a statistical perspective in the context of typical market research data. Chapter 11 proposes an efficient on-line learning method called adaptive natural gradient learning. It can solve the plateau problems and can successfully be applied to learning involving large data sets.

The last section presents four application chapters, Chapters 12 to 15. Chapter 12 introduces rough clustering, a technique based on a simple extension of rough set theory to cluster analysis, and the applicability where group membership is unknown. Rough clustering solutions allow the multiple cluster membership of objects. The technique is demonstrated through the analysis of a data set containing scores associated with psychographic variables, obtained from a survey of shopping orientation and Web purchase intentions. Chapter 13 presents a survey of medical data mining focusing upon the use of heuristic techniques. The chapter proposes a forward-looking responsibility for mining practitioners that includes evaluating and justifying data mining methods–a task especially salient when heuristic methods are used. The chapter specifically considers the characteristics of medical data, reviewing a range of mining applications and approaches. In Chapter 14, machine learning techniques are used to predict the behavior of credit card users. The performance of these techniques is compared by both analyzing their correct classification rates and the knowledge extracted in a linguistic representation (rule sets or decision trees). The use of a linguistic representation for expressing knowledge acquired by learning systems aims to improve the user understanding. Under this assumption and to make sure that these systems will be accepted, several techniques have been developed by the artificial intelligence community, under both the symbolic and the connectionist approaches. The goal of Chapter 15 is to integrate evolutionary learning tools into the knowledge discovery process and to apply them to the large-scale, archaeological spatial-temporal data produced by the surveys. This heuristic approach presented in the chapter employs rough set concepts in order to represent the domain knowledge and the hypotheses.

This book will be useful to policy makers, business professionals, academics and students. We expect that the promising opportunities illustrated by the case studies and the tools and techniques described in the book will help to expand the horizons of KDD and disseminate knowledge to both the research and the practice communities.

We would like to acknowledge the help of all involved in the collation and the review process of the book, without whose support the project could not have been satisfactorily completed. Most of the authors of chapters included in this volume also served as referees for articles written by other authors. Thanks also to several other referees who have kindly refereed chapters accepted for this volume. Thanks go to all those who provided constructive and comprehensive reviews and comments. A further special note of thanks goes to all the staff at Idea Group Publishing, whose contributions throughout the whole process from inception to final publication have been invaluable.

In closing, we wish to thank all the authors for their insight and excellent contributions to this book. In addition, this book would not have been possible without the ongoing professional support from Senior Editor Dr. Mehdi Khosrowpour, Managing Editor Ms. Jan Travers and Development Editor Ms. Michele Rossi at Idea Group Publishing. Finally, we want to thank our families for their love and support throughout this project.

Ruhul Sarker, Hussein Abbass and Charles Newton
Editors

SECTION ONE

INTRODUCTION

Chapter I

Introducing Data Mining and Knowledge Discovery

R. Sarker, H. Abbass and C. Newton
University of New South Wales, Australia

The terms Data Mining (DM) and Knowledge Discovery in Databases (KDD) have been used interchangeably in practice. Strictly speaking, KDD is the umbrella of the mining process and DM is only a step in KDD. We will follow this distinction in this chapter and present a simple introduction to the Knowledge Discovery in Databases process from an optimization perspective.

INTRODUCTION

Our present information age society thrives and evolves on knowledge. Knowledge is derived from information gleaned from a wide variety of reservoirs of data (databases). Not only does the data itself directly contribute to information and knowledge, but also the trends, patterns and regularities existing in the data files. So it is important to be able to, in an efficient manner, extract useful information from the data and the associated properties of the data, i.e., patterns and similarities.

A new area of research, data extraction or data mining, has evolved to enable the identification of useful information existing in the data reservoirs. To understand and recognize the major initiatives in this research area, we will briefly describe the terminology and approaches to data mining and knowledge discovery in databases.

Knowledge discovery in databases (KDD) is the process of extracting models and patterns from large databases. The term *data mining* (DM) is often used as a synonym for the KDD process although strictly speaking it is just a step within KDD. DM refers to the process of applying the discovery algorithm to the data. We define the KDD process as:

KDD is the process of model *abstraction* from large databases and *searching* for valid, novel, and nontrivial *patterns* and *symptoms* within the abstracted model.

There are four keywords in the definition: abstraction, search, patterns, and symptoms. The database is a conventional element in KDD.

Abstraction: Abstraction is the process of mapping a system language Λ_1 to approximately an equivalent language Λ_2. The mapping is strong when it maps the system while neither losing existing patterns (completeness) nor introducing new patterns (soundness). Formally, Giunchiglia and Walsh (1992) define an abstraction, written f: $\Sigma_1 \rightarrow \Sigma_2$, as a pair of formal systems (Σ_1, Σ_2) with languages Λ_1 and Λ_2, respectively, and an effective total function f_Λ: $\Lambda_1 \rightarrow \Lambda_2$.

Search: It is more convenient to visualize the discovery process in terms of searching. One can measure the complexity and in turn the feasibility, of the discovery process by studying the search space. Most KDD steps can be seen as search-space reduction techniques. For example, the main goal for creating a target data set, data cleaning, and data reduction and projection is to reduce the noise in the data and to select a representative sample which then reduces the search space. The mining algorithm is the search technique used to achieve the overall process's objective.

Patterns: A pattern is an expression, η, in a language, Λ, describing a subset of facts, $\varphi_\eta \subseteq \varphi$, from all the facts, φ, which exist in the database (Fayyad, Piatetsky-Shapiro & Smyth, 1996c).

Symptoms: Although symptoms can be seen as patterns, the process of discovering the symptoms has more dimensions than finding simple descriptive patterns. Identification of symptoms is a major task for KDD if it is the intention to use it for decision support. The KDD process's role is to clear the noise within the system and discover abnormal signals (symptoms) that may contribute to potential problems.

In this definition, the term *process* implies that KDD consists of different steps such as: data preparation, search for patterns, knowledge evaluation and refinement. The discovered patterns should be *valid* with some degree of certainty, and *novel* (at least to the system and preferably to the users). *Nontrivial* delineates that the discovered patterns should not be obvious in the domain knowledge. They should, however, represent a substantial discovery to the user; otherwise the cost of the KDD process will not be justified.

THE KDD STEPS

In the literature, there have been some variations of the different steps involved in the KDD process. Although these variations do not differ in their entirety, some of them are more descriptive than others. Before we present the proposed steps, it is worth listing the KDD steps suggested in the literature as well as the criteria for selecting KDD's applications. The steps for the KDD process have been varied in the literature. Fayyad, Piatetsky-Shapiro and Smyth (1996c) identify nine steps: application domain understanding, creating a target data set, data cleaning and processing, data reduction and projection, choosing the mining task, choosing the mining algorithm, mining the data, interpreting the mined patterns, and consolidating the discovered knowledge. John (1997) sums up these steps into six: data extraction, data cleaning, data engineering, algorithm engineering, running the mining algorithm, and analyzing the results.

As we saw in the previous discussion, there are different views of what should be the steps for KDD. We believe the validity of a step is a case-by-case decision. For example, if the database is robust, then data cleaning is not needed or at least it will not take a significant effort or time. Nevertheless, we believe that the KDD steps should be broken up into detail rather than being summarized to make each step as clear as possible to the data analyst—whether it will take time or effort in the application or not. For example, we will propose a data conversion step. Most of the applications reported in the literature (Bryant & Rowe, 1998; Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy, 1996a) deal with advanced databases' file systems with high reliability at the level of data accuracy (see the NASA database in Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy,1996 for example). In many real-life applications, the data are in plain output files using old programming environments such as COBOL. Data conversion becomes an issue that we will consider and emphasize because it requires significant time as well as effort during the discovery process.

We propose to have 13 steps for KDD to be efficiently automated. The base on which each step is defined is either in terms of time span or technical knowledge and human effort. These steps are:

- problem definition and determining the mining task,
- data description and selection,
- data conversion,
- data cleaning,
- data transformation,
- data reduction and projection,
- domain-specific data preprocessing,
- feature selection,
- choosing the mining algorithm,
- algorithm-specific data preprocessing,
- applying the mining algorithm,
- analyzing and refining the results, and
- knowledge consolidation.

In the rest of this section, we will highlight some issues concerning some of these steps.

Problem definition and evaluation is the basic step used to conclude whether the mining process should take place or not and, if it should, what are its targets and aims. In general, the aims may vary from prediction to classification, description, etc. We will depend on the comprehensive criteria defined in Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy (1996a) to evaluate the validity of the problem for KDD.

In data description and selection, suitable files are selected as well as suitable data fields. There may exist many files/tables in the database. However, not all are suitable for knowledge discovery. After an initial study, suitable and unsuitable data should be identified. The selection here may involve selection of fields that are potentially useful for multiple mining tasks.

In data conversion, the database file system suitable for the mining process is identified and the data is converted from its original stored format to the selected one.

Data cleaning removes (or reduces) noise and errors in the data. If the proportion of inconsistent or noise data is quite small, their deletion may have little effect on the results. Inconsistency may also cause sufficient trouble for the mining algorithm and small SQL-like statements may track this type of inconsistency and facilitate the mining task.

With data transformation we mean reflecting the logical relations between the tables into a single table that contains all the information needed for the mining process. Many of the mining algorithms do not work on multiple-tables and therefore we need somehow to combine the tables into one. Redundancy may arise as a result and one needs to be careful here as this combination of tables may change the class frequency which will in turn affect the mining process.

We would like to differentiate between domain-specific data preprocessing and data reduction and projection. When projecting data, information is condensed into a fewer number of attributes; for example, calculating the financial ratios from an accounting database.

Domain-specific data preprocessing is a set of operations, using domain-specific knowledge, that makes the attribute valid from the domain viewpoint. For example, assume that one of the attributes is the market price over 10 years. In some cases, this attribute is only relevant if it is discounted to eliminate the effect of inflation. From our point of view, this task is neither reduction nor projection but data preprocessing imposed by the domain.

Furthermore, we prefer to divide the preprocessing into two types, domain-specific and algorithm-specific preprocessing. In the market price example, the algorithm does not impose the preprocessing. On the other hand, using feed-forward neural networks with sigmoid activation functions requires the normalization of at least the output. This is not the case when using decision trees, for example. Accordingly, algorithm-specific preprocessing should not alter the database and a view of the data should be made available to each algorithm with the preprocessing taking place on the view level. On the contrary, domain-specific preprocessing should be made available to all algorithms and not restricted to a specific one and accordingly, it should alter the transformed database. Actually, we suggest that domain-specific preprocessing should be accounted for within the data warehouse unlike algorithm-specific preprocessing, which works on a snapshot from the data warehouse. The reason for this is that the significance of an attribute is a function of the mining process and is valid over the whole set of the algorithms used for the same mining process, but the form of this attribute presented to each mining-algorithm is a function of the algorithm itself.

The DM step comprises the application of one or more computational techniques that proved acceptable in terms of their computational efficiency and ability to produce a particular enumeration of patterns (or models) over the data. It is recognized that the search space of models is huge and sometimes infinite. Practical computational constraints, therefore, place limits on the search space that can be explored by a data mining algorithm.

DATA MINING AND SEARCH

In data mining, there are three primary components: *model representation*, *model evaluation* and *search*. The two basic types of search methods used in data mining consist of two components: *Parameter Search* and *Model Search* (Fayyad, Piatetsky-Shapiro & Smyth 1996). In parameter search, the algorithm searches for the set of parameters for a fixed model representation, which optimizes the model evaluation criteria given the observed data. For relatively simple problems, the search is simple and the optimal parameter estimates can be obtained in a closed form. Typically, for more general models, a closed form solution is not available. In such cases, iterative methods, such as the gradient descent method of back-propagation for neural networks, are commonly used. The gradient descent method is one of the popular search techniques in conventional optimization (Hillier & Lieberman, 2001).

Model search occurs as a loop over the parameter search method: the model representation is changed so that a family of models is considered. For each specific model representation, the parameter search method is instantiated to evaluate the quality of that particular model. Implementations of model search methods tend to use heuristic search techniques since the size of the space of possible models often prohibits exhaustive search and closed form solutions are not easily obtainable. Here, heuristic search plays a key role in finding good solutions. A review of modern heuristic techniques is provided in Abbass (2002). Moreover, conventional optimization techniques are gaining popularity among data mining researchers and practitioners. Optimization theory provides the data mining community with a mature set of techniques that are ready to provide high quality solutions. However, research into the scalability of optimization techniques is still an open question.

DATA MINING FUNCTIONS

The general data mining tools, classified by data mining tasks, are presented below as provided by Abbass (2000):

Mining Task	Data Mining Tool	Chapters in Volume 1	Chapters in Volume 2
Feature Selection	Dependency models, Optimization	3, 5, 13	2, 8
Point Prediction/ Estimation	Regression methods, Neural networks, Regression decision trees, Support vector machines, Optimization	3, 13	7, 9
Classification	Statistical regression models, Neural networks, Classification decision trees, Support vector machines, Optimization	3, 7, 13	3, 5, 10, 11, 14
Rule discovery	Decision trees, Optimization, Learning classifier systems	3, 4, 6, 9, 10	
Clustering	Density estimation methods, Neural networks, Clustering techniques, Optimization	3, 2, 12, 13	12
Association methods	Association rules, Density estimation models, Optimization	13	

The basic functions of the data mining process include feature selection, summarization, association, clustering, prediction, and classification. These are summarized below.

Feature selection is concerned with the identification of a subset of features that significantly contributes to the discrimination or prediction problem. Bradley et al. (1998) formulated the feature selection problem as a mathematical program with a parametric objective function and linear constraints. Another approach uses a very fast iterative linear-programming-based algorithm for solving the problem that terminates in a finite number of steps (Mangasarian,1997).

Summarization involves methods for finding a compact description for a subset of data. Summarization can be performed using a bar chart or statistical analysis. This is useful for understanding the importance of certain attributes when compared against each other (Hui & Jha , 2000). More sophisticated methods involve the derivation of summary rules (Agrawal at al. 1996), multivariate visualization techniques, and the discovery of functional relationships between variables (Zembowicz & Zytkov, 1996).

Association rules determine how the various attributes are related. The association rules are also known as *Dependency Modeling,* which exists in two levels: the *structural* level of the model specifies (often in graphical form) which variables are locally dependent on which, whereas the *quantitative* level of the model specifies the strengths of the dependencies using some numerical scale (Fayyad, Piatetsky-Shapiro & Smyth 1996).

Clustering identifies a finite set of categories or clusters to describe the data (Jain & Dubes 1988). The categories may be mutually exclusive and exhaustive or consist of a richer representation such as hierarchical or overlapping categories (Fayyad, Piatetsky-Shapiro & Smyth 1996). Unlike classification, the number of desired groups is unknown. As a result, the clustering problem is usually treated as a two-stage optimization problem. In the first stage, the number of clusters is determined followed by the next stage, where the data is fitted to the best possible cluster. However, one needs to be careful here, as this type of sequential optimization techniques does not guarantee the optimality of the overall problem.

The use of *regression* modeling for *point estimation* is basically an unconstrained optimization problem that minimizes an error function. *Artificial neural networks* are widely used for prediction, estimation and classification (Balkin & Ord, 2000; Shin & Han, 2000; Zhang, Patuwo & Hu, 2001). In terms of model evaluation, the standard squared error and cross entropy loss functions for training artificial neural networks can be viewed as log-likelihood functions for regression and classification, respectively (Geman, Bienenstock & Doursat,1992; Ripley, 1994). Regression trees and rules are also used for predictive modeling, although they can be applied for descriptive modeling as well.

In *classification*, the basic goal is to predict the most likely state of a categorical variable (the class) given the values of the other variables. This is fundamentally a density estimation problem (Scott, 1992). A number of studies have been undertaken in the literature for modeling classification as an optimization problem (Bradley, Fayyad & Mangasaria, 1999) including *discriminant* analysis for classification which uses an unconstrained optimization technique for error minimization (Ragsdale, 2001). Classification using decision trees and boosting algorithms is presented in Chapter 3 of this volume.

Rule discovery (RD) is one of the most important data mining tasks. The basic idea is to generate a set of symbolic rules that describe each class or category. Rules should usually be simple to understand and interpret. RD can be a natural outcome of the classification process as a path in a decision tree from the root node to a leaf node represents a rule. However, redundancy is often present in decision trees and the extracted rules are always simpler than the tree. It is also possible to generate the rules directly without building a decision tree as an intermediate step. In this case, *learning classifier systems* (LCS) play a key method to rule discovery.

DATA MINING TECHNIQUES

In this section, we will present some of the most commonly used techniques for data mining. This list should not be considered complete, but rather a sample of the techniques for data mining.

Bayesian methods (BM) represent a powerful class of techniques to data mining. Actually, BM are the only set of techniques that can be proven in a strict mathematical sense to work under uncertainty. The ability of these techniques to capture the underlying relationship between the attributes and their use of probabilities as their method of prediction increase their reliability and robustness in data mining applications. However, BMs are slow and more research is needed to overcome the scalability of these techniques.

Feed-forward Artificial Neural Networks (FFANNs) are one of the most commonly used *artificial neural networks* (ANNs) architectures for data mining. FFANNs are in essence non-parametric regression methods, which approximate the underlying functionality in data by minimizing a loss function. Artificial neural networks are not so common among a part of the community as they are known to be slow. However, recent advances in artificial neural networks present fast training algorithms as well as adoptive networks (Yao, 1999). As a rule of thumb, ANNs are more accurate than many data mining techniques and the choice decision of the appropriate data mining tool is usually a cost-benefit analysis when it comes to real-life applications. If 1% increase in the accuracy means thousands of dollars or a human life, one can deposit more time and effort in the mining process to gain this 1%.

Decision trees (DTs) are either univariate or multivariate (Abbass, Towsey & Finn, 2001). *Univariate decision trees* (UDTs) approximate the underlying distribution by partitioning the feature space recursively with axis-parallel hyperplanes. The underlying function, or relationship between inputs and outputs, is approximated by a synthesis of the hyper-rectangles generated from the partitions. *Multivariate decision trees* (MDTs) have more complicated partitioning methodologies and are computationally more expensive than UDTs. The split at a node in an MDT depends on finding a combination of attributes that optimally (or satisfactorily) partitions the input space. The simplest combination of attributes is taken to be linear. Even in the simple case, the process is very expensive since finding a single linear hyperplane that optimally splits the data at a node is an NP-hard problem. A path from the root node to a leaf node in both UDTs and MDTs represent the rule for defining the class boundary of the class present at the leaf node.

Support Vector Machines (SVMs) (Burges, 1998; Cherkassky & Mulier, 1998; Vapnik, 2000) are powerful tools for both classification and point estimation. They classify points by assigning them to one of two disjoint halfspaces. These halfspaces are either in the original input space of the problem for linear classifiers or in a higher dimensional feature space for nonlinear classifiers. SVMs represent a good example of data mining techniques that are based on optimization theory.

Optimization methods (Bradley, Fayyad & Mangasarian, 1999) provide another alternative set of techniques that produce robust results. A major problem with these techniques is scalability and slow convergence. Global optimization can be combined with heuristics to overcome the slow performance of optimization techniques (Bagirov, Rubinov & Yearwood, 2001). However, conventional heuristics are much faster by many orders of magnitude than conventional optimization. This encourages more research into the scalability of optimization techniques for data mining.

PRACTICAL ISSUES

The KDD process is usually very expensive and spans over a long period of time. Therefore, one needs to be careful in all the steps involved as, for example, a blind application of a data mining technique may lead to the discovery of meaningless patterns. A main objective of KDD is to simplify the underlying model in the data for the use and understandability of the decision maker and those who own the data, regardless of whether or not they have the prerequisite knowledge of the techniques being used. The expected output from these steps is 'knowledge for use' – not a research output. In many cases, the steps need to be performed iteratively to ensure the quality of the KDD process.

With respect to the issue of the basis for selecting KDD applications, Fayyad, Piatetsky-Shapiro and Smyth (1996c) define four practical criteria and five technical criteria for selecting KDD applications. The former are the potential for significant impact of the application, no good alternatives exist, organization support and potential for privacy and legal issues. The latter are the availability of sufficient data, relevance of attributes, low noise levels in the data, confidence intervals, and prior knowledge.

We have briefly described the process to extract and elucidate the knowledge locked up in large databases. As previously mentioned a major effort has been experienced on data mining techniques so that knowledge can be extracted most efficiently. Many approaches are being developed to assist in this process. This book highlights many of the more successful approaches to facilitate the data mining phase of knowledge discovery. In many real-life applications, most of the time deposited for KDD is not spent in the mining step, but in the manipulation cleaning of the database. Research publications, however, focus on the formal stages (the feature selection and mining steps). We have tried in this book to avoid this bias and give place to real-life applications that use ad hoc approaches or experience to "do" KDD.

CONCLUSIONS

In this chapter, we have provided a brief introduction to data mining and knowledge discovery from databases. From the discussion in this chapter, it is clear that most KDD techniques are based on heuristic and optimization techniques. This book is a collection of examples of the use of optimization methods for data mining.

REFERENCES

Abbass, H. A. (2002). From Evolution to Swarm to Immune to …? A Simple Introduction to Heuristics, In *Data Mining: A Heuristic Approach*, H.A. Abbass, R. Sarker and C. Newton (Eds), Idea Group Publishing, USA, Chapter 1.

Abbass, H. A. (2000). Introducing Data Mining, In *Short Lectures in Data Mining and Hueristics*, H. A. Abbass (Ed.), UNSW, ADFA, Canberra, 1-9.

Abbass, H.A., Towsey M., & Finn G. (2001). C-Net: A Method for Generating Non-deterministic and Dynamic Multivariate Decision Trees. *Knowledge and Information Systems: An International Journal*, Springer-Verlag, 5(2).

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. & Verkamo, A. I. (1996). Fast Discovery of Association Rules, In U. Fayyad, G. Piatetsky-Shaprio, P. Smyth and R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, MIT Press, Cambridge, MA., 307-328.

Bagirov, A. M., Rubinov, A. M. & Yearwood, J. (2001). A heuristic algorithm for feature selection based on optimization techniques, Chapter 2, *Heuristic and Optimization for Knowledge Discovery* (this volume), Idea Group Publishing, USA.

Balkin, S. & Ord, J. (2000). Automatic Neural Network Modeling for Univariate Times Series, *International Journal of Forecasting, 16*, 509-515.

Bradley, P. S. & Mangasarian, O. L. (1998). Feature Selection via Concave Minimization and Support Vector Machines, *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98)*, J. Shavlik, editor, Morgan Kaufmann, San Francisco, California, 82-90.

Bradley, P. S., Fayyad, U. M. & Mangasarian, O. L. (1999). Mathematical Programming for Data Mining: Formulations and Challenges, *INFORMS Journal on Computing 11*, 217-238.

Bradley, P. S., Mangasarian, O. L. & Street, W. N. (1998). Feature Selection via Mathematical Programming, *INFORMS Journal on Computing 10*, 209-217.

Bradley, P. S., Mangasarian, O. L. & Street, W. N. (1997). Clustering via Concave Minimization, *Advances in Neural Information Processing Systems 9*, MIT Press, Cambridge, MA 368-374

Bryant, C. & Rowe, R. (1998). Knowledge Discovery in Databases: Application to Chromatography, *Trends in Analytical Chemistry, 17(1)*, 18-24.

C. Burges (1998). A Tutorial on Support Vector Machines for Pattern Recognition, *Data Mining and Knowledge Discovery, 2(2)*, 121-167.

Cherkassky, V. & Mulier, F. (1998). *Learning from Data – Concepts, Theory and Methods*, John Wiley & Sons, New York.

CPLEX Optimization Inc., Incline Village, Nevada. *Using the CPLEX(TM) Linear Optimizer and CPLEX(TM) Mixed Integer Optimizer* (Version 2.0), 1992.

Fayyad, U. & Stolorz, P. (1997). Data Mining and KDD: Promise and Challenges, *Future Generation Computer Systems, 13*, 99-115.

Fayyad, U., Piatetsky-Shaprio, G., Smyth, P. & Uthurusamy, R. (1996). (eds.), *Advances in Knowledge Discovery and Data Mining*, MIT Press, Cambridge, MA.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996a). From Data Mining to Knowledge Discovery, Chapter 1, In U. Fayyad, G. Piatetsky-Shaprio, P. Smyth and R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, MIT Press, Cambridge, MA., 1-34.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996b). The KDD Process for Extracting Useful Knowledge from Volumes of Data, *Communications of the ACM, 39*, 27-34.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996c). "Knowledge discovery and data mining: Towards a unifying framework". In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining*, pp. 1-36. AAI/MIT press.

Frawley, W., Piatetsky-Shapiro, G. and Matheus, C. (1991). "Knowledge discovery in databases: an overview". In G. Piatetsky-Shapiro and B. Frawley (Eds.), *Knowledge Discovery in Databases*. Cambridge, Mass: AAAI/MIT press.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural Networks and Bias/Variance Dilemma, *Neural Computation, 4*, 1-58.

Giunchiglia, F. & Walsh, T. (1992). "A theory of abstraction". *Artificial Intelligence*, 56(2), 323-390.

Hertz, J., Krogh, A. & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, California.

Hillier, F. & Lieberman, G. (2001). *Introduction to Operations Research*, McGrawHill, Boston.

Hui, S. & Jha, G. (2000). Data Mining for Customer Service Support, *Information & Management, 38*, 1-13.

Jain, A. K. & Dubes, R. C. (1988). *Algorithms for Clustering Data*, Englewood Cliffs, NJ: Prentice-Hall.

John, G. (1997). "Enhancements to the data mining process". Ph.D. thesis, Stanford University.

Mangasarian, O. L. (1997). Mathematical Programming in Data Mining, *Data Mining and Knowledge Discovery, 1(2)*, 183-201.

McLachlan, G. (1992). *Discriminate Analysis and Statistical Pattern Recognition*, New York: Wiley.

Ragsdale, C. T. (2001). *Spreadsheet Modeling and Decision Analysis*, South-Western College Publishing, USA.

Ripley, B. D. (1994). Neural Networks and Related Methods for Classification, *Journal of the Royal Statistics Society, 56(3)*, 409-437.

Scott, D. W. (1992). *Multivariate Density Estimation*, John Wiley and Sons, New York.

Shin, T. & Han, I. (2000). Optimal Single Multi-Resolution by Genetic Algorithms to Support Artificial Neural Networks for Exchange-Rate Forecasting, *Expert Systems with Applications, 18*, 257-269.

Vapnik, V. (2000). The *Nature of Statistical Learning Theory*, Springer, New York, Second Edition.

Weiss, S. I. & Kulikowski, C. (1991). *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Networks, Machine Learning and Expert Systems*, San Francisco, Calif.: Morgan Kaufmann.

Yao X. (1999). Evolving artificial neural networks, *Proceedings of the IEEE*, 87(9), 1423-1447.

Zembowicz, R. & Zytkov, J. M. (1996). From Contingency Tables to Various Forms of Knowledge in Databases, In U. Fayyad, G. Piatetsky-Shaprio, P. Smyth and R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, MIT Press, Cambridge, MA, 329-349.

Zhang, G., Patuwo, B. & Hu, M. (2001). A Simulation Study of Artificial Neural Networks for Nonlinear Time-Series Forecasting, *Computers and Operations Research, 28*, 381-396.

SECTION TWO

SEARCH AND OPTIMIZATION

Chapter II

A Heuristic Algorithm for Feature Selection Based on Optimization Techniques

A.M. Bagirov, A.M. Rubinov and J. Yearwood
University of Ballarat, Australia

The feature selection problem involves the selection of a subset of features that will be sufficient for the determination of structures or clusters in a given dataset and in making predictions. This chapter presents an algorithm for feature selection, which is based on the methods of optimization. To verify the effectiveness of the proposed algorithm we applied it to a number of publicly available real-world databases. The results of numerical experiments are presented and discussed. These results demonstrate that the algorithm performs well on the datasets considered.

INTRODUCTION

Feature extraction and selection is an important stage in the solution of pattern recognition problems. The feature selection problem involves the selection of a subset of features that will be sufficient in making predictions. There are various reasons for refining a feature set leading to the selection of suitable feature variables. Schaafsma (1982) gives a comprehensive description of reviews for feature selection. Among these are: the use of a minimal number of features to construct a simpler model, to give a simplest interpretation of such a model and to accelerate the decision making process. In some datasets the number of features and observations can reach several thousand and in such a situation the solution of classification problems without feature selection becomes fairly hard.

There exist various approaches to the solution of feature selection problems. We can note statistical (John, Kohavi & Pfleger, 1994; Kira & Rendell, 1992; Kittler, 1986; Koller & Sahami, 1996; McLachlan, 1992; Siedlecki & Sklansky, 1988), decision tree (Quinlan, 1993), neural network (Hecht, 1990; Hagan et al., 1996) and mathematical programming (Bradley, Mangasarian & Street, 1998; Bradley & Mangasarian, 1998; Chang, 1973; Fu, 1968; Hand, 1981) approaches among them. The book by Liu & Motoda (1998) gives a comprehensive description of statistical, decision tree and neural network approaches. In the papers by Bradley et al. (1998), and Bradley and Mangasarian (1998) the feature selection is formulated as a mathematical programming problem with a parametric objective function. Feature selection is achieved by generating a separating plane in the feature space. In the paper by Kudo and Sklansky (2000) a comparison of various feature selection algorithms is presented.

In this chapter, we suggest an algorithm for the solution of the feature selection problem based on techniques of convex programming. We consider feature selection in the context of the classification problem. In the above references, as a rule, datasets with two classes are considered. For statistical methods the selection problem with more than two classes is much more difficult than the problem with two classes where there is a close tie with multiple regression (McLachlan, 1992). The algorithm suggested in this paper allows one to consider datasets with an arbitrary number of classes.

The algorithm calculates a subset of most informative features and a smallest subset of features. The first subset provides the best description of a dataset whereas the second one provides the description which is very close to the best one. A subset of informative features is defined by using certain thresholds. The values of these thresholds depend on the objective of the task. Numerical experiments with several real-world databases have been carried out. We present their results and discuss them.

FEATURE SELECTION

The purpose of a feature selection procedure is to find as small a set as possible of informative features of the object under consideration, which describes this object from a certain point of view. The following issues are very important for understanding the problem.

1. It is convenient to consider (and define) informative features in the framework of classification. In other words it is possible to understand whether a certain feature is informative for a given example if we compare this example with another one from a different class. The following example confirms this observation.

Assume we consider a group A of people, who suffer from heart disease and recover in a hospital in a city E_1. We consider two features a_1 and a_2 of patients from this group. The feature a_1 describes a certain patient characteristic peculiar to this disease (for example, blood pressure). The feature a_2 describes how far from the hospital a patient resides. Consider a group B of patients from the same hospital in city E_1, who suffer from cancer. It is probable that a_1 is an informative feature for the group A in comparison with the group B and a_2 is not an informative feature.

Consider now a group C of patients from a hospital in a different city E_2, who suffer from heart disease. Then probably a_1 is not an informative feature and a_2 is an informative feature for A in comparison with C. Now assume that we consider a group D of patients from the hospital in E_2 who suffer from cancer. Then both a_1 and a_2 are informative features for the comparison of groups A and D.

2. Our goal is to find a sufficiently small set of informative features and to remove as many superfluous features as possible. We shall formalize this problem in the framework of a model later on. Note that this problem can have many different solutions. Assume for example, that it has been found that three features a_1, a_2 and a_3 of a certain object are informative. It is possible that there exists a certain correlation between these features or even a functional dependence between them (but this fact is unknown for the researcher). Assume for the sake of simplicity that there is a linear dependence $c_1a_1+c_2a_2+c_3a_3=k$. Then the feature a_3 can be expressed through a_1 and a_2 so it can be considered as superfluous. Thus we can consider (a_1, a_2) as a sufficient set of informative features. In the same manner we can consider also (a_2, a_3) and (a_1, a_3) as sufficient sets of informative features.
3. It follows from the above that the set of informative features, which describe a given object, is a categorical attribute of this object. This is also a fuzzy attribute in a certain informal sense. It leads to the following heuristic conclusion: it is useless to apply very accurate methods in order to find this set. However if we use heuristic not very accurate methods we need to have experimental confirmation of the results obtained.

In order to confirm that the results obtained are correct, we will use the following strategy: we consider a particular method for the search of a subset of informative features. Then we apply an auxiliary method based on a completely different idea to confirm that the discovered set of features describes the given object. If the auxiliary method confirms that the obtained set of features gives a correct description of the object, then the result of the main algorithm can be accepted. Otherwise, further investigation is required.

We now describe the approach that will be used in this chapter for feature selection (in the framework of classification). Assume we have a finite set A of vectors in n-dimensional space R^n and we wish to give a compact description of this set in terms of informative variables. This description can be given by means of a single point, which is called the centre of the set A. To define the centre, consider the deviation $d(x,A)$ of a point $x \in R^n$ from the set A. By definition $d(x,A)$ is the sum of distances from x to points $a^i \in A$:

$$d(x, A) = \sum_{a^i \in A} \| x - a^i \| \tag{1}$$

Here $\|\bullet\|$ is a norm in R^n . In the sequel we will consider $\|\bullet\|=\|\bullet\|_p$ where

$$\| x - a^j \| = \| x - a^i \|_p = (\sum_{l=1}^{n} | x_l - a_l^i |^p)^{1/p}.$$

Definition 1. The point x^0 is called the centre of the set *A* (with respect to a norm $\|\bullet\|$) if x^0 minimizes the deviation from A, that is, x^0 is a solution of the problem

$$\sum_{a^i \in A} \| x - a^i \| \to \min \quad \text{subject to} \quad x \in R^n \tag{2}$$

As a rule, one point cannot give a sufficiently precise description of the entire set *A*. To obtain this description, we need to have more points, which can be considered as centres of clusters of the set *A*. However we accept the hypothesis that for the search for such a categorical and fuzzy attribute as a set of informative features it is enough to consider only one representative of a set, namely its centre. It is assumed that this representative possesses some properties of the set, which allow one to replace the set itself by its centre in the study of the problem under consideration. The results of numerical experiments, which were confirmed by another (different) approach, demonstrate that this hypothesis leads to good results for many databases.

Consider now two subsets of the *n*-dimensional space R^n. To underline the dimension we include *n* in the notation. Thus we denote these sets by $A^1(n)$ and $A^2(n)$. Let $x^i(n)$ be the centre of the set $A^i(n), i= 1,2$. The quality of the description of the sets $A^1(n)$ and $A^2(n)$ by their centres can be expressed by numbers $N_1(n)$ and $N_2(n)$, where $N_1(n)$ is the number of the points from $A^1(n)$ which are closer to centre $x^2(n)$ of the other set $A^2(n)$ than to the centre $x^1(n)$ of $A^1(n)$. $N_2(n)$ has a similar meaning. We shall consider a four-tuple $(A^1(n),A^2(n), x^1(n), x^2(n))$. The number of "bad'' points $N(n) = N_1(n)+N_2(n)$ can be considered as a certain numerical characteristic of this four-tuple. (A point belonging to one of the sets is "bad'' if this point is closer to the centre of the other set.)

We wish to find the most informative features which allow us to distinguish sets $A^1(n)$ and $A^2(n)$ and remove the least informative features. Since $x^1(n)$ and $x^2(n)$ are representatives of the sets $A^1(n)$ and $A^2(n)$, respectively, which can replace the corresponding sets under consideration, we can assume that the closest coordinate of centres indicates the least informative feature. Then we can try to remove this coordinate. If we remove it, we will have new sets $A^1(n\text{-}1)$ and $A^2(n\text{-}1)$ and we can find centres $x^1(n\text{-}1)$ and $x^2(n\text{-}1)$ of these sets. Consider a new four-tuple and calculate the numerical characteristic of this four-tuple $(A^1(n\text{-}1), A^2(n\text{-}1), x^1(n\text{-}1), x^2(n\text{-}1))$. If this characteristic $N(n\text{-}1) = N_1(n\text{-}1)+N_2(n\text{-}1)$ is close enough to $N(n)$, we can deduce that the eliminated coordinate has little influence on this characteristic. So this coordinate need not belong to a set of informative features. On the other hand if the difference between *N(n)* and *N(n-1)* is sufficiently large, then we can not remove even the closest coordinate of centres, so all *n* coordinates belong to a set of informative features.

To be more precise we need to define what "sufficiently large'' means. To achieve this we will introduce certain thresholds.

The procedure is based on an *inner* description of a set by means of its centre and on comparison of two sets by means of their centres. We can treat this approach as an inner approach to feature selection. The opposite to an inner approach is an outer approach, which is based on a separation of two sets. The main idea behind the outer approach is the following. Having two sets A_1 and A_2 we can find a discrimination function *c* such that $c(a^1)>0$ for $a^1\in A_1$ and $c(a^2)<0$ for $a^2\in A_2$. Then if a new point is presented we can calculate $c(a)$. If $c(a)>0(c(a)<0,$ respectively) then we bring *a* to $A_1(A_2$, respectively). Various versions of this approach have been studied in Abello, Pardalos & Resende (2001) and very sophisticated methods for the determination of a discrimination function *c* have been proposed. However in some instances we can use very simple discrimination functions. One of them is applied in this paper, when we consider an auxiliary algorithm for confirmation of our main feature selection algorithm.

FEATURE SELECTION AS A CONVEX PROGRAMMING PROBLEM

We consider a dataset which contains *m* classes, that is, we have *m* nonempty finite sets A_i, $i = 1,...,m$ in R^n consisting of r_i, $i = 1,...,m$ points, respectively. Using the idea presented in the previous section we suggest an algorithm for the solution of the feature selection problem. Below we will consider two cases for p=1 and p=2. Let

$$N_1=\{j:j=1,...,|A_1|\},\ N_i=\{j:j=|A_i|+1,...,|A_{i-1}|+|A_i|\},\ i=2,...,m$$

where $|A_i|$ denotes the cardinality of the set $A_i=1,...,m$. First we will consider the case when $m=2$. Let $\varepsilon>0$ be some tolerance and $T_i\in\{1,2,...\}$, i=1,2,3 be the thresholds.

Algorithm 1. Feature selection
Step 1. Initialization. Set $k=0$, $I_k=\{1,...,n\}$.
Step 2. Determination of centres of the sets A_i, $i=1,2$. (See Definition 1). Compute the centre of A_i by solving the following problems of convex programming:

$$\sum_{j\in N_i}\| x^i - a^j \|_p \to \min \quad \text{subject to } x^i \in R^n, p = 1,2. \tag{3}$$

Here

$$\| x^i - a^j \|_1 = \sum_{l\in I_k} | x_l^i - a_l^j |, \| x^i - a^j \|_2 = [\sum_{l\in I_k} (x_l^i - a_l^j)^2]^{1/2}.$$

Step 3. Find points of the set A_i, $i=1,2,$ which are closer to the centre of the other set. Let x^{i*}, $i=1,2$ be solutions to the problem (3). Compute the sets:

$$N_1^k = \{j \in N_1 : \| x^{2*} - a^j \|_p \leq \| x^{1*} - a^j \|_p\},\ N_2^k = \{j \in N_2 : \| x^{1*} - a^j \|_p \leq \| x^{2*} - a^j \|_p\}.$$

Set $N_3^k = N_1^k \cup N_2^k$. If $k=0$ then go to Step 5, otherwise go to Step 4.
Step 4. Calculate $L_i = max\{|N_i^t|:t=0,...,k\}$, $i=1,2,$ $L_3 = max\{|N_1^t|+N_2^t|:t=0,...,k\}$
If $\max\{L_i-T_i:i=1,2,3\}>0$ then I_{k-1} is a subset of most informative features and the algorithm terminates. Otherwise go to Step 5.
Step 5. To determine the closest coordinates. Calculate

$d_0=min\{|x^{1*}_l-x^{2*}_l|:l\in I_k\}$

and define the following set: $R_k=\{l\in I_k:|x^{1*}_l-x^{2*}_l|\leq d_0+\varepsilon\}$.
Step 6. Construct the set: $I_{k+1} = I_k \backslash R_k$. If $I_{k+1} = \varnothing$ then I_k is the subset of most informative features. If $|I_{k+1}|=1$ then I_{k+1} is the subset of most informative features. Then the algorithm terminates, otherwise set $k=k+1$ and go to Step 2.

Remark 1. An algorithm for the case when the number of classes $m>2$ can be obtained from Algorithm 1 by replacing Steps 3, 4 and 5 by the Steps 3', 4' and 5', respectively.
Step 3'. To find points of a set A_i, $i=1,...,m$, which are closer to the centres of other sets.
Let x^{i*}, $i=1,...,m$ be solutions to the problem (3). Compute the sets:

$$N_i^k = \{j \in N_i : \min\{\| x^{t*} - a^j \|_p : t = 1,...,m, t \neq i\} \leq \| x^{i*} - a^j \|_p\}, i = 1,...,m.$$

Set $N_{m+1}^k=\cup\{N_i^k:i=1,...,m\}$ If $k=0$ then go to Step 5', otherwise go to Step 4'.
Step 4'. Calculate

$$L_i = \max\{| N_i^t | : t = 0,...,k\}, i = 1,...,m,\ \ L_{m+1} = \max\{\sum_{i=1}^{m} | N_i^t | : t = 0,...,k\}.$$

If $\max\{L_i-T_i:i=1,...,m+1\}>0$ then I_{k-1} is a subset of most informative features and the algorithm terminates. Otherwise go to Step 5'.
Step 5'. To determine the closest coordinates. Calculate

$d_l=max\{|x^{i*}_l-x^{t*}_l|:i,t=1,...,m\}$, $d_0=min\{d_l:l\in I_k\}$.

and define the following set: $R_k=\{l\in I_k: d_l\leq d_0+\varepsilon\}$.

Remark 2. It should be noted that the subset of most informative features calculated by the Algorithm 1 depends on the vector of thresholds $T = (T_1, T_2, ..., T_{m+1})$. In numerical experiments we shall consider two cases:

1. $T_i = | A_i | /100, T_{m+1} = \sum_{i=1}^{m} T_i;$ 2. $T_i = 2 | A_i | /100, T_{m+1} = \sum_{i=1}^{m} T_i.$

Remark 3. In order to confirm the results obtained by Algorithm 1 we shall use cross-validation with the features calculated in the current stage.

We shall use some indicators in order to confirm the results obtained by Algorithm 1. One of them is the accuracy e_i for the set A_i, $i=1,...,m$ and total accuracy

e_{tot} for all sets. Assume that misclassified points, that is, points from a given set, which are closer to the centre of the other set, are known. Then the accuracy e_i for the set A_i is calculated as follows:

$$e_i = 100(|A_i|-m_i)/|A_i|, \qquad (4)$$

where m_i is a number of misclassified points for the set A_i, $i=1,...,m$. In the same manner we can define total accuracy for all sets:

$$e_{tot} = 100(|A|-M)|A|, \qquad (5)$$

where $A = \sum_{i=1}^{m} |A_i|, M = \sum_{i=1}^{m} m_i$.

As mentioned above, the centre of the set as a rule cannot give a sufficiently precise description of the entire set, so the accuracies e_i and e_{tot} as a rule are not very high. However, we use this indicator in order to recognize the importance of the removed coordinate at each iteration of the algorithm. That is, we again include the dimension n of the set under consideration in the notation. Assume that after some iterations we have n-dimensional vectors. Denote the corresponding sets by $A_i(n)$ and let $e_i(n)$ be the accuracy for the set $A_i(n)$. Assume that a particular coordinate was removed, then we have the set $A_i(n-1)$ with the accuracy $e_i(n-1)$. The approximate equality $e_i(n)=e_i(n-1)$ can be considered as confirmation that the removed coordinate does not have any influence on the structure of the set A_i. On the contrary, if the difference between $e_i(n)$ and $e_i(n-1)$ sharply increases then we can suppose that the structure of sets $A_i(n)$ and $A_i(n-1)$ is different, so the coordinate that was removed is indeed informative.

We can consider different indicators which confirm the outcome of Algorithm 1. One of them is based on a very simple and rough classification algorithm. This algorithm provides a classification with not very high accuracy, which is compatible with the accuracy of the description of a set by its centre. However this algorithm does not require very much computational time, so it can be used to verify the results obtained by the feature selection algorithm. We suppose that the dataset under consideration contains two classes A_i, $i=1,2$. We define the following quantity for any $x \in R^n$:

$$\omega(x) = \frac{\rho(x,A_1)}{1+\rho(x,A_2)} - \frac{\rho(x,A_2)}{1+\rho(x,A_1)}.$$

where $\rho(x,A_i) = \min\{\|x-y\| : y \in A_i\}$, $i=1,2$. It is easy to see that $\omega(x)=-\rho(x,A_2)<0$ for all $x \in A_1$ and $\omega(x)=\rho(x,A_2)>0$ for all $x \in A_2$. Then knowing training sets A_1 and A_2 we can suggest the following algorithm for classification. Let B be a set of new observations and $N=|B|$.

Algorithm 2. Classification algorithm
Step 1. Initialization. Set $k=1$
Step 2. Take $x^k \in B$ and calculate $\rho_k = \rho(x^k,A_i) = \min \|x^k-a\|, a \in A_i$, $i=1,2$.
Step 3. If $\frac{\rho_1}{1+\rho_2} < \frac{\rho_2}{1+\rho_1}$

then $x^k \in A_1$ otherwise $x^k \in A_2$. Set $k=k+1$. If $k \leq N$ then go to Step 2, otherwise stop.

We shall apply this algorithm with cross-validation. The accuracy of Algorithm 2 is not very high, however we again use it only for confirmation of results obtained by the main Algorithm 1. If the coordinate, which is removed by Algorithm 1, does not change the accuracy of the classification algorithm, we can suppose that the structure of sets A_i does not change, so this coordinate is not very informative. Otherwise this coordinate is informative and cannot be removed. The following observation is very important: *two different verification approaches, based on the accuracy for the set A_i and the total accuracy from the classification algorithm confirm that the algorithm works well in all examples under consideration.*

Note that the confirmation approaches described above can be very easily implemented; however, they can only indirectly verify the results. To obtain the real confirmation of results obtained by the algorithm, we need to consider a very difficult problem of clustering (Bagirov, Rubinov, Stranieri & Yearwood, 2001). If the clustering based only on informative features can give a good description of known sets under consideration, then we can be sure that the algorithm works well. This procedure was carried out for all examples described in the next section (Bagirov, Rubinov & Yearwood, 2001). *The results obtained confirm that the proposed feature selection algorithm works well.*

RESULTS OF NUMERICAL EXPERIMENTS

In this section we discuss the results of numerical experiments for Algorithm 1 with some known datasets. We use the following notation for the description of results from the numerical experiments:

list—the list of attributes calculated by the algorithm for feature selection;

e_i —accuracy for *i-th* set (in %); e_{tot}—total accuracy for all sets (in %);

n_f —the number of objective function evaluations for the calculation of the corresponding subset of attributes. The number $n_f(i)$ for the *i*-th stage is calculated by the following formula: $n_f(i)=n_f(i-1)+n_{fi}$, where n_{fi} is the number of the objective function evaluations during the *i*-th stage;

t—the computational time (in seconds). The computational time $t(i)$ for the *i*-th stage is calculated by the following formula: $t(i)=t(i-1)+t_i$ where t_i is the computational time during the *i*-th stage;

e_{cv}—accuracy reached by the Algorithm 1 using the corresponding set of attributes and cross-validation (in %); e_{cl} —accuracy reached by Algorithm 2 using the corresponding set of attributes and cross-validation (in %).

T denotes the vector of thresholds, *I*—the subset of most informative features. Following Remark 2 we shall consider two vectors of thresholds. We use the list of attributes as they are given in the description of the datasets. The accuracies e_i and e_{tot} are calculated by (4) and (5), respectively. In the same manner accuracies e_{cv} and e_{cl} are calculated.

We used the discrete gradient method (Bagirov, 1999) for solving the convex programming problem of (3). The first task in dealing with the data under consideration was to normalize the features. This was done by a nonsingular matrix *M* so that the mean values of all features were 1.

Numerical experiments have been carried out using a PC Pentium-S with a 150 MHz CPU and the code was written in Fortran-90. It should be noted that some datasets under consideration contain both continuous and categorical features. We consider categorical features as they are given in the datasets.

THE AUSTRALIAN CREDIT DATASET

The Australian credit dataset has been studied before by Quinlan (1993). It contains two classes with 14 attributes (6 of them are continuous and the remaining 8 are categorical) and 690 observations. The set A_1 consists of 383 instances and the set A_2 contains 307 instances. We used 10-fold cross validation. The results of the numerical experiments are presented in Table 1.

For the 1-norm we obtain the following results:
1. $T = (4,3,7)$: $I=\{3,5,7,8,9,10,13,14\}$; 2. $T=(8,6,14)$: $I=\{5,7,8,910\}$.

For the 2-norm results were as follows:
1.$T = (4,3,7)$: $I=\{3,5,6,7,8,9,10,13,14\}$; 2. $T= (8,6,14)$: I=$\{3,5,7,8,9,10,11,14\}$.

The results for the 1-norm presented in Table 1 show that we can take (3,5,7,8,9,10,13,14) as a subset of most informative features (Stage 3). This subset provides the best description of the dataset. Results in columns e_{cv} and e_{cl} confirm this. e_{cv} begins to decrease after Stage 3. As a smallest subset of most informative

Table 1: Results for the Australian credit dataset

Stages	list	e_1	e_2	e_{tot}	n_f	t	e_{cv}	e_{cl}
1-norm								
1	1-14	85.9	68.1	78.0	-	-	78.5	79.0
2	2,3,5,7-10,13,14	85.9	68.1	78.0	16736	53.88	78.5	80.6
3	3,5,7-10,13,14	86.2	68.1	78.1	26372	74.59	78.5	79.7
6	5,7-10	85.6	67.4	77.5	40053	99.52	77.8	75.9
7	7-10	83.8	67.4	76.5	42619	103.20	76.8	61.3
8	8-10	77.6	68.1	73.3	43769	104.69	73.4	16.2
2-norm								
1	1-14	87.5	74.9	81.9	-	-	81.9	81.5
2	1-11,13,14	87.5	74.9	81.9	10242	37.79	81.9	80.6
5	3,5-11,13,14	87.2	74.6	81.6	34783	120.02	81.5	80.4
7	3,5,7-11,14	86.4	74.6	81.2	46194	152.20	81.2	80.1
8	3,5,7-10,14	84.3	75.9	80.6	50846	163.74	80.7	79.3
9	3,7-10,14	83.8	76.6	80.6	54690	172.64	80.6	78.8
10	7-10,14	81.2	75.6	78.7	57908	179.50	79.3	68.4
11	7-10	80.4	72.0	76.7	60128	183.84	77.4	61.2

features we can take (5, 7, 8, 9, 10) (Stage 6). We can see that e_{cv} and e_{cl} sharply decrease after Stage 6 which confirms our choice.

The results for the 2-norm presented in Table 1 show that we can take (3,5,6,7,8,9,10,11,13,14) as a subset of most informative features (Stage 5). It provides a description of the dataset which slightly differs from the best one. e_{cv} and e_{cl} begin to decrease after Stage 5. This subset contains the subset which was obtained by using the 1-norm. As a smallest subset of most informative features we can take (3,7,8,9,10,14) (Stage 9). This subset again contains the corresponding subset obtained by using the 1-norm. We can again see that e_{cv} and e_{cl} sharply decrease after Stage 9.

THE WISCONSIN DIAGNOSTIC BREAST CANCER DATASET

This dataset was created by W.H. Wolberg, General Surgery Department, University of Wisconsin, Clinical Sciences Center, W.N. Street and O.L. Mangasarian, Computer Sciences Department, University of Wisconsin. It contains two classes, 30 continuous attributes and 569 observations. The sets A_1 and A_2 contain 357 and 212 instances, respectively. We used 10-fold cross-validation. Results of the numerical experiments are presented in Table 2.

For the 1-norm results were as follows:

1. $T = (4,2,6)$: $I=\{3,4,6,7,8\}$; 2. $T=(7,4,11)$: $I=\{4,7,8\}$.

We obtain the following results for the 2-norm:

1. $T = (4,2,6)$: $I=\{4,7,8\}$; 2. $T=(7,4,11)$: $I=\{4,7,8\}$.

From the results for the 1-norm presented in Table 2 we see that (1, 2, 3, 4, 5, 6, 7, 8) is a subset of most informative features (Stage 3). The results in columns e_{cv} and e_{cl} confirm this. As a smallest subset of most informative features we can take (4, 7, 8) (Stage 8). We can see that e_{cv} and e_{cl} essentially decrease after Stage 8. It follows from the results for the 2-norm presented in Table 2 that (1, 2, 3, 4, 6, 7, 8) is the subset of most informative features (Stage 4). This is a subset of the corresponding set obtained by using the 1-norm. We can see that e_{cv} and e_{cl} begin to decrease after Stage 4. We can take (4, 7, 8) (Stage 8) as a smallest subset of most informative features which coincides with the corresponding set obtained by using the 1-norm. e_{cv} and e_{cl} sharply decrease after Stage 8.

THE HEART DISEASE DATASET

The heart disease dataset comes from the Cleveland Clinic Foundation and was supplied by Robert Detrano, V.A. Medical Center, Long Beach, USA. It is part of the collection of databases at the University of California, Irvine collated by David Aha. The dataset contains two classes, 13 attributes (9 of them are continuous and 4 are categorical) and 297 observations. The sets A_1 and A_2 consist of 137 and 160 instances, respectively. Results of numerical experiments are given in Table 3.

Table 2: Results for the breast cancer dataset

Stages	list	e_1	e_2	e_{tot}	n_f	t	e_{cv}	e_{cl}
1-norm								
1	1-10	96.1	87.7	93.0	-	-	92.9	91.6
2	1-9	96.1	87.7	93.0	6836	13.51	92.9	92.7
3	1-8	96.1	87.7	93.0	12974	24.44	92.9	92.5
5	1,3,4,6-8	96.1	87.3	92.8	19875	35.26	92.5	89.6
7	4,6-8	94.1	87.7	91.7	23443	39.87	91.6	90.0
8	4,7,8	96.1	85.4	92.1	24793	41.41	92.1	90.4
9	7,8	94.1	86.3	91.2	25597	42.23	90.9	88.9
10	7	90.8	84.0	88.2	26005	42.67	88.8	80.2
2-norm								
1	1-10	95.2	87.3	92.3	-	-	92.3	91.4
2	1-9	95.2	87.3	92.3	7192	17.19	92.1	91.8
4	1-4,6-8	95.2	87.3	92.3	17426	38.66	92.5	91.8
7	4,6-8	94.1	87.7	91.7	26325	54.04	91.4	90.4
8	4,7,8	95.5	87.3	92.4	27912	56.41	92.5	90.7
9	7,8	93.3	86.8	90.9	28885	57.67	90.9	87.9
10	7	90.8	84.0	88.2	29408	58.33	88.8	80.2

Table 3: Results for the heart disease dataset with

Stages	list	e_1	e_2	e_{tot}	n_f	t	e_{cv}	e_{cl}
1-norm								
1	1-13	67.9	89.4	79.5	-	-	77.8	77.4
2	1,3,5,7-13	67.9	89.4	79.5	13981	11.91	77.8	76.4
5	3,7,9-13	67.9	90.0	79.8	35506	25.98	77.4	70.1
6	7,9-13	66.4	86.9	77.4	39635	28.23	76.0	59.4
7	7,9,10,12,13	66.4	89.4	78.8	42348	29.66	77.4	54.5
8	7,9,10,12	65.6	87.5	77.4	44274	30.59	75.4	48.6
9	7,10,12	64.2	85.6	75.8	45265	31.08	72.2	9.0
2-norm								
1	1-13	75.2	80.0	77.8	-	-	77.8	78.1
2	1-4,6-13	75.2	80.0	77.8	8651	10.16	77.8	77.1
4	2,3,6-13	75.2	80.0	77.8	24273	27.02	77.8	77.1
5	2,3,7-13	75.2	81.9	78.8	30629	33.23	77.4	77.4
8	2,7,9,10,12,13	75.2	81.9	78.8	44892	45.75	77.1	62.5
9	7,9,10,12,13	75.2	80.6	78.1	47920	48.11	77.8	54.2
10	7,9,10,12	74.5	79.4	77.1	50099	49.71	77.1	48.6
11	9,10,12	74.5	78.1	76.4	51898	50.97	76.0	43.1

We used 9-fold cross-validation.

Results for the 1-norm are as follows:
1. T= (2,3,5): I={3,7,9,10,11,12,13}; 2. T= (4,6,10): I= {7,9,10,12,13}.
Results for the 2-norm are as follows:
1. T= (2,3,5): I={2,7,9,10,12,13}; 2. T= (4,6,10): I= {7,9,10,12,13}.

The results for the 1-norm presented in Table 3 show that (3,7,9,10,11,12,13) is a subset of most informative features (Stage 5) and (7,9,10,12,13) is a smallest subset of most informative features (Stage 7). We can see that e_{cv} decreases and e_{cl} sharply decreases after Stages 5 and 7, which confirms the obtained results. It follows from the results for the 2-norm presented in Table 3 that (2, 7, 9, 10, 12, 13) is a subset of most informative features (Stage 8) and (7, 9, 10, 12, 13) is a smallest subset of most informative features (Stage 9). The results presented in columns e_{cv} and e_{cl} confirm this. We can see the smallest subsets of most informative features are the same for the 1-norm and 2-norm.

CONCLUSION

In this chapter an algorithm for feature selection in datasets has been proposed and discussed. We formulated feature selection as a convex programming problem and achieved this task by determining centres to describe classes in a feature space. The closest coordinates of the centers are removed step by step. If removal of a feature leads to significant changes in the structure of the classes then the algorithm terminates and the previous subset of features is considered as the least a subset of most informative features. One could represent the classes by their mean values, but results of numerical experiments show that such an approach does not lead to good results. This algorithm allows one to consider datasets with an arbitrary number of classes.

Numerical experiments on publicly available real-world databases have been carried out. The results from these experiments show that the proposed algorithm quickly calculates subsets of informative features. They also show that the number of such subsets is not, in general, unique.

The algorithm allows the calculation of sets of most informative features as well as very small subsets of these features which consist of "almost all" informative features. These subsets can be used further for solving classification problems.

Comparison of the proposed algorithm with other feature selection algorithms and application of its results for solving classification problems are the subject of further investigation.

ACKNOWLEDGMENT

The authors are grateful to the anonymous referees for their useful comments which have improved the quality of this chapter. This research was supported by the Australian Research Council.

REFERENCES

Abello, J., Pardalos, P.M. & Resende, M.G.C. (eds.) (2001). *Handbook of Massive Datasets*, Kluwer Academic Publishers, Dordrecht.

Bagirov, A.M. (1999). Derivative-free methods for unconstrained nonsmooth optimization and its numerical analysis. *Investigacao Operacional, 19*, 75-93.

Bagirov, A.M., Rubinov, A.M., Stranieri, A. & Yearwood, J. (2001). A global optimization approach to classification in medical diagnosis and prognosis. In: *Proceedings of 34-th Hawaii International Conference on System Sciences*, Hawaii.

Bagirov, A.M., Rubinov, A.M. & Yearwood, J. (2001). A global optimization approach to classification. Research Report 01/4 University of Ballarat, Australia.

Bradley, P.S., Mangasarian, O.L. & Street, W.W. (1998). Feature selection via mathematical programming. *INFORMS Journal on Computing*, 10, 209-217.

Bradley, P.S. & Mangasarian, O.L. (1998). Feature selection via concave minimization and support vector machines, In: *Machine Learning Proceedings of the Fifteenth International Conference (ICML 98),* J. Shavlik (ed.), Morgan Kauffman, San Francisco, 82-90.

Chang, C.Y. (1973). Dynamic programming as applied to feature subset selection in a pattern recognition system. *IEEE Trans. Syst. Man. Cybern. SMC-3*, 166-171.

Cost, S. & Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning, 10*, 57-78.

Fu, K.-S. (1968). *Sequential Methods in Pattern Recognition and Machine Learning.* New York: Academic.

Hagan, M., Demuth, H. & Beale, M.(1996). *Neural Network Design.* PWS Publishing Company, Boston.

Hand, D.J. (1981). Branch and bound in statistical data analysis. *Statistician, 30*, 1-30.

Hecht, R. (1990). *Neurocomputing.* Addison-Wesley Publishing Company, Amsterdam.

John, G.H., Kohavi, R. & K. Pfleger, K. (1994). Irrelevant features and the subset selection problem. In: *Proceedings of the 11th International Conference on Machine Learning*, San Mateo, CA, Morgan Kauffman.

Kira, K. & Rendell, L. (1992). The feature selection problem: traditional methods and a new algorithm, In: *Proceedings of the Ninth National Conference on Artificial Intelligence*, AAAI Press/The MIT Press, 129-134.

Kittler, J. (1986). Feature selection and extraction, In: *Handbook of Pattern Recognition and Image Processing*, Young and Fu (eds.), Academic Press, New York.

Koller, D. & Sahami, M. (1996). Toward optimal feature selection. In: L. Saitta, editor. *Machine Learning - Proceedings of the Thirteenth International Conference (ICML 96)* - Bari, Italy, July 3-6, 1996, 284-292, San Francisco, Morgan Kauffman.

Kudo, M. & Sklansky, J. (2000). Comparison of algorithms that select features for pattern classifiers. *Pattern recognition, 33*, 25-41.

Liu, H. & Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publeshers, Boston.

McKay, R.J. & Campell, N.A. (1982). Variable selection techniques in discriminant analysis I. Description. *Br. J. Math. Statist. Psychol.,35*, 1-29.

McKay, R.J. & Campell, N.A. (1982). Variable selection techniques in discriminant analysis II. Allocation. *Br. J. Math. Statist. Psychol.,35*, 30-41.

McLachlan, G.J. (1992). *Discriminant Analysis and Statistical Pattern Recognition*, New

York: A Wiley-Interscience Publication, John Wiley & Sons, Inc.

Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning.* San Mateo: Morgan Kaufmann.

Ridout, M.S. (1981). Algorithm AS 233. An improved branch and bound algorithm for feature selection. *Appl. Statist., 37*, 139-147.

Schaafsma, W. (1982). Selecting variables in discriminant analysis for improving upon classical procedures. In: *Handbook of Statistics, 2*, P.R. Krishnaiah and L. Kanal (eds.). Amsterdam: North Holland, 857-881.

Siedlecki, W. & Sklansky, J. (1988). On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence, 2*,197-220.

Chapter III

Cost-Sensitive Classification Using Decision Trees, Boosting and MetaCost

Kai Ming Ting
Monash University, Australia

This chapter reports results obtained from a series of studies on cost-sensitive classification using decision trees, boosting algorithms, and MetaCost which is a recently proposed procedure that converts an error-based algorithm into a cost-sensitive algorithm. The studies give rise to new variants of algorithms designed for cost-sensitive classification, and provide insights into the strength and weaknesses of the algorithms. First, we describe a simple and effective heuristic of converting an error-based decision tree algorithm into a cost-sensitive one via instance weighting. The cost-sensitive version performs better than the error-based version that employs a minimum expected cost criterion during classification. Second, we report results from a study on four variants of cost-sensitive boosting algorithms. We find that boosting can be simplified for cost-sensitive classification. A new variant which excludes a factor used in ordinary boosting has an advantage of producing smaller trees and different trees for different scenarios; while it performs comparably to ordinary boosting in terms of cost. We find that the minimum expected cost criterion is the major contributor to the improvement of all cost-sensitive adaptations of ordinary boosting. Third, we reveal a limitation of MetaCost. We find that MetaCost retains only part of the performance of the internal classifier on which it relies. This occurs for both boosting and bagging as its internal classifier.

INTRODUCTION

Cost-sensitive classification allows one to assign different costs to different types of misclassifications. For example, in the field of natural disaster prediction, the cost of having a disaster undetected is much higher than the cost of having a false alarm. Thus, cost-sensitive classification seeks to minimize the total misclassification cost. In contrast, conventional classification seeks to minimize the total errors regardless of cost.

Cost-sensitive tree induction employing the greedy divide-and-conquer algorithm has attracted much interest recently. Breiman, Friedman, Olshen and Stone (1984) describe two different methods of incorporating variable misclassification costs into the process of tree induction. These methods adapt the test selection criterion in the tree growing process. Pazzani, Merz, Murphy, Ali, Hume and Brunk (1994) reported negative empirical results when using one of Breiman et al.'s formulations to induce cost-sensitive trees. They found that the cost-sensitive trees do not always have lower misclassification costs, when presented with unseen test data, than those trees induced without cost consideration. Using a post-processing approach, Webb (1996) shows that applying a cost-sensitive specialization technique to a minimum error tree can reduce its misclassification costs by a small margin. In contrast to Pazzani et al.'s study, Ting (in press) shows convincingly that, by applying a simple heuristic, a truly cost-sensitive tree can be effectively learned directly from the training data, employing the greedy divide-and-conquer algorithm. The paper extends this line of research into improving the performance by combining multiple trees.

Boosting has been shown to be an effective method of combining multiple models in order to enhance the predictive accuracy of a single model (Quinlan, 1996; Freund & Schapire, 1996; Bauer & Kohavi, 1999). Boosting is amenable to cost-sensitive adaptation and recent research has reported some success (Ting & Zheng, 1998; Fan, Stolfo, Zhang & Chan, 1999). However, the relative performance between the proposed methods has yet to be investigated, and other forms of adaptations are also possible.

In this paper, we study two new variants of cost-sensitive boosting, and two recently proposed variants by Fan et al (1999) and Ting and Zheng (1998). All these variants must relearn their models when misclassification cost changes. An alternative method that converts an error-based model to a cost-sensitive model simply applies a minimum expected cost criterion (Michie, Spiegelhalter & Taylor, 1994) to the error-based model, and the same model can be reused when cost changes. Therefore, it is important to investigate whether the cost-sensitive variants have any advantage over this simple alternative. This study aims at improving our understanding of the behavior of the four cost-sensitive boosting algorithms and how variations in the boosting procedure affect misclassification cost.

MetaCost (Domingos, 1999) is a recently proposed method for making an arbitrary classifier cost-sensitive. The method has an interesting design that uses a "meta-learning" procedure to relabel the classes of the training examples and then employs the modified training set to produce a final model. MetaCost has been shown to outperform two forms of stratification which change the frequency of classes in the training set in proportion to their cost.

MetaCost depends on an internal cost-sensitive classifier in order to relabel classes of training examples. But the study by Domingos (1999) made no comparison between MetaCost's final model and the internal cost-sensitive classifier on which MetaCost depends. This comparison is worth making as it is credible that the internal cost-sensitive classifier may outperform the final model without the additional computation required to derive the final model.

We first provide background information about cost-sensitive decision tree induction using instance weighting heuristic in the next section. Then, we describe the boosting procedure used in this chapter: AdaBoost, along with cost-sensitive adaptations and provide the details of the MetaCost procedure in the following two sections. The next section reports the experiments and discusses the results and findings. We conclude in the final section.

COST-SENSITIVE DECISION TREE INDUCTION VIA INSTANCE WEIGHTING

Let N be the total number of instances from the given training set, and N_i be the number of class i instances. Similarly, let $N(t)$ and $N_i(t)$ be the number of instances and class i instances in node t of a decision tree. The probability that an instance is in class i given that it falls into node t is given by the ratio of the total number of class i instances to the total number of instances in this node.

$$p(i \mid t) = \frac{N_i(t)}{\sum_j N_j(t)} \tag{1}$$

When node t contains instances that belong to a mixture of classes, the standard greedy divide-and-conquer procedure for inducing trees (e.g., Breiman et al.,1984; Quinlan, 1993) uses a test selection criterion to choose a test at this node such that the training instances which fall into each branch, as a result of the split, become more homogeneous. One of the commonly used criteria is entropy, that is, $-\Sigma_i p(i|t)$ $log[p(i|t)]$. At each node, the tree growing process selects a test which has the maximum gain in entropy until the node contains only a single-class collection of instances.

To avoid overfitting, a pruning process is employed to reduce the size of the tree such that the estimated error is a minimum. In short, the standard tree induction procedure seeks to produce a *minimum error* tree. For a detail account of the decision tree induction algorithm, please refer to Breiman et al. (1984) and Quinlan (1993).

Our intuition for cost-sensitive tree induction is to modify the weight of an instance proportional to the cost of misclassifying the class to which the instance belonged, leaving the sum of all training instance weights still equal to N. The last condition is important because there is no reason to alter the size of the training set, which is equivalent to the sum of all training instance weights, while the individual instance weights are adjusted to reflect the relative importance of instances for making future prediction with respect to cost-sensitive classification. The heuristic is formalized as follows.

Let $C(i)$ be the cost of misclassifying a class i instance; the weight of a class i instance can be computed as

$$w(i) = C(i) \frac{N}{\sum_j C(j) N_j}, \tag{2}$$

such that the sum of all instance weights is $\Sigma_i\, w(i)\, N_i = N$. For $C(i) \geq 1$, $w(i)$ has the smallest value $0 < [(N)/(\Sigma_j\, C(j)\, N_j)] \leq 1$ when $C(i)$=1; and the largest value $w(i) = [(C(i)\, \Sigma_j\, N_j)/(\Sigma_j\, C(j)\, N_j)] \geq 1$ when $C(i) = max_j\, C(j)$.

Similar to $p(i|t)$, $p_w(i|t)$ is defined as the ratio of the total weight of class i instances to the total weight in node t:

$$p_w(i \mid t) = \frac{W_i(t)}{\sum_j W_j(t)} = \frac{w(i) N_i(t)}{\sum_j w(j) N_j(t)} \tag{3}$$

The standard greedy divide-and-conquer procedure for inducing minimum error trees can then be used without modification, except that $W_i(t)$ is used instead of $N_i(t)$ in the computation of the test selection criterion in the tree growing process and the error estimation in the pruning process. Thus both processes are affected due to this change.

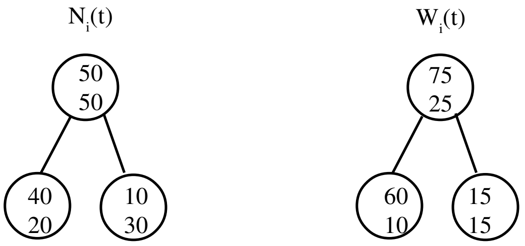
We modified C4.5 (Quinlan, 1993) to create C4.5CS. We only need to initialize the training instance weights to $w(i)$ since C4.5 has already employed $W_i(t)$ for the computation discussed above[1].

This modification effectively converts the standard tree induction procedure that seeks to minimize *the number of errors* regardless of cost to a procedure that seeks to minimize *the number of errors with high weight or cost*. Note that minimizing the latter does not guarantee that the total misclassification cost is minimized; this is because the number of low cost errors is usually increased as a result.

The advantage of this approach is that the whole process of tree growing and tree pruning is the same as that used to induce minimum error trees. This can be viewed as a generalization of the standard tree induction process where *only the initial instance weights determine the type of tree to be induced—minimum error trees or minimum high cost error trees.*

Figure 1 shows an example of a split on the same attribute test using unit instance weights (in the left diagram) and different instance weights (in the right diagram). The sum of the instance weights for each class is shown in each node. With

Figure 1: Splitting on the same test—using unit instance weights (left) and different instance weights (right)



unit weights, each sum is equivalent to the number of instances for each class, $N_i(t)$. This example has two equiprobable classes, where N_1=N_2=50 at the root of the tree. The right diagram shows the result of the same split when C(1)=3 and C(2)=1. Employing Equation (1), the weights of all instances are modified to w(1)=1.5 and w(2)=0.5. As a result, the sums of the class i instance weights at the root are W_1 = 75 and W_2 = 25. This example shows that initializing the instance weights to $w(i)$ amounts to changing the class distribution of the training data.

To classify a new instance, C4.5CS predicts the class that has the maximum weight at a leaf, as in C4.5.

Ting (in press) has demonstrated the instance weighting heuristic works well. Compared with C4.5, the cost-sensitive version (C4.5CS) performs better by an average relative reduction of 14% in term of misclassification cost and an average relative reduction of 34% in terms of tree size over twelve data sets. Compared with C5 (Quinlan, 1997), the improved version of C4.5 which can produce a cost-sensitive tree, C4.5CS performs comparably in terms of cost and significantly better in term of tree size with an average relative reduction of 16%. The summarized result reported by Ting (in press) is given in Table 1.

Here we describe how the cost of misclassification can be specified in a cost matrix, and how the cost matrix is related to $C(i)$ in Equation (2). In a classification task of J classes, the misclassification costs can be specified in a cost matrix of size J xJ. The rows of the matrix indicate the predicted class, and the column indicates the actual class. The off-diagonal entries contain the costs of misclassifications; and

Table 1: Mean ratios for C4.5CS versus C4.5 and C4.5CS versus C5 over twelve data sets (from Ting, in press)

	C4.5CS vs C4.5	C4.5CS vs C5
Misclassification Cost ratio	.86	.98
Tree Size ratio	.66	.84

on the diagonal lie the costs for correct classifications, which are zero in this case since our main concern here is total misclassification costs of an induced tree.[2]

Let *cost*(*i*,*j*) be the cost of misclassifying a class *i* instance as belonging to class *j*. In all cases, *cost*(*i*,*j*)=0.0 for *i* = *j*. A cost matrix must be converted to a cost vector *C*(*i*) in order to use Equation (2) for instance, weighting. In this paper, we employ the form of conversion suggested by Breiman et al. (1984):

$$C(i) = \sum_{j=1}^{J} cost(i, j) \tag{4}$$

In our experiments, without loss of generality, we impose a unity condition—at least one *cost*(*i*,*j*)=1.0 which is the minimum misclassification cost. The only reason to have this unity condition or normalization[3] is to allow us to measure the number of *high cost errors*, which is defined as the number of misclassification errors that have costs more than 1.0.

ADABOOST AND COST-SENSITIVE ADAPTATIONS

AdaBoost uses a base learner to induce multiple individual classifiers in sequential trials, and a weight is assigned to each training example. At the end of each trial, the vector of weights is adjusted to reflect the importance of each training example for the next induction trial. This adjustment effectively increases the weights of misclassified examples and decreases the weights of the correctly classified examples. These weights cause the learner to concentrate on different examples in each trial and so lead to different classifiers. Finally, the individual classifiers are combined to form a composite classifier. The AdaBoost procedure, shown in Figure 2, is a version of boosting that uses the confidence-rated predictions described in Schapire and Singer (1999).

In order to have a fair comparison with other cost-sensitive algorithms, AdaBoost can be made cost-sensitive by applying a minimum expected cost criterion during classification. This is made possible by assuming weighted votes for each class are proportional to class probability,

$$P(i \mid x) \propto \sum_{k:H_k(x)=y} \alpha_k H_k(x) \tag{9}$$

$$H^*(x) = \underset{j}{\arg\ \min} \sum_{i} \sum_{k:H_k(x)=i} \alpha_k H_k(x)\ cost(i, j) \tag{10}$$

where *cost*(*i*,*j*) is the cost of misclassifying a class *i* example as class *j*.

Four modifications of AdaBoost are assessed in this chapter. All these variants evaluate cost during the training process. The simplest is the only variant that does not use confidence-rated predictions in the weight-update rule of Step (iii). This

Figure 2: The AdaBoost procedure

Given a training set T containing N examples (x_n, y_n) where x_n is a vector of attribute values and $y_n \in Y$ is the class label, $w_k(n)$ denotes the weight of the nth example at the kth trial.

Initialization: $w_1(n)=1$.

In each trial $k = 1,...,K$, the following three steps are carried out.

(i) A model H_k is constructed using the training set under the weight distribution w_k. Let $H_k(x)$ denote the predicted class, and $H_k(x) \in [0,1]$ denote the confidence level of the prediction.

(ii) Classify T using H_k, and compute r_k as defined by

$$r_k = \frac{1}{N}\sum_n \delta\ w_k(n)\, H_k(x_n) \tag{5}$$

where $\delta = -1$ if $H_k(x_n) \neq y_n$ and $\delta = +1$ otherwise. Then, compute

$$\alpha_k = \frac{1}{2}\ln\left(\frac{1+r_k}{1-r_k}\right) \tag{6}$$

(iii) The weight vector $w_{(k+1)}$ for the next trial is updated as follows:

$$w_{k+1}(n) = w_k(n)\exp(-\delta\ H_k(x_n)\ \alpha_k) \tag{7}$$

If $w_{(k+1)}(n) < 10^{-8}$, set it to 10^{-8} to solve the numerical underflow problem (Bauer & Kohavi, 1999). Normalize $w_{(k+1)}(n)$ so that the sum of all $w_{(k+1)}(n)$ equals to N.

After K models are induced, they are ready for prediction. For each classification, the final prediction is the combined prediction from the K models using the maximum vote criterion, computed as follows.

$$H^*(x) = \underset{y \in Y}{\arg\max} \sum_{k:H_k(x)=y} \alpha_k\ H_k(x) \tag{8}$$

variant, denoted CSB0, can be obtained by modifying Equation (7) to

$$w_{k+1}(n)=C_\delta\, w_k(n), \tag{11}$$

where $C_- = cost(y_n, H_k(x_n))$ and $C_+ = 1$. Ting and Zheng (1998) introduced this variant.

The next two variants are created in this paper to study the effects of including cost and the parameter α in the weight-update rule. Both utilize cost in the same form but one does not use α in the formulation. The second variant CSB1 is obtained by modifying Equation (7) to

$$w_{k+1}(n)=C_\delta\, w_k(n)\, exp(-\delta\, H_k(x_n)). \tag{12}$$

Because α is not used, CSB1 requires only the computation classify T using H_k in Step (ii) to update weights.

The third variant CSB2 is obtained by changing Equation (7) to

$$w_{k+1}(n)=C_\delta\, w_k(n)\, exp(-\delta\, H_k(x_n)\alpha_k), \tag{13}$$

CSB2 reduces to AdaBoost if $cost(i,j)=1$, $\forall\ i \neq j$.

The last variant called AdaCost is due to Fan et al. (1999). This procedure incorporates a cost adjustment function β_δ in Equations (5) and (7). The first modified equation is

$$r_k = \frac{1}{N}\sum_n \delta\, w_k(n)\, H_k(x_n)\, \beta_\delta\,, \tag{14}$$

where $\beta_+ = -0.5c_n + 0.5$ and $\beta_- = 0.5c_n + 0.5$; and c_n is the cost of misclassifying the nth example.

The weight-update rule in AdaCost is

$$w_{k+1}(n)=C_\delta\, w_k(n)\, exp(-\delta\, H_k(x_n)\alpha_k\, \beta_\delta). \tag{15}$$

For two-class problems, c_n is equivalent to the cost of misclassifying class y_n example in a fixed cost scenario. This is described by a cost matrix $c_n = cost(y_n,j)$ for $y_n \neq j$. Because c_n must be normalized to [0,1] (Fan et al., 1999), each value is divided by the maximum misclassification cost.

In our implementation all four variants incorporate the minimum expected cost criterion defined in Equation (10) and also a weight initialization process defined in Equation (2).

To conduct the experiments described in the experiment section, C4.5 (Quinlan, 1993) underlies the model H_k in the boosting procedure. Only the default settings of C4.5 are used. The confidence level of a prediction is a Laplacian estimate of the class probability. The parameter K controlling the number of classifiers generated in the procedure is set at 10 for all experiments.

METACOST

MetaCost (Domingos, 1999) is based on the Bayes optimal prediction that minimizes the expected cost $R(j|x)$ (Michie, Spiegelhalter & Taylor, 1994):

$$R(j\mid x)=\sum_{i=1}^{I} P(i\mid x)\;\; cost(i,j).$$

To apply the Bayes optimal prediction rule, first compute the expected cost for each class and then choose the class that has the minimum expected cost to be the final prediction. Note that this is the same rule that is used to derive Equation (10).

The Bayes optimal prediction rule implies a partition of the example space into I regions, such that class i is the minimum expected cost prediction in region i. If misclassifying class i becomes more expensive relative to misclassifying others, then parts of the previously non-class i regions shall be reassigned as region i since it is now the minimum expected cost prediction.

The MetaCost procedure applies the above principle by first estimating class probabilities using bagging (Bauer & Kohavi, 1999), and then relabeling the training examples with their minimum expected cost classes, and finally relearning a model using the modified training set.

We interpret the process of estimating class probabilities and applying the Bayes optimal prediction rule as constructing an internal cost-sensitive classifier for MetaCost. With this interpretation, we formalize the MetaCost procedure as a three-step process, depicted in Figure 3.

The procedure begins to learn an internal cost-sensitive model by applying a cost-sensitive procedure which employs a base learning algorithm. Then, for each of the training examples, assign the predicted class of the internal cost-sensitive model to be the class of the training example. Finally, learn the final model by applying the same base learning algorithm to the modified training set.

Figure 3: The MetaCost procedure

```
Given $T$: a training set containing $N$ examples $(x_n,y_n)$ where $x_n$ is a vector of
attribute-values and $y_n$ is the class label, $L$: a base learning algorithm, $C$: a cost-
sensitive learning procedure, and $cost$: a cost matrix.

MetaCost($T$,$L$,$C$,$cost$)

    (i)   Learn an internal cost-sensitive model by applying $C$:
          $H^* = C(T,L, cost)$.

    (ii)  Modify the class of each example in $T$ from the prediction of $H^*$:
          $y_n = H^*(x_n)$.

    (iii) Construct a model $H$ by applying $L$ to $T$.

Output: $H$.
```

The performance of MetaCost relies on the internal cost-sensitive procedure in the first step. Getting a good performing internal model in the first step is crucial to getting a good performing final model in the third step.

MetaCost in its original form (Domingos, 1999) assumes that its internal cost-sensitive procedure is obtained by applying the Bayes optimal prediction rule to an existing error-based procedure. Thus, the cost-sensitive procedure *C* consists of first getting the class probability from a model *h* defined as follows. Choose an error-based procedure *E* which employs a training set *T* and a base learning algorithm *L* to induce the model *h* = *E*(*T*,*L*), without reference to cost. Given a new example *x*, *h* produces a class probability for each class:

$P(i \mid x) = h(x)$, for each class *i*.

Then, apply the Bayes rule or minimum expected cost criterion:

$$H^*(x) = \underset{j}{\arg\min} \sum_i P(i \mid x)\, cost(i, j)$$

However, a cost-sensitive procedure, that takes cost directly into consideration in the training process is another option. In this paper, both types of cost-sensitive procedures are used to evaluate their effects on MetaCost. We use the error-minimizing boosting algorithm AdaBoost in place of *E* and a cost-sensitive version of the boosting procedure in place of *C*. Both AdaBoost and the cost-sensitive boosting procedures have been described in the last section.

Table 2: Description of data sets

Data set	Size	No. of Attributes	
		Numeric	Nominal
breast cancer (W)	699	9	0
liver disorder	345	6	0
credit screening	690	6	9
echocardiogram	131	6	1
solar flare	1389	0	10
heart (Cleveland)	303	13	0
hepatitis	155	6	13
horse colic	368	7	15
house-voting 84	435	0	16
hypothyroid	3168	7	18
kr-vs-kp	3169	0	36
pima diabetes	768	8	0
sonar	208	60	0
tic-tac-toe	958	0	9

EXPERIMENTS

In this section, we empirically evaluate the performances of the cost-sensitive boosting procedures as well as the MetaCost procedure. Fourteen two-class natural data sets from the UCI machine learning repository (Blake, Keogh & Merz, 1998) are used in the experiments. Table 2 shows the characteristics of the data sets. They are breast cancer (Wisconsin), liver disorder, credit screening, echocardiogram, solar flare, heart disease (Cleveland), hepatitis, horse colic, house-voting 84, hypothyroid, king-rook versus king-pawn, pima Indian diabetes, sonar and tic-tac-toe data sets. Only two-class data sets are used because AdaBoost, in its basic form, is designed for two-class problems only (Schapire & Singer, 1999).

Recall that a cost matrix must be assigned for AdaBoost and its variants. Suppose i is the majority class, that is, $P(i) > P(j)$, then assign $cost(i,j)$=1 and $cost(j,i)=f > 1$. This means misclassifying a minority class example is f times more costly than misclassifying a majority class example. This simulates the situation often found in practice where it is most important to correctly classify the rare classes. In this paper, all correct classifications are assumed to have no cost, that is, for all i, $cost(i,i)$=0.

For each of the data sets, we report the sum of three averages, where each average is the result of a run of two 10-fold cross-validations using a fixed cost factor. The three cost factors used are f = 2, 5, and 10. Thus, in each set of experiments (having three cost factors and 14 data sets), there are altogether 42 runs. We also show the number of runs in which one algorithm wins, draws and loses to another to indicate whether the difference in performance is significant. We conduct a one-tail sign-test on the number of win/loss records and consider the difference is significant only if the level is better than 95%.

The key measure to evaluate the performance of the algorithms for cost-sensitive classification is the *total cost of misclassifications* made by a classifier on a test set (i.e., $\Sigma_m\, cost(actual(m), predicted(m))$). In addition, we also use a second measure: the *number of high cost errors*. It is the number of misclassifications associated with costs higher than unity made by a classifier on a test set. The second measure is used primarily to explain the behavior of the algorithms.

Empirical Evaluation of Cost-Sensitive Boosting Algorithms

Comparison With AdaBoost

We evaluate the four variants of cost-sensitive boosting and AdaBoost in this section and investigate the reason(s) why one is performing better than the other. We first take AdaBoost as the baseline for comparison because it is the base algorithm for the cost-sensitive variants. To investigate the relative performance among the variants, we then take AdaCost, the most complex adaptation, as the baseline comparison. Note that all algorithms use the minimum expected cost criterion, including AdaBoost.

Tables 3 and 4 show the experimental results of AdaBoost, CSB0, CSB1, CSB2 and AdaCost in terms of cost and the number of high cost errors, respectively. The tables include ratios taking AdaBoost as the baseline, reported in the last four columns. A ratio of less than 1 represents an improvement due to a variant relative to AdaBoost. The mean values for each of the five algorithms and the geometric mean of the ratios are provided in the last row.

From the last row in each table note that:

- In terms of cost, CSB0, CSB1 and CSB2 have comparable performance. Each

Table 3: Cost for AdaBoost, CSB0, CSB1, CSB2 and AdaCost

	Cost					Cost Ratio			
Data set	AdaBoost	CSB0	CSB1	CSB2	AdaCost	CSB0 vs AdaB	CSB1 vs AdaB	CSB2 vs AdaB	AdaC vsAdaB
bcw	14.55	14.85	15.85	14.65	14.70	1.02	1.09	1.01	1.01
bupa	71.95	59.60	60.25	58.45	108.40	.83	.84	.81	1.51
crx	68.00	59.35	60.80	59.60	66.80	.87	.89	.88	.98
echo	35.75	22.75	24.40	24.10	38.20	.64	.68	.67	1.07
flare	315.85	259.30	246.75	244.00	318.20	.82	.78	.77	1.01
h-d	42.55	36.50	34.75	35.65	40.45	.86	.82	.84	.95
hepa	24.35	20.20	20.20	19.60	22.30	.83	.83	.80	.92
horse	58.05	47.65	50.05	49.55	58.05	.82	.86	.85	1.00
hv84	11.25	11.35	10.40	11.85	10.80	1.02	.92	1.05	.96
hypo	26.10	25.15	24.65	26.95	24.40	.96	.94	1.03	.93
krkp	14.55	19.25	21.20	19.60	16.70	1.32	1.46	1.35	1.15
pima	123.15	111.55	113.15	110.95	136.85	.91	.92	.90	1.11
sonar	26.75	23.25	23.25	23.65	23.70	.87	.87	.88	.89
ttt	52.10	76.00	80.60	75.65	68.15	1.46	1.55	1.45	1.31
mean	63.21	56.20	56.16	55.30	67.69				
geomean						.92	.94	.93	1.05

Table 4: High cost errors for AdaBoost, CSB0, CSB1, CSB2 and AdaCost

	High Cost Errors					High Cost Error Ratio			
Data set	AdaBoost	CSB0	CSB1	CSB2	AdaCost	CSB0 vs AdaB	CSB1 vs AdaB	CSB2 vs AdaB	AdaC vs AdaB
bcw	1.25	1.00	0.85	0.90	1.10	.80	.68	.72	.88
bupa	9.00	6.00	5.40	5.60	16.60	.67	.60	.62	1.84
crx	9.05	7.05	5.25	5.70	9.45	.78	.58	.63	1.04
echo	5.05	2.85	2.80	2.65	5.15	.56	.55	.52	1.02
flare	55.90	43.80	37.60	39.90	56.80	.78	.67	.71	1.02
h-d	5.40	4.20	3.05	3.45	5.85	.78	.56	.64	1.08
hepa	3.50	2.50	2.35	2.45	3.35	.71	.67	.70	.96
horse	9.00	5.95	5.60	5.70	9.30	.66	.62	.63	1.03
hv84	1.20	0.95	0.65	0.75	1.20	.79	.54	.63	1.00
hypo	3.70	3.35	3.00	3.10	3.55	.91	.81	.84	.96
krkp	2.00	1.55	1.50	1.45	1.95	.78	.75	.73	.98
pima	16.55	11.10	10.80	10.45	20.05	.67	.65	.63	1.21
sonar	2.65	2.00	1.15	1.55	2.55	.75	.43	.58	.96
ttt	4.30	2.85	1.55	1.65	7.15	.66	.36	.38	1.66
mean	9.18	6.80	5.83	6.09	10.29				
geomean						.73	.60	.63	1.09

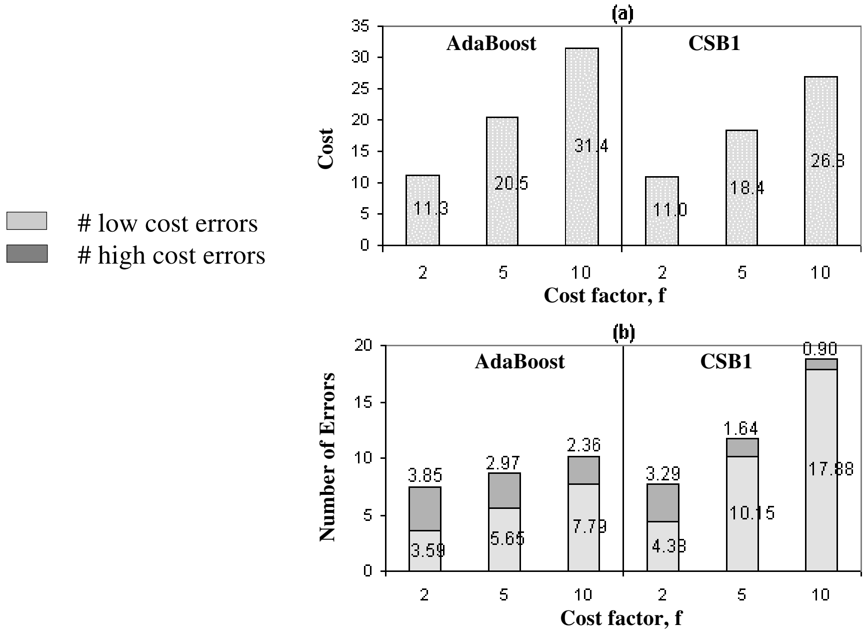
performs better than AdaBoost. The mean relative cost is reduced by 8%, 6% and 7%, respectively over the 14 data sets. AdaCost performs worse than AdaBoost with a mean relative cost increase of 5%.

- In terms of high cost errors, CSB0, CSB1 and CSB2 perform significantly better than AdaBoost in all data sets. The mean relative reductions are 27%, 40% and 37%. Again, AdaCost performs worse than AdaBoost with a mean relative increase of 9%.

Relative to AdaBoost, the first three variants work harder in reducing high cost errors when the cost factor increases. Figure 4 shows this effect for CSB1. The mean high cost error ratio for CSB1 versus AdaBoost decreases from .82 to .49 and then to .21 as the cost factor increases from 2 to 5 and then to 10. Similar trends emerge using cost, though to a lesser degree. The mean cost ratios are .96, .94 and .90 for *f*= 2, 5, and 10, respectively.

Even though the first three variants are adept at reducing high cost errors, net increase in cost can arise. For example note the higher relative costs in Table 3 for the krkp and ttt data sets, yet the reduced incidence of high cost error in Table 4. In these two data sets, cost increases because the gain from reducing high cost errors is outweighed by introducing more low cost errors. Figure 4(b) shows the trade-off between the two types of errors for AdaBoost and CSB1 as cost factor increases.

Figure 4: The average cost and numbers of high and low cost errors for AdaBoost and CSB1 as cost factor increases. Each bar shows the average value over all data sets for each cost factor. (a) CSB1 has lower average cost than AdaBoost in each cost factor. (b) CSB1 has fewer high cost errors than AdaBoost in each cost factor, but more low cost errors.

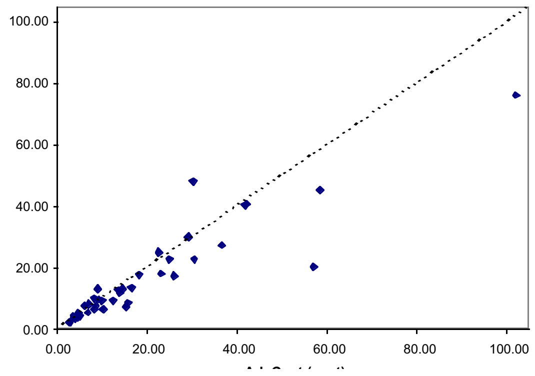


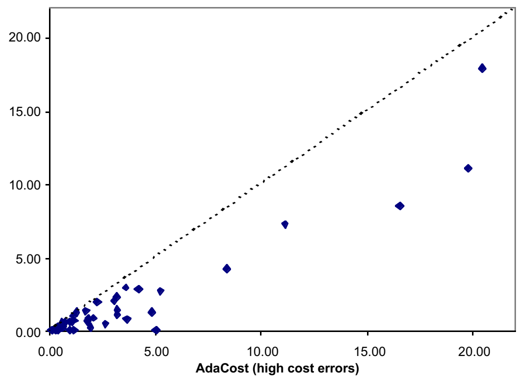
The mean high cost errors for the four variants in Table 4 reveal some interesting features of these algorithms. The simplest variant has a mean high cost error of 6.80; the next two variants have comparable means of 5.83 and 6.09; and the most complex variant has a mean of 10.29. For the two simplest variants (CSB0 versus CSB1), the mean ratio is 1.23. CSB0 performs significantly worse than CSB1. The win/tie/loss record is 7/8/27, which is significant at a level better than 99.9%. That is, it is evident that *the confidence-rated prediction used in the weight-update rule, which is the only difference between the two algorithms, plays a significant role in minimizing high cost errors for cost-sensitive boosting*.

The comparison between CSB1 and CSB2 shows that incorporating an additional factor α into the weight-update rule (the only difference between the two) does not help to further minimize high cost errors. CSB1 versus CSB2 has a mean ratio of .94. The win/tie/loss record is 21/10/11, which is not significant at the 95% level. In fact, incorporating a second factor β into the rule, as in AdaCost, increases the number of high cost errors significantly.

Relative to AdaCost, both CSB1 and CSB2 perform better with a mean reduction of 10% in relative cost and a mean reduction of more than 40% in relative high cost errors. Figure 5 shows the relative performance of all 42 runs for CSB1 versus AdaCost. The distribution of the scatter plots for CSB2 versus AdaCost is similar to Figure 5. Note that to enhance readability, one extreme point in the cost scatter plot is excluded.

Figure 5: CSB1 versus AdaCost in terms of cost and the number of high cost errors in all 42 runs





In terms of cost, CSB1 performs better than AdaCost in 30 runs and worse in 12 runs; and CSB2 performs better than AdaCost in 31 runs and worse in 11 runs. Both results are significant at a level better than 99.5%. In terms of high cost errors, CSB1 has fewer errors than AdaCost in 40 runs and has 2 equal runs; and CSB2 has fewer errors in all except 1 equal run. In addition, AdaCost is more likely to produce much higher cost and more high cost errors than either CSB1 or CSB2. This latter result leads us to investigate why AdaCost performs poorly in the next section.

Why Does AdaCost Perform Poorly?

Fan et al. (1999) require that β_+ is nonincreasing with respect to c_n. This means that the reward for correct classification is low when the cost is high, and vice versa. This seems to be counter-intuitive, and it could be the source of AdaCost's poor performance.

We alter the form of β_δ and assume β_+ is non-decreasing as is β_-, and they are both equal to the cost of misclassification, that is, $\beta_+ = \beta_- = c_n$. We denote the resultant algorithm as AdaCost(β_1). We also employ a second modification, called AdaCost(β_2), which is identical to AdaCost(β_1) except that β_δ is excluded in Equation (14). In other words, Equation (5) is used instead. This is to investigate whether cost needs to be taken into consideration in Step (ii) of the boosting procedure.

The experimental results of these two variants of AdaCost are shown in Tables 5 and 6. Taking AdaCost as the baseline, ratios are also shown.

Changing to $\beta_+ = \beta_- = c_n$, there is a mean reduction of 11% in relative cost over the original algorithm, and a mean reduction of 30% in relative high cost errors. This is on a par with CSB0, CSB1 and CSB2 in terms of cost. However performance remains worse than CSB1 and CSB2 in terms of high cost errors. Taking AdaCost(β_1) as the base, the mean high cost error ratios are .78 and .83 for CSB1 and CSB2, respectively. The wins/draw/loss records are 31/9/1 for CSB1 and 34/5/3 for CSB2, which are both significant at a level better than 99.99%.

This result clearly shows that the original form of β_δ and the requirement that β_+ is nonincreasing were poorly selected (Fan et al., 1999). In fact, Fan et al.'s derivation of the upper bound on the training misclassification cost implies both β_+ and β_- must be nondecreasing. Note also that in its original form, AdaCost does not reduce to AdaBoost when $c_n = 1$ because $\beta_+ \neq \beta_-$ whereas AdaCost(β_1) does reduce to AdaBoost under the same condition.

The second result in this section demonstrates that AdaCost(β_2) performs comparably with AdaCost(β_1). Both in terms of cost and high cost errors, AdaCost(β_2) wins in eight data sets and loses in six data sets to AdaCost(β_1). This result suggests that the inclusion of β_δ in Step (ii) of the boosting procedure has minimal effect on its overall performance, and it can be excluded.

The Effects of Weight Initialization and Minimum Expected Cost Criterion

In this section, we investigate the effects of weight initialization and the

Table 5: Cost for two variants of AdaCost

	Cost		Cost Ratio	
Data set	AdaC(β_1)	AdaC(β_2)	AdaC(β_1) vs AdaC	AdaC(β_2) vs AdaC
bcw	14.65	15.80	1.00	1.07
bupa	57.65	60.30	.53	.56
crx	57.00	59.60	.85	.89
echo	26.60	28.05	.70	.73
flare	273.75	287.15	.86	.90
h-d	38.05	37.05	.94	.92
hepa	20.35	19.35	.91	.87
horse	50.45	49.80	.87	.86
hv84	11.15	10.85	1.03	1.00
hypo	25.10	25.00	1.03	1.02
krkp	18.75	18.30	1.12	1.10
pima	111.05	112.25	.81	.82
sonar	23.30	22.00	.98	.93
ttt	70.40	67.20	1.03	.99
mean	57.02	58.05		
geomean			.89	.89

Table 6: High cost errors for two variants of AdaCost

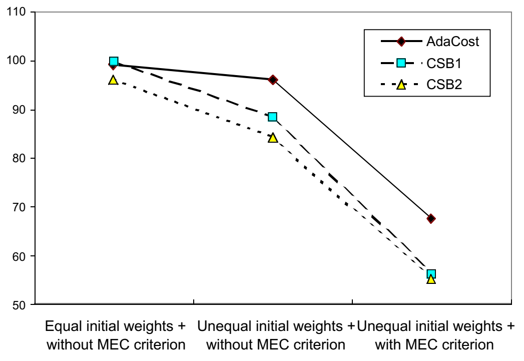
	High Cost Errors		High Cost Error Ratio	
Data set	AdaC(β_1)	AdaC(β_2)	AdaC(β_1) vs AdaC	AdaC(β_2) vs AdaC
bcw	0.95	1.05	.86	.95
bupa	5.90	6.30	.36	.38
crx	6.55	7.05	.69	.75
echo	3.70	3.75	.72	.73
flare	48.80	50.95	.86	.90
h-d	4.35	4.10	.74	.70
hepa	2.75	2.50	.82	.75
horse	6.50	6.35	.70	.68
hv84	1.00	0.90	.83	.75
hypo	3.35	3.30	.94	.93
krkp	1.60	1.45	.82	.74
pima	12.05	12.25	.60	.61
sonar	1.85	1.60	.73	.63
ttt	3.10	2.65	.43	.37
mean	7.32	7.44		
geomean			.70	.68

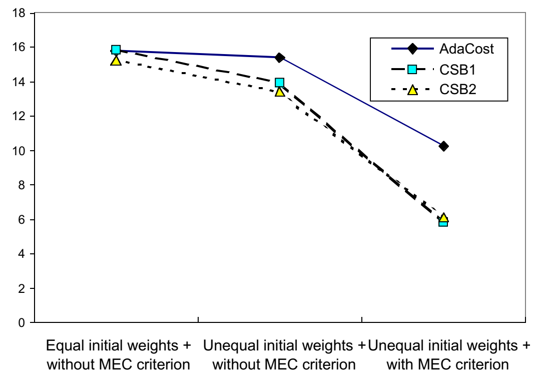
minimum expected cost criterion on the performance of CSB1, CSB2 and AdaCost. For each of the cost-sensitive boosting algorithms, we change parameters one at a time. Begin with equal initial weights and the maximum vote criterion; then change only the initial weight setting as defined in Equation (2); and finally introduce unequal initial weights and the minimum expected cost criterion. The results are summarized in Figure 6.

From both in terms of cost and high cost errors, note that:

- Initial weights proportional to cost improve the variants' performance with a small percentage gain compared with using equal initial weights.

Figure 6: The effects of initial weights and the minimum expected cost (MEC) criterion on CSB1, CSB2 and AdaCost





- The minimum expected cost criterion contributes significantly to the performance of cost-sensitive boosting, compared with the maximum vote criterion. The three variants which use the minimum expected cost criterion reduce mean relative cost by more than 30% over variants that don't. In terms of high cost errors, AdaCost improves its performance by a mean relative reduction of 43%, and both CSB1 and CSB2 improve by a mean relative reduction of more than 60%.
- Without the minimum expected cost criterion, the influence of α on the weight-update rule is the only reason for the better performance of CSB2 over CSB1. With the criterion, the influence becomes minimal as CSB1 and CSB2 perform comparably.

In fact, CSB1 performs better by excluding α altogether. This is in the minimum expected cost criterion described in Equations (9) and (10). Comparing CSB1 without and with α, the mean ratios are 1.00 and .96 for cost and high cost error, respectively. The latter accounts for CSB1 without a to win in 21 runs, lose in 7 runs and draw in 14 runs. This is significant at a level better than 99%. As a consequence, *a simplified CSB1 procedure is prominent, consisting of only Steps (i) and (iii), in which 'classify T using H_k' can be done in conjunction with the weight update,* as there is no need to compute r_k and α_k.

Using Confidence Levels of All Classes in Each Model

Thus far, we use the confidence level of the predicted class only (which has the highest level among all classes) in each of the *K* models. This follows directly from the AdaBoost procedure. However, the minimum expected cost criterion may perform better if the confidence levels of all classes in each model are taken into consideration. We investigate this issue in this section.

Let $H^i_k(x)$ be the confidence level for class *i* from model *k* when classifying example *x*. We modify Equation (10) accordingly, and exclude α_k since it has minimal effect on the performance.

$$H^*(x) = \underset{j}{\arg\ \min} \sum_i \sum_k H^i_k(x)\ cost(i,j) \tag{16}$$

Variants that use confidence levels of all classes are denoted as AdaBoost' and CSB1', for example. Tables 7 and 8 show the results using AdaBoost' and CSB1'.

In terms of high cost errors, AdaBoost' and CSB1' perform better than AdaBoost and CSB1, respectively, with a mean relative reduction of more than 25%. But the result is mixed in terms of cost. Using confidence levels of all classes improves AdaBoost’s performance by a mean reduction of 9% in relative cost. However, it increases the mean relative cost by 2% for CSB1.

Note that both AdaBoost and AdaBoost' induce exactly the same *K* models. This is the same for CSB1 and CSB1'. Thus, the difference in performance is solely due to how the same models are used in conjunction with the minimum expected cost criterion during classification. Confidence levels of all classes provide complete information on how to minimize high cost errors. This method is almost always better in reducing high cost errors than that using only the confidence level of the predicted class. The reduction comes at a price of increasing low cost errors. At times, the increase of low cost errors could outweigh the gain due to the high cost error reduction. This explains why CSB1' has higher cost than CSB1 in eight out of the fourteen data sets while reducing the number of high cost errors in all but one data set.

In terms of high cost errors, the best performing algorithm CSB1' performs significantly better than AdaBoost' with a mean ratio of .62 and the win/tie/loss record is 37/4/1. It performs marginally better than its closest contender CSB2', with a mean ratio of .93. In terms of cost, AdaBoost' and the four variants have almost similar performance. Figure 7 shows the performances of the four variants relative to AdaBoost'.

Figure 8 shows the mean tree size and high cost errors over the fourteen data sets for AdaBoost' and the four variants. The tree size is the total number of nodes for all models generated in each boosting session. The result shows a strong correlation between an algorithm that produces small trees and its success in reducing high cost errors among the five variants. Bauer and Kohavi (1999) have a

Table 7: Cost for AdaBoost' and CSB1' which employ confidence levels of all classes for each model.

	Cost		Cost Ratio	
Data set	AdaBoost'	CSB1'	AdaBoost' vs AdaBoost	CSB1' vs CSB1
bcw	14.45	16.40	.99	1.03
bupa	58.50	57.05	.81	.95
crx	59.30	65.85	.87	1.08
echo	24.90	23.20	.70	.95
flare	242.75	223.95	.77	.91
h-d	40.45	37.45	.95	1.08
hepa	22.65	19.90	.93	.99
horse	49.00	52.75	.84	1.05
hv84	10.95	11.15	.97	1.07
hypo	26.10	24.30	1.00	.99
krkp	14.15	20.70	.97	.98
pima	111.95	113.10	.91	1.00
sonar	26.05	25.55	.97	1.10
ttt	58.70	91.25	1.13	1.13
mean	54.28	55.90		
geomean			.91	1.02

Table 8: High cost errors for AdaBoost' and CSB1' which employ confidence levels of all classes for each model

	High Cost Errors		High Cost Error Ratio	
Data set	AdaBoost'	CSB1'	AdaBoost' vs AdaBoost	CSB1' vs CSB1
bcw	1.15	0.90	.92	1.06
bupa	4.40	2.90	.49	.54
crx	7.15	4.45	.79	.85
echo	2.25	1.65	.45	.59
flare	33.30	27.85	.60	.74
h-d	4.20	2.45	.78	.80
hepa	2.75	1.80	.79	.77
horse	5.35	3.55	.59	.63
hv84	1.05	0.55	.88	.85
hypo	3.70	2.90	1.00	.97
krkp	1.85	1.40	.93	.93
pima	10.40	7.40	.63	.69
sonar	2.50	0.70	.94	.61
ttt	2.25	0.95	.52	.61
mean	5.88	4.25		
geomean			.71	.74

related finding: For data sets in which tree size increases in boosting trials, AdaBoost fails to reduce the total errors of a single tree, and vice versa.

Figure 9 shows that the number of high cost leaves and low cost leaves for each algorithm. A high cost leaf in a tree is the terminal node in which the predicted class incurs high cost if it misclassifies a test example; a low cost leaf is similarly defined.

Except AdaBoost', the other variants produce smaller trees as cost factor increases. This is reflected in the reduced number of both high cost leaves and low cost leaves. CSB1' produces the smallest trees in all three cost factors. The result shows that cost-sensitive induction is able to adapt to different cost scenarios by producing different trees, whereas cost-insensitive induction always uses the same trees.

It is interesting to compare the difference between the performance of the old version of AdaBoost that does not use confidence-rated predictions (Schapire, Freund, Bartlett, & Lee, 1997) with AdaBoost' described in this chapter. Table 9 shows AdaBoost' is better than the old version with a relative mean cost reduction of 12%. While the performance difference is small (within 10%) in eleven out of the fourteen data sets, the performance difference is much larger in favour of AdaBoost' in the other three data sets; in particular in the krkp data set, the relative performance

Figure 7: The geometric mean cost ratio and high cost error ratio for the four cost-sensitive boosting algorithms relative to AdaBoost'

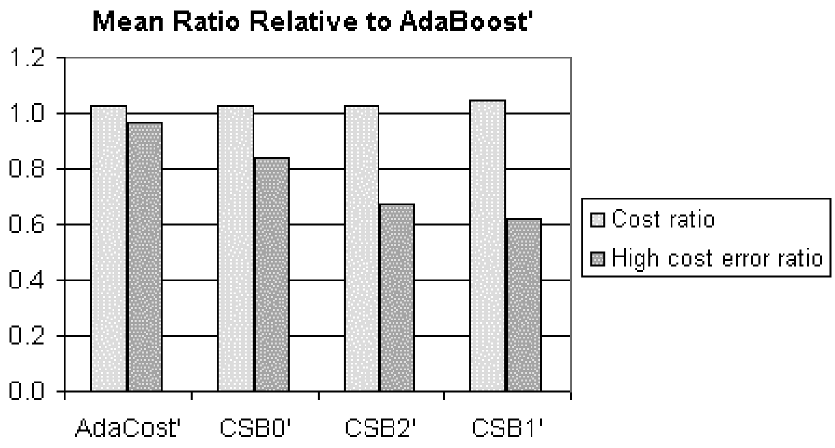


Figure 8: Correlation between tree size and high cost errors

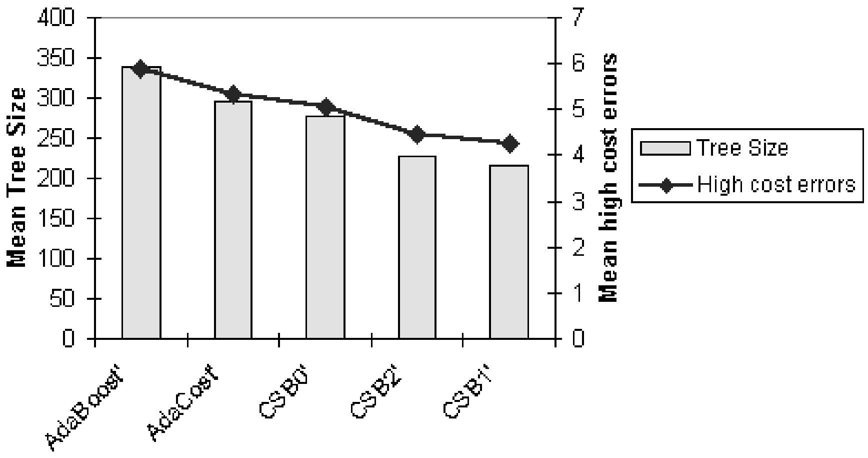
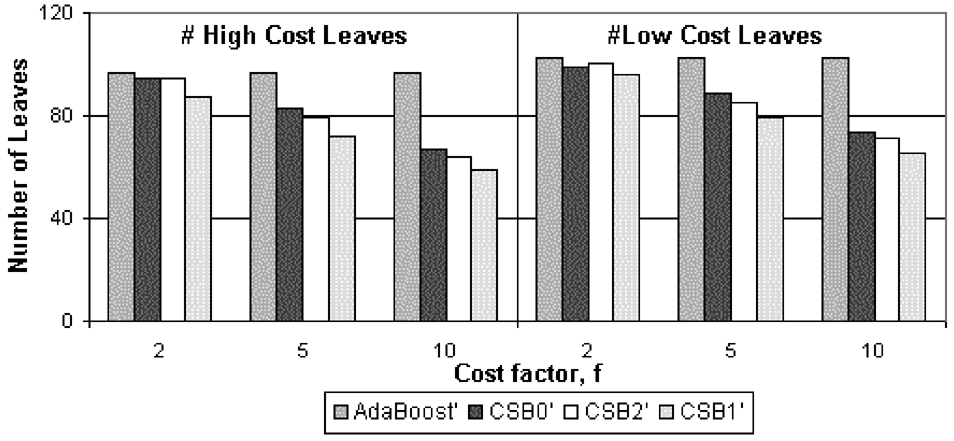


Figure 9: The average number of high cost leaves and low cost leaves in the trees produced by AdaBoost', CSB0', CSB2'and CSB1' as cost factor increases.



can be more than 300% in favor of AdaBoost'. The same table also provides results for C4.5CS, and it shows that both versions of cost-sensitive AdaBoost perform significantly better than C4.5CS with a great margin.

Empirical Evaluation of MetaCost

In this section, we empirically evaluate the performance of MetaCost's final model H and its internal classifier H^* produced by boosting and bagging. We denote MetaCost_A as the algorithm that uses AdaBoost in MetaCost, and MetaCost_CSB uses CSB (CSB is equivalent to CSB0 whose weight update rule is described in Equation (11)), and MetaCost_B uses bagging. The experimental setting is the same as the previous section unless stated otherwise. When bagging is used, we also employ 10 classifiers in each run, as in boosting.

MetaCost using AdaBoost and CSB

Table 10 shows the results of the comparison between MetaCost_A and AdaBoost and between MetaCost_CSB and CSB in terms of misclassification costs and cost ratios. The relative performance between the two versions of MetaCost are shown in the last column. A summary of the mean costs and geometric mean ratios are also shown.

In terms of cost, we have the following observations:

- MetaCost usually does not perform better than its internal classifier. AdaBoost performs better than MetaCost_A in nine data sets, performs worse in four data sets, and has equal performance in one data set. CSB performs better than MetaCost_CSB in ten datasets and only marginally worse in four datasets. MetaCost retains only a portion of performance of its internal classifier. Using CSB, MetaCost_CSB retains between 68% and 99% of CSB's performance. In only six out of fourteen data sets, MetaCost_CSB performs comparably to CSB, with a maximum gain of 2% in relative cost.

Table 9: Comparison of C4.5CS, the old version of AdaBoost and AdaBoost' in terms of cost.

	Cost			Cost Ratio
Data Set	C4.5CS	AdaBoost(old)	AdaBoost'	AdaBoost(old) vs AdaBoost'
bcw	29.80	14.90	14.45	1.03
bupa	84.60	56.75	58.50	.97
crx	69.20	63.45	59.30	1.07
echo	30.25	22.85	24.90	.92
flare	251.05	219.45	242.75	.90
h-d	61.60	36.95	40.45	.91
hepa	32.10	21.55	22.65	.95
horse	52.75	49.80	49.00	1.02
hv84	14.60	14.15	10.95	1.29
hypo	25.80	27.60	26.10	1.06
krkp	23.90	52.75	14.15	3.73
pima	151.35	108.55	111.95	.97
sonar	57.75	24.85	26.05	.95
ttt	110.85	80.20	58.70	1.37
mean	71.11	56.70	54.28	
geomean				1.12

- MetaCost_CSB is not significantly better than MetaCost_A with six wins, seven losses, and one draw. Nevertheless, MetaCost_CSB performs better than MetaCost_A on average across the fourteen data sets. In cases where MetaCost_CSB performs better, the gain can be as much as 60% in relative cost as in the kr-vs-kp data set. In cases where MetaCost_CSB performs worse, the maximum loss is only 22% in relative cost in the sonar data set.

MetaCost Using Bagging

MetaCost originally uses bagging as its internal classifier (Domingo, 1999). The aims of this section are to investigate how well it performs compared with MetaCost using boosting and whether MetaCost's final model performs better than bagging. Table 11 shows the result.

MetaCost using bagging is found to perform significantly worse than bagging in terms of cost. Bagging performs better, though not significantly, than AdaBoost in terms of cost (AdaBoost' is comparable to bagging, nevertheless. Refer to results in Table 9.) But the result does not carry over to MetaCost; both MetaCost_A and MetaCost_B have comparable mean relative performance across the fourteen data sets. Boosting has been shown to outperform bagging (Bauer & Kohavi, 1999) in reducing total errors. Our experiment here clearly shows that the result does not carry over to cost-sensitive classification and MetaCost in terms of cost.

In addition, we compute the percentage of training examples in which the original class is altered to a different class in Step (ii) of the MetaCost procedure. Bagging modified an average of 9% of training examples across the twenty-four datasets, and AdaBoost modified an average of 22%. AdaBoost modified more examples than bagging in all datasets. The detailed results are given in Table 12. The aggressiveness of AdaBoost in relabeling classes indicates that its prediction is comparatively easier to be influenced by the cost factor. This in turn implies that AdaBoost's class probabilities could be more evenly distributed across different classes than those by bagging.

Table 10: Comparison of MetaCost, AdaBoost and CSB in terms of cost

Data Set	MetaC_A	AdaB vs MetaC_A	MetaC_CSB	CSB vs MetaC_CSB	MetaC_CSB vs MetaC_A
	cost	cost ratio	cost	cost ratio	cost ratio
bcw	18.50	.85	20.35	.74	1.10
bupa	57.75	.99	57.90	.99	1.00
crx	64.25	1.00	61.10	.92	.95
echo	23.15	.97	24.10	1.01	1.04
flare	215.15	1.02	226.05	1.02	1.05
h-d	39.30	.96	39.70	.95	1.01
hepa	25.15	.90	21.95	.95	.87
horse	53.40	.95	50.65	.99	.95
hv84	13.45	1.14	11.30	1.02	.84
hypo	27.60	1.11	25.45	1.01	.92
krkp	54.45	1.15	21.95	.86	.40
pima	110.70	.99	112.35	.96	1.01
sonar	26.40	.99	32.25	.76	1.22
ttt	114.95	.73	120.05	.68	1.04
mean	60.30		58.94		
geomean		.98		.91	.93

Table 11: Result for MetaCost using bagging, AdaBoost and CSB in terms of cost

Data Set	Bagging	MetaC_B	Bagging vs MetaC_B	Bagging vs AdaB	MetaC_B vs MetaC_A	MetaC_B vs MetaC_CSB
	cost	cost	cost ratio	cost ratio	cost ratio	cost ratio
bcw	13.75	21.10	.65	.92	1.14	1.04
bupa	63.30	74.85	.85	1.12	1.30	1.29
crx	59.65	57.55	1.04	.94	.90	.94
echo	26.15	29.40	.89	1.14	1.27	1.22
flare	243.40	242.75	1.00	1.11	1.13	1.07
h-d	38.05	49.50	.77	1.03	1.26	1.25
hepa	21.65	25.25	.86	1.00	1.00	1.15
horse	48.35	51.15	.95	.97	.96	1.01
hv84	10.55	11.10	.95	.75	.83	.98
hypo	24.45	23.25	1.05	.89	.84	.91
krkp	15.20	24.45	.62	.29	.45	1.11
pima	113.15	128.60	.88	1.04	1.16	1.14
sonar	25.85	40.15	.64	1.04	1.52	1.24
ttt	59.80	107.95	.55	.75	.94	.90
mean	54.52	63.36				
geomean			.82	.89	1.01	1.08

The Effect of K

It has been shown that increasing *K*, the number of classifiers, in the boosting procedure can reduce the number of errors. It is interesting to see the effect of *K* on MetaCost and its internal boosting procedures in terms of cost and high cost errors.

Figure 10 shows an example of performance comparison between MetaCost_CSB and CSB as *K* in the boosting procedure increases from 5, 10, 20, 50, 75 to 100 classifiers in the satellite dataset. In terms of high cost errors, both MetaCost_CSB and CSB initially reduce the errors as *K* increases and then stabilize. Although CSB stabilizes earlier at *K*=20, with comparison to MetaCost_CSB which stabilizes at *K*=75, CSB always has fewer errors than MetaCost_CSB. Both MetaCost_CSB and CSB have similar profiles in the figures. As *K* increases cost initially falls, but then increases. For MetaCost_CSB, the cost increases beyond the point at which the high cost errors stabilized at *K*=75; for CSB it is at *K*=20. The increased total cost is due to the increase in low cost errors while the boosting procedure continues its effort to reduce high cost errors, eventually without success.

In terms of tree size, MetaCost_CSB produces a smaller tree as *K* increases, from a size of 550 nodes at *K*=5 to 398 at *K*=100. On the other hand, CSB produces a combined tree size of 2166 nodes at *K*=5, and increases to 18881 at *K*=100.

The results in this section indicate that it is possible to use the number of high cost errors as a stopping criterion to determine the optimal number of trees in the cost-sensitive boosting procedure.

CONCLUSIONS

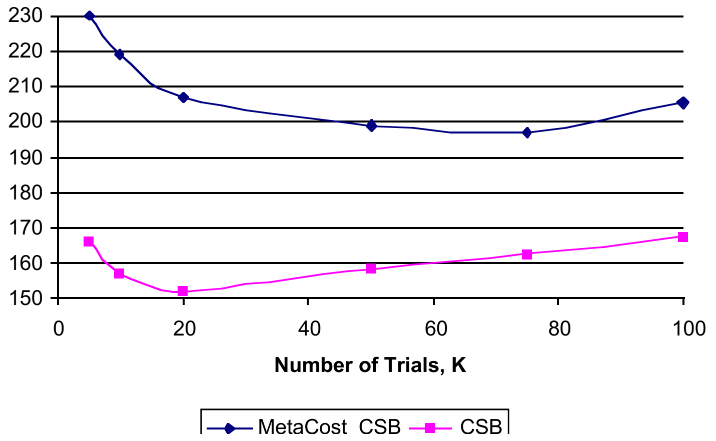
This chapter provides a systematic study on the various issues relating to two cost-sensitive algorithms: cost-sensitive boosting and MetaCost. In both cases we have found that the current algorithms can be simplified without loss of performance. For instance, among the four increasingly complex variants of cost-sensitive boosting algorithms studied, one of the simplest (CSB1') has been shown to be comparable to the others in terms of cost but better in terms of model size. In the study involving MetaCost, we find that the internal classifier does not benefit from the meta-learning process in terms of cost. This occurs for both boosting and bagging as its internal classifier. Based on our results, we do not recommend using MetaCost when the aim is to minimize the misclassification cost. MetaCost is only recommended if the aim is to have a more comprehensive model and the user is willing to sacrifice part of the performance.

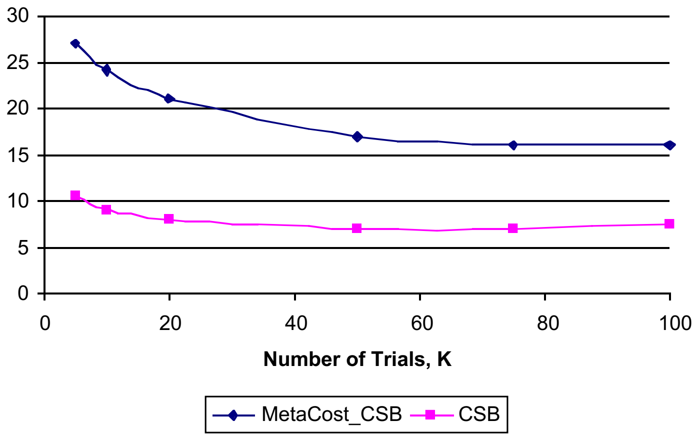
We have studied several variations of AdaBoost by changing the following independent variables: the weight update rules (which produces the four main variants), with or without (i) confidence-rated predictions, (ii) unequal initial instance weights, (iii) the minimum expected cost criterion, and (iv) confidence levels of all classes in the minimum expected cost criterion. We have measured the following dependent variables: misclassification cost, number of high cost errors and tree size.

Table 12: Average percentage of training examples in which the original class is altered

Data Set	MetaCost_Bagging	MetaCost_AdaBoost
bcw	2.3	5.6
bupa	14.3	41.1
crx	7.6	18.6
echo	16.7	43.3
flare	23.9	29.2
h-d	9.3	26.6
hepa	7.4	20.1
horse	18.1	36.4
hv84	2.5	7.1
hypo	0.4	1.7
krkp	0.3	4.9
pima	11.3	32.7
sonar	10.1	25.6
ttt	6.5	21.6
mean	9.3	22.5

Figure 10: The satellite data set: Comparing MetaCost_CSB with CSB in terms of cost and the number of high cost errors as K *in the boosting procedure increases*





It turns out that reducing high cost errors often increases low cost errors. In some situations, the gain of reducing high cost errors is outweighed by introducing more low cost errors, resulting in a net increase in cost.

The minimum expected cost criterion is the major contributor to the improvement of all cost-sensitive adaptations of AdaBoost. Using confidence levels of all classes works better with the criterion than using only the highest confidence level.

Incorporating the confidence-rated predictions in the weight-update rule is an important ingredient in cost-sensitive boosting. A new variant, which uses the confidence-rated predictions but not other factors such as α and β, performs best at minimizing high cost errors. It almost always performs better than AdaBoost (with the minimum expected cost criterion) in this measure in our experiments. This variant simplifies boosting by excluding Step (ii) in the AdaBoost procedure. We also find a strong correlation between an algorithm that produces small tree size and its success in reducing high cost errors.

AdaCost performs worse than other forms of adaptations because it uses an inappropriate form of β. With an appropriate form, it improves the performance substantially, and β can be excluded from Step (ii) of the boosting procedure without losing performance. Nevertheless, AdaCost is not the best choice for cost-sensitive classification because even with an appropriate form of β, it still performs worse than the other three variants of cost-sensitive boosting in terms of tree size and high cost errors.

An interesting result we find is that it is possible to use the number of high cost errors as a stopping criterion to determine the optimal number of trees in the cost-sensitive boosting procedure.

ACKNOWLEDGMENTS

The anonymous reviewers have provided helpful comments to improve this chapter. Ruhul Sarker assisted with the editing of the final version of this chapter. Thanks to Ross Quinlan for providing C4.5.

REFERENCES

Blake, C., Keogh, E. & Merz, C.J. (1998). *UCI Repository of machine learning databases* [http:// www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California.

Bauer, E. & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*, 105-139. Boston: Kluwer Academic Publishers.

Bradford, J.P., Kunz, C., Kohavi, R., Brunk, C. & Brodley, C.E. (1998). Pruning Decision Trees with Misclassification Costs. *Proceedings of the Tenth European Conference on Machine Learning, LNAI-1398*, 131-136. Springer-Verlag.

Breiman, L., J.H. Friedman, R.A. Olshen & C.J. Stone (1984). *Classification And Regression Trees*, Belmont, CA: Wadsworth.

Domingos, P. (1999). MetaCost: A General Method for Making Classifiers Cost-Sensitive. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 155-164. ACM Press.

Fan, W., Stolfo, S.J., Zhang, J. & Chan, P.K. (1999). AdaCost: Misclassification cost-sensitive boosting. *Proceedings of the Sixteenth International Conference on Machine Learning*, 97-105. San Francisco: Morgan Kaufmann.

Freund, Y. & Schapire, R.E. (1996). Experiments with a new boosting Algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*. 148-156. San Francisco: Morgan Kaufmann.

Knoll, U., Nakhaeizadeh, G., & Tausend, B. (1994). Cost-Sensitive Pruning of Decision Trees. *Proceedings of the Eighth European Conference on Machine Learning*, 383-386. Springer-Verlag.

Michie, D., Spiegelhalter, D.J. & Taylor, C.C. (1994). *Machine learning, neural and statistical classification*. Englewood Cliffs: Prentice Hall.

Pazzani, M., C. Merz, P. Murphy, K. Ali, T. Hume & C. Brunk (1994). Reducing Misclassification Costs. *Proceedings of the Eleventh International Conference on Machine Learning*, 217-225. Morgan Kaufmann.

Quinlan, J.R. (1993). *C4.5: Program for machine learning*. San Mateo: Morgan Kaufmann.

Quinlan, J.R. (1996). Bagging, boosting, and C4.5. *Proceedings of the 13th National Conference on Artificial Intelligence*, 725-730. AAAI Press. Menlo Park: AAAI Press.

Quinlan, J.R. (1997). C5. [http://rulequest.com].

Schapire, R.E., Y. Freund, P. Bartlett, & W.S. Lee (1997). Boosting the margin: A new explanation for the effectiveness of voting methods. *Proceedings of the Fourteenth International Conference on Machine Learning*, 322-330. Morgan Kaufmann.

Schapire, R.E. & Singer, S. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning. 37(3)*, 297-336. Boston: Kluwer Academic Publishers.

Ting, K.M. (in press). An Instance-Weighting Method to Induce Cost-Sensitive Trees. *IEEE Transaction on Knowledge and Data Engineering*.

Ting, K.M. and Zheng, Z. (1998). Boosting cost-sensitive trees. *Proceedings of the First International Conference on Discovery Science, LNAI-1532*, 244-255. Berlin: Springer-Verlag.

Turney, P.D. (1995). Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm. *Journal of Artificial Intelligence Research, 2*, 369-409.

Webb, G.I. (1996). Cost-Sensitive Specialization. *Proceedings of the 1996 Pacific Rim International Conference on Artificial Intelligence*, 23-34. Springer-Verlag.

ENDNOTES

[1] C4.5 uses fractional weights for the treatment of missing values. See Quinlan (1993) for details.

[2] In general, the costs of correct classifications can be non-zero. Minimizing the costs of correct classifications is a different issue outside the scope of this chapter.

[3] Note that an arbitrary cost matrix can be normalized to become a cost matrix satisfying this unity condition.

Chapter IV

Heuristic Search-Based Stacking of Classifiers

Agapito Ledezma, Ricardo Aler and Daniel Borrajo
Universidad Carlos III de Madrid

Currently, the combination of several classifiers is one of the most active fields within inductive learning. Examples of such techniques are boosting, bagging and stacking. From these three techniques, stacking is perhaps the least used one. One of the main reasons for this relates to the difficulty to define and parameterize its components: selecting which combination of base classifiers to use and which classifiers to use as the meta-classifier. The approach we present in this chapter poses this problem as an optimization task and then uses optimization techniques based on heuristic search to solve it. In particular, we apply genetic algorithms to automatically obtain the ideal combination of learning methods for the stacking system.

INTRODUCTION

One of the most active and promising fields in inductive machine learning is the ensemble of classifiers approach. An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples (Dietterich, 1997). The purpose of combining classifiers consists of improving the accuracy of a single classifier. Experimental results show that this is usually achieved.

There are several ways to construct such ensembles, but currently the most frequently used ones are bagging (Breiman, 1996), boosting (Freund & Schapire, 1995) and, less widely used, stacking (Wolpert, 1992). Bagging constructs a set of classifiers by subsampling the training examples to generate different hypotheses.

After the different hypotheses are generated, they are combined by a voting mechanism. Boosting also uses the voting system to combine the classifiers. But, instead of subsampling the training examples, it generates the hypotheses sequentially. In each repetition, a new classifier is generated whose focus are those instances that were handled incorrectly by the previous classifier. This is achieved by giving a weight to each instance in the training examples and adjusting these weights according to their importance after every iteration. Both, bagging and boosting use classifiers generated by the same base-learning algorithm and obtained from the same data. Finally, stacking can combine classifiers obtained from different learning algorithms using a high level classifier–the metaclassifier—to combine the lower level models. This is based on the fact that different classifiers are obtained from the same data and different learning algorithms use different biases to search the hypothesis space. This approach expects that the metaclassifier will be able to learn how to decide between the predictions provided by the base classifiers to improve their accuracy, much in the same way as a committee of experts.

One problem associated with stacked generalization is identifying which learning algorithm should be used to obtain the metaclassifier, and which ones should be the base classifiers. The approach we present in this chapter poses this problem as an optimization task, and then uses optimization techniques based on heuristic search to solve it. In particular, we apply genetic algorithms (Holland, 1975) to automatically obtain the ideal combination of learning methods for the stacking system.

BACKGROUND

The purpose of this section is to give enough background to understand the rest of the paper. Here, we will explain concepts related to ensembles of classifiers, bagging, boosting, stacking, and genetic algorithms.

Ensemble of Classifiers

The combination of multiple classifiers to improve the accuracy of a single classifier has had good results over several datasets that appear in recent papers about ensembles of classifiers (Bauer & Kohavi, 1999; Breiman, 1996; Freund & Schapire, 1996; Quinlan, 1996). According to Dietterich (1997), an ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples. There are many ways to construct an ensemble of classifiers. Bauer and Kohavi (1999) have made a comparison of algorithms based on voting systems. Dietterich (2000) carried out a survey of the main methods to construct an ensemble of classifiers. One way to construct an ensemble of classifiers is based on subsampling the training set to generate a different set of hypotheses and then combine them. This is called bagging (Breiman, 1996). The second way is to create classifiers sequentially, giving more importance to examples that were misclassified in the previous classifier. The latter is called boosting (Schapire, 1990). Both bagging and boosting use a single learning algorithm to generate all the classifiers, whereas stacking combines classifiers from different learning algorithms. There are other methods to combine a set of classifiers (Dietterich, 2000), but for experimental purposes in this study we consider here the most well known methods, bagging, boosting and stacking.

Bagging

Breiman (1996) introduced the *B*ootstrap *agg*regat*ing* algorithm (bagging) that generates different classifiers from different bootstrap samples and combines decisions from the different classifiers into a single prediction by voting (the class that gets more votes from the classifiers wins). Figure 1 shows the bagging algorithm.

Each bootstrap sample is generated by sampling uniformly (with replacement) the m-sized training set until a new set with m instances is obtained. Obviously, some of the instances of the training set will be cloned in the bootstrap sample, and some will be missing. Then, a classifier C_i is built from each of the bootstrap samples B_1, B_2, … B_T. Each classifier corresponding to each bootstrap sample will focus on some instances of the training set, and therefore the resulting classifiers will usually differ from each other. The final classifier C* is the ensemble of the C_i classifiers combined by means of the uniform voting system (all classifiers have equal weight).

Since an instance has probability $1-(1-1/m)^m$ of being in the bootstrap sample, each bootstrap replicate has, on average, 63.2% of unique training instances (1 - 1/e = 63.2%). For this reason the generated classifiers will be different if the base learning algorithm is unstable (e.g., decision tree or neural networks). If the generated classifiers are good and do not correlate, the performance will be improved. This will occur if they make few mistakes and different classifiers do not misclassify the same instances frequently.

Figure 1: The bagging algorithm

```
The Bagging Algorithm
Input:
Training set S
Base Learning Algorithm B
Number of bootstrap samples T

Procedure:
For i = 1 to T {
                S' = bootstrap sample from S  (S' is a sample with replacement from S)
                $C_i$ = B (S')   (create a new classifier from S')
}
C*(x) = $\arg\max_{y \in Y} \sum_{i:C_i(x)=y} 1$ (the most often predicted label y)
Output
Classifier C*
```

Boosting

Another method to construct an ensemble of classifiers is known as boosting (Schapire, 1990), which is used to boost the performance of a weak learner. A weak learner is a simple classifier whose error is less than 50% on training instances. There are many variants of the boosting idea, but in this study we will describe the widely used AdaBoost algorithm (*ada*ptive *boost*ing) that was introduced by Freund & Schapire (1995). Sometimes the AdaBoost algorithm is also known as AdaBoostM1.

Figure 2: AdaBoostM1 Algorithm.

Similarly to bagging, boosting manipulates the training set to generate a set of classifiers of the same type (e.g., decision trees). But, while in bagging, classifiers are generated independently from each other, in boosting they are generated sequentially, in such a way that each one complements the previous one. First, a classifier is generated using the original training set. Then, those instances misclassified by this classifier are given a larger weight (therefore, misclassifying this instance makes the error larger). Next, a new classifier is obtained by using the previously weighted training set. The training error is calculated and a weight is assigned to the classifier in accordance with its performance on the training set. Therefore, the new classifier should do better on the misclassified instances than the previous one, hence complementing it. This process is repeated several times. Finally, to combine the set of classifiers in the final classifier, AdaBoost (like bagging) uses the voting method. But unlike bagging that uses equal weights for all classifiers, in boosting the classifiers with lower training error have a higher weight in the final decision.

More formally, let S be the training set and T the number of trials, the classifiers C_1, C_2,.... C_T are built in sequence from the weighted training samples (S_1, S_2,...,S_T). The final classifier is the combination of the set of classifiers in which the weight of each classifier is calculated by its accuracy on its training sample. The AdaBoost algorithm is displayed in Figure 2.

An important characteristic of the AdaBoost algorithm is that it can combine some *weak* learners to produce a *strong* learner, so long as the error on the weighted training set is smaller than 0.5. If the error is greater than 0.5, the algorithm finishes the execution. Were it not so, the final prediction of the ensemble would be random.

Stacking

Stacking is the abbreviation used to refer to stacked generalization (Wolpert, 1992). Unlike bagging and boosting, it uses different learning algorithms to generate the ensemble of classifiers. The main idea of stacking is to combine classifiers from different learners such as decision trees, instance-based learners, etc. Since each one uses different knowledge representation and different learning biases, the hypothesis space will be explored differently, and different classifiers will be obtained. Thus, it is expected that they will not be correlated.

Once the classifiers have been generated, they must be combined. Unlike bagging and boosting, stacking does not use a voting system because, for example, if the majority of the classifiers make bad predictions, this will lead to a final bad classification. To solve this problem, stacking uses the concept of *meta learner*. The meta learner (or level-1 model) tries to learn, using a learning algorithm, how the decisions of the base classifiers (or level-0 models) should be combined.

Given a data set S, stacking first generates a subset of training sets $S_{1,\ldots,}S_T$ and then follows something similar to a cross-validation process: it leaves one of the subsets out (e.g., S_j) to use later. The remaining instances $S^{(-j)}= S - S_j$ are used to generate the level-0 classifiers by applying *K* different learning algorithms, *k* = 1,

…, K, to obtain K classifiers. After the level-0 models have been generated, the S_j set is used to train the meta learner (level-1 classifier). Level-1 training data is built from the predictions of the level-0 models over the instances in S_j , which were left out for this purpose. Level-1 data has K attributes, whose values are the predictions of each one of the K level-0 classifiers for every instance in S_j . Therefore, a level-1 training example is made of K attributes (the K predictions) and the target class, which is the right class for every particular instance in S_j. Once the level-1 data has been built from all instances in S_j, any learning algorithm can be used to generate the level-1 model. To complete the process, the level-0 models are regenerated from the whole data set S (this way, it is expected that classifiers will be slightly more accurate). To classify a new instance, the level-0 models produce a vector of predictions that is the input to the level-1 model, which in turn predicts the class.

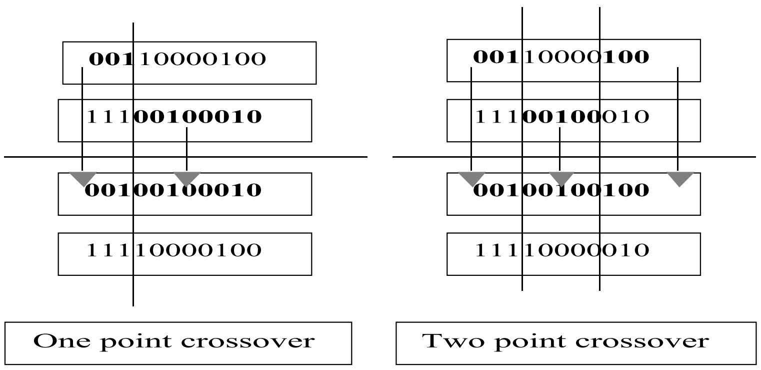
There are many ways to apply the general idea of stacked generalization. Ting and Witten (1997) use probability outputs from level-0 models instead of a simple class prediction as inputs to the level-1 model. LeBlanc and Tibshirani (1993) analyze the stacked generalization with some regularization (nonnegative constraint) to improve the prediction performance on one artificial dataset. Other works on stacked generalization have developed a different focus (Chan & Stolfo, 1995; Fan, Chan & Stolfo, 1996).

Genetic Algorithms for Optimization Problems

Genetic algorithms (GAs) are search procedures loosely connected to the theory of evolution by means of artificial selection (Holland, 1975). In classical search terms, GAs can be viewed as a kind of beam search procedure. Its main three components are:

- The beam (or population). It contains the set of points (candidate solutions or individuals) in the search space that the algorithm is currently exploring. All points are usually represented by means of bit strings. This domain independent representation of candidate solutions makes GAs very flexible.
- The search operators. They transform current candidate solutions into new candidate solutions. Their main characteristic is that, as they operate on bit strings, they are also domain independent. That is, they can be used to search in any domain. GAs operators are also based in biological analogies. The three most frequently used operators are:
  - Reproduction: it just copies the candidate solution without modification.
  - Crossover: it takes two candidate solutions, mixes them and generates two new candidate solutions. There are many variations of this operator (mainly, single point, two point and uniform). See Figure 3.
  - Mutation: it flips a single bit of a candidate solution (it mutates from 0 to 1, or from 1 to 0). The bit is selected randomly from the bits in the individual.
- The heuristic function (or fitness function). This function measures the worth of a candidate solution. The goal of a GA is to find candidate solutions that maximize this function.

Figure 3. One and two point crossover operations



GA's start from a population made up of randomly generated bit strings. Then, genetic operators are applied to worthy candidate solutions (according to the heuristic function) until a new population is built (the new generation). A GA continues producing new generations until a candidate solution is found that is considered to be good enough, or when the algorithm seems to be unable to improve candidate solutions (or until the number of generations reaches a predefined limit). More exactly, a GA follows the algorithm below:

```
1. Randomly generate initial population G(0)
2. Evaluate candidate solutions in G(0) with the heuristic function
3. Repeat until a solution is found or population converges
   3.1. Apply selection-reproduction: G(i) -> $G_a(0)$
   3.2. Apply crossover: $G_a(i)$ -> $G_b(i)$
   3.3. Apply mutation: $G_b(i)$ -> $G_c(i)$
   3.4. Obtain a new generation G(i+1) = $G_c(i)$
   3.5. Evaluate the new generation G(i+1)
   3.6. Let i=i+1
```

The production of a new generation G(i+1) from G(i) (Steps 3.1, 3.2, and 3.3) is as follows. First, a new population $G_a(i)$ is generated by means of selection. In order to fill up a population of n individuals for $G_a(0)$, candidate solutions are stochastically selected with replacement from G(i) n times. The probability for selecting a candidate solution is the ratio between its fitness and the total fitness of the population. This means that there will be several copies of very good candidate solutions in $G_a(0)$, whereas there might be none of those whose heuristic evaluation is poor. However, due to stochasticity, even bad candidate solutions have a chance of being present in $G_a(0)$. This method is called 'selection proportional to fitness', but there are several more, like tournament and ranking (Goldberg,1989).

$G_b(i)$ is produced by applying crossover to a fixed percentage p_c of randomly selected candidate solutions in $G_a(i)$. As each crossover event takes two parents and produces two offspring, crossover happens $p_c/2$ times. In a similar manner, $G_c(i)$ is obtained from $G_b(0)$ by applying mutation to a percentage p_m of candidate solutions.

GA-STACKING APPROACH

In this section, the approach we have taken to build stacking systems will be explained. Given that stacked generalization is composed by a set of classifiers from a different set of learning algorithms, the question is: What algorithm should be used to generate the level-0 and the level-1 models? In principle, any algorithm can be used to generate them. For instance, Ting and Witten (1997) showed that a linear model is useful to generate the level-1 model using probabilistic outputs from level-0 models. Also, our classifiers at level-0 are heterogeneous, and any algorithm can be used to build the metaclassifier. In the present study, to determine the optimum distribution of learning algorithms for the level-0 and level-1 models, we have used a GA.

GA-Stacking

Two sets of experiments were carried out to determine whether stacked generalization combinations of classifiers can be successfully found by genetic algorithms. We also wanted to know whether GA stacking can obtain improvements over the most popular ensemble techniques (bagging and boosting) as well as any of the stand-alone learning algorithms.

As indicated previously, the application of genetic algorithms to optimization problems requires one to define:

- the representation of the individuals (candidate solutions)
- the fitness function that is used to evaluate the individuals
- the selection-scheme (e.g., selection proportional to fitness)
- the genetic operators that will be used to evolve the individuals
- the parameters (e.g., size of population, generations, probability of crossover, etc.)

The representations of the possible solutions (individuals) used in the two sets of performed experiments are chromosomes with five genes (see Figure 4). Each gene represents the presence of a learning algorithm. Since we want to use seven possible learning algorithms, each gene has three binary digits to represent them. The first four genes of the chromosome represent the four learning algorithms to build the level-0 classifiers and the last gene represents the algorithm to build the

Figure 4: Individual description

$$\underbrace{\mathbf{010}111\mathbf{000}110}_{\text{level-0 models}}\overbrace{\mathbf{001}}^{\text{level-1 model}}$$

metaclassifier.

To evaluate every individual, we used the classification accuracy of the stacking system as the fitness function.

EXPERIMENTAL RESULTS

In this section, we present the empirical results for our GA-stacking system. GA-Stacking has been evaluated in several domains of the UCI database (Blake & Merz, 1998). We have performed two sets of experiments. In the first one, we show some preliminary results, which are quite good but suffer some overfitting. In the second set of experiments, steps were taken to avoid the said overfitting. Both sets of experiments differ only in the way GA individuals are tested (i.e., the fitness function). In this work, we have used the algorithms implemented in Weka (Witten & Frank, 2000). This includes all the learning algorithms used and a implementation of bagging, boosting and stacking.

Basically, the implementation of GA-stacking combines two parts: a part coming from Weka that includes all the base learning algorithms and another part, which was integrated into Weka, that implements a GA. The parameters used for the GA in the experiments are shown in Table 1. The elite rate is the proportion of the population carried forward unchanged from each generation. Cull rate refers to the proportion of the population that is deemed unfit for reproduction. Finally, the mutation rate is the proportion of the population that can suffer change in the configuration at random.

For the experimental test of our approach we have used seven data sets from the repository of machine learning databases at UCI. Table 2 shows the data sets' characteristics.

The candidate learning algorithms for GA stacking we used were:

- C4.5 (Quinlan, 1996). It generates decision trees—(C4.5)

Table 1: Genetic algorithm's parameters

Parameters	Values
Population size	10
Generations	10
Elite rate	0.1
Cull rate	0.4
Mutation rate	0.067

Table 2: Datasets description

Dataset	Attributes	Attributes Type	Instances	Classes
Heart disease	13	Numeric-nominal	303	2
Sonar classification	60	Numeric	208	2
Musk	166	Numeric	476	2
Ionosphere	34	Numeric	351	2
Dermatology	34	Numeric-nominal	366	6
DNA splice	60	Nominal	3190	3
Satellite images	36	Numeric	6435	6

- PART (Frank & Witten, 1998). It forms a decision list from pruned partial decision trees generated using the C4.5 heuristic—(PART)
- A probabilistic Naive Bayesian classifier (John & Langley, 1995)—(NB)
- IB1. This is Aha's instance based learning algorithm (Aha & Kibler, 1991)—(IB1)
- Decision Table. It is a simple decision table majority classifier (Kohavi, 1995) —(DT)
- Decision Stump. It generates single-level decision trees—(DS)

Preliminary Experiments

Experiments in this subsection are a first evaluation of GA-stacking. They use the GA-stacking scheme described in previous sections. Here, we will describe in more detail how individuals are tested (the fitness function), and how the whole system is tested. Each data set was split randomly in two sets. One of them was used as the training set for the GA fitness function. It contained about 85% of all the instances. Individual fitness was measured as the accuracy of the stacking system codified in it. The rest was used as a testing set to validate the stacked hypothesis obtained. In these experiments we used only two of the data sets shown in Table 2 (dermatology and ionosphere).

Table 3 shows the results for this preliminary approach. Training and test accuracy columns display how many instances in the training and testing sets were correctly predicted for each of the systems. The first part of the table contains the stand-alone algorithms. The second part shows results corresponding to traditional bagging and boosting approaches (with C4.5 as the base classifier). Finally, results for GA-Stacking are shown. The best GA-stacking individual in the last generation

Table 3: Preliminary results

Algorithm	Ionosphere		Dermatology	
	Training Accuracy	Test Accuracy	Training Accuracy	Test Accuracy
C4.5	98.42	82.86	96.67	92.42
Naive Bayes	85.13	82.86	98.67	96.97
PART	98.73	88.57	96.67	93.94
IBk (one neighbor)	100.00	82.86	100.00	92.42
IB1	100.00	82.86	100.00	92.42
Decision Stump	84.18	80.00	51.33	45.45
Decision Table	94.94	88.57	96.67	87.88
Ensembles				
Bagging with C4.5	97.78	85.71	97.00	93.94
Boosting with C4.5	100.00	91.43	100.00	96.97
GA-Stacking(last generation solutions)	100.00	85.71	100.00	95.45
GA-Stacking (solutions of previous generations)	97.78	<u>94.29</u>	98.67	<u>98.48</u>

has a 100% in the training set in both domains, but drops to 85.71% and 95.45% in testing. On the other hand, some individuals from previous generations manage to get a 94.29% and 98.48% testing accuracies. This means that GA-stacking overfits the data as generations pass on. Despite this overfitting, GA-stacking found individuals which are better than most of the base classifiers. Only in the ionosphere domain, the decision table is better in testing than the overfitted individual (88.57 vs. 85.71). Also, performance of GA-Stacking is very close to bagging and boosting (both using C4.5 to generate base classifiers). Non-overfitted individuals (those obtained in previous generations) show an accuracy better than any of the other ensemble systems. In addition, stacking systems only use four level-0 classifiers whereas bagging and boosting use at least 10 base classifiers. This shows that the approach is solid. The following section shows how we overcame the problem of overfitting.

Main Experiments (Overfitting avoidance)

In the preliminary experiments, individuals overfit because they were evaluated with the same training instances that were used to build the stacking associated with the individual. Obviously, the individual will do very well with the instances it has been trained with. Therefore, in order to avoid the overfitting, the fitness value was calculated with a separate validation set (different from the testing set used to evaluate GA-stacking itself). To do so, the set of training instances was partitioned randomly into two parts. Eighty percent of training instances were used to build the stacking system associated with each individual, and the remaining 20%–the validation set- was used to give a non-biased estimation of the individual accuracy. The latter was used as the fitness of the individual. It is important to highlight that now fewer instances are used to build the stacking systems, as some of them are used to prevent overfitting.

For testing all the systems we used the testing data set, just as we did in the preliminary experiments. However, now a five-fold cross-validation was used. Therefore, results shown are the average of the five cross-validation iterations. In the satellite images domain no cross-validation was carried out because the sets for training and testing are available separately and the data set donor indicated that it is better not to use cross-validation.

Results are divided in two tables. Table 4 shows the testing accuracy of all the base algorithms over the data sets.

Table 5 displays testing results for the three ensemble methods, including GA-stacking. The best results for each domain are highlighted. Only in the heart domain, one of the base classifiers obtained greater accuracy than any ensemble method. The configuration of the stacked generalization found by the genetic algorithms obtained a greater accuracy in four of the seven domains used in the experiment in contrast to the other ensemble methods. In the domains where stacking has lesser accuracy than bagging or boosting the difference is very small except in the musk domain, where the difference is +5% in favor of boosting.

*Table 4: Average accuracy rate of independent classifiers in the second experiment *without cross-validation.*

Dataset	C4.5	PART	NB	IB1	DT	DS
Heart	74.00	82.00	<u>83.00</u>	78.00	76.33	70.67
Sonar	72.38	73.81	67.62	79.52	71.90	73.93
Musk	83.33	85.21	74.38	86.46	82.08	71.46
Ionosphere	90.14	89.30	82.82	87.61	89.30	82.82
Dermatology	94.33	94.59	97.03	94.59	87.83	50.40
DNA splice	94.12	92.11	95.63	75.62	92.49	62.42
Satellite images*	85.20	84.10	79.60	89.00	80.70	41.75

Table 5: Average accuracy rate of ensemble methods in the second experiment

Dataset	Bagging	Boosting	GA-Stacking
Heart	76.33	79.67	<u>80.67</u>
Sonar	80.00	79.05	<u>80.47</u>
Musk	87.29	<u>88.96</u>	83.96
Ionosphere	<u>92.11</u>	91.83	90.42
Dermatology	94.59	97.03	<u>98.11</u>
DNA splice	94.56	94.43	<u>95.72</u>
Satellite images	88.80	<u>89.10</u>	88.60

CONCLUSIONS

In this chapter, we have presented a new way of building heterogeneous stacking systems. To do so, a search on the space of stacking systems is carried out by means of a genetic algorithm. Empirical results show that GA-stacking is competitive with more sophisticated ensemble systems such as bagging and boosting. Also, our approach is able to obtain high accuracy classifiers which are very small (they contain only five classifiers) in comparison with other ensemble approaches.

Even though the GA-Stacking approach has already produced very good results, we believe that future research will show its worth even more clearly:

- Currently, GA-stacking searches the space of stacked configurations. However, the behavior of some learning algorithms is controlled by parameters. For instance, the rules produced by C4.5 can be very different depending on the degree of pruning carried out. Our GA-stacking approach would only have to include a binary coding of the desired parameters in the chromosome.
- Search depends on the representation of candidate hypothesis. Therefore, it would be interesting to use other representations for our chromosomes. For instance, the chromosome could be divided into two parts, one for the level 0 learners and another for the level 1 metaclassifier. A bit in the first part would

indicate whether a particular classifier is included in level 0 or not. The metaclassifier would be coded in the same way we have done in this paper (i.e., if there are 8 possible metaclassifiers, 3 bits would be used).
- Different classifiers could be better suited for certain regions of the instance space. Therefore, instead of having a metaclassifer that determines what output is appropriate according to the level 0 outputs (as currently), it would be interesting to have a metaclassifier that knows what level 0 classifier to trust according to some features of the instance to be classified. In that case, the inputs to the metaclassifier would be the attributes of the instance and the output the right level 0 classifier to use.

REFERENCES

Aha, D. & Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning, 6,* 37-66.

Bauer, E. & Kohavi, R.(1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36 (1/2)*, 105-139.

Blake, C.L. & Merz, C.J. (1998). *UCI Repository of machine learning databases*. Retrieved November 1, 2000, from http://www.ics.uci.edu/~mlearn/MLRepository.html

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24(2)*, 123-140.

Chan, P. & Stolfo, S. (1995). A comparative evaluation of voting and meta-learning on partitioned data. In *Proceedings of Twelfth International Conference on Machine Learning* (90-98). Morgan Kaufmann.

Dietterich, T. (1997). Machine learning research: four current directions. *AI Magazine 18(4)*, 97-136.

Dietterich, T. (2000). Ensemble methods in machine learning. In Proceedings of *First International Workshop on Multiple Classifier Systems* (pp. 1-15). Springer-Verlag.

Fan, D., Chan, P., & Stolfo, S. (1996). A comparative evaluation of combiner and stacked generalization. In *Proceedings of AAAI-96 workshop on Integrating Multiple Learning Models*, 40-46.

Frank, E. & Witten, I. (1998). Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning* (144-151). Morgan Kaufmann.

Freund, Y. & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, Springer-Verlag, 23-37.

Freund, Y. & Schapire, R. E. (1996). Experiment with a new boosting algorithm. In L. Saitta, ed., *Proceedings Thirteenth International Conference on Machine Learning,* Morgan Kaufmann, 148-156.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

John, G.H. & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, 338-345.

Kohavi R. (1995). The power of decision tables. In *Proceedings of the Eighth European*

Conference on Machine Learning.

LeBlanc, M. & Tibshirani, R. (1993). Combining estimates in regression and classification. *Technical Report 9318*. Department of Statistic, University of Toronto.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of Thirteenth National Conference on Artificial Intelligence*, AAAI Press and MIT Press, 725-730.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5(2)*, 197-227.

Ting, K. & Witten, I. (1997). Stacked generalization: When does it work. In *Proceedings International Joint Conference on Artificial Intelligence*.

Whitley, L.D.(1991). Fundamental principles of deception in genetic search. In Gregor y J.E (Ed.) *Foundations of genetic algorithms*, San Mateo, CA: Morgan Kaufmann, 221-241.

Witten, I. & Frank, E. (2000). *Data mining: Practical machine learning tools and techniques with Java implementation*. San Francisco, CA: Morgan Kaufmann.

Wolpert, D. (1992). Stacked generalization. *Neural Networks 5*, 241-259.

Chapter V

Designing Component-Based Heuristic Search Engines for Knowledge Discovery

Craig M. Howard
Lanner Group Limited, UK
University of East Anglia, UK

The overall size of software packages has grown considerably over recent years. Modular programming, object-oriented design and the use of static and dynamic libraries have all contributed towards the reusability and maintainability of these packages. One of the latest methodologies that aims to further improve software design is the use of component-based services. The Component Object Model (COM) is a specification that provides a standard for writing software components that are easily interoperable. The most common platform for component libraries is on Microsoft Windows, where COM objects are an integral part of the operating system and used extensively in most major applications.
This chapter examines the use of COM in the design of search engines for knowledge discovery and data mining using modern heuristic techniques and how adopting this approach benefits the design of a commercial toolkit. The chapter describes how search engines have been implemented as COM objects and how representation and problem components have been created to solve rule induction problems in data mining.

BACKGROUND

In traditional software projects the application was a single executable, built by a specific compiler and linker for a particular operating system and platform. As projects got larger, code was separated out into modules, or libraries, to improve the management of the source code but ultimately was still built into a single program file; this is referred to as static linking of libraries. An improvement on static libraries was dynamic libraries, where only the prototypes of available library functions are supplied to the core program at compile time. When the program is executed, the operating system has to locate the library using path rules to scan the file system and dynamically link the two together. As well as further improving the management of larger projects, dynamically linked libraries have the added advantage of being reusable by multiple programs. For example, when two programs use the same graphics library to build the user interface, the programs are built using function prototypes and require only a single copy of the dynamic library to be located on the system. By removing common functionality from the executables, the overall size of the executables is reduced and only one copy of the library is required to support multiple programs. In addition, isolating the shared functions is not only good software engineering practice but also reduces problems caused by numerous copies of the same code in different locations, making projects and version control easier to manage. Even with these improvements there are still a number of drawbacks. Firstly, if the library needs to be updated or changed (maybe for just one program), existing programs will need to be rebuilt to use this new version even if the prototypes have not changed and the library is backward compatible with earlier versions. Secondly, sharing libraries between different platforms and even compilers is not straightforward, for example, a dynamic library created for Digital Unix would not work under Solaris, and getting a Windows dynamic linked library (DLL) written in Microsoft Visual Basic to work with Borland C++ is not always an easy task.

The problem being addressed in this chapter is the design of component-based software with a particular application to rule generation for data mining. The problem of generating rules describing a class of records in a database can be formulated as an optimisation problem (Rayward-Smith, Debuse, & de la Iglesia, 1996). Heuristic search engines can be used to generate and evaluate rules whilst taking into account a number of constraints such as limiting the complexity of the rule and biasing either accuracy or coverage. A feasible solution in the data mining problem domain is represented by any valid rule that can be evaluated against the database. A number of data mining toolkits using techniques such as genetic algorithms and simulated annealing are listed on the KDNugetts Web site (KDNuggets, 2001), in particular the Datalamp toolkit (Howard, 1999a), which is based on much of the work described in this chapter.

INTRODUCTION TO COM

The Component Object Model (COM) is a specification that provides a standard for writing software components (Microsoft, 1999a). The standard is based on defining clear interfaces between components to promote interoperability and reusability across multiple languages and platforms using object interaction at the binary level (Microsoft, 1999b). The major design goals of COM are:

- **Language Independence.** COM components can be written in most modern programming languages including C++, Visual Basic, Java, etc., and the requirement being that the language can produce compiled code that is capable of calling functions via function pointers. In short, compiled objects must be compatible with the v-tables (virtual tables) found in C++. An additional advantage of this approach is that many scripting languages used in web pages, e.g., VBScript, are also compatible using dispatch interfaces making it possible for Web sites to create and call COM objects. Web pages can therefore create COM objects that, for example, talk to backend database servers to extract data required to build the page.
- **Vendor Independence.** A COM object exports the interfaces it implements in a standardised format known as the Interface Definition Language (IDL) using type libraries; as such there are no details of the underlying implementation. If two objects export identical interfaces, one can transparently replace the other.
- **Reduced Version Problems.** Interfaces should be immutable. Once an interface has been published and used by anyone outside the development environment, it must not change even to the extent that it must retain the original order of the functions (this maintains a consistent and compatible v-table). If an interface is modified in some way then it should be republished using a version number or different name.

COM is sometimes described as being middleware (Sherlock & Cronin, 2000) as it is most often used to develop components that sit between the presentation tier and data tier of an n-tier application. The reusable pieces of software implement services that make up the main functionality of an application; the ability to easily replace one component with another simplifies the process of fixing or modifying services without having to touch the user interface or backend databases. With the advent of the Internet and cheaper office PCs, it is common for hundreds of clients (or more) to be accessing data and carrying out tasks through various front-end applications. By having the functionality of a system placed at the middle (business) layer, the internal operations are separated from both the users and the data, thus improving the maintainability of the overall application.

Development of software components has increased rapidly in the past few years with most vendors of tools and applications having made some commitment to the component-based paradigm (Allen & Frost, 1998). Since much of the development of COM took place at Microsoft, and given wide support by the industry, it is easy to see why the majority of COM-based software is found on the

Windows platform. However, support for COM is steadily growing on other platforms as the popularity of this design methodology continues to grow; one such platform is the Unix operating system. Support for component design has been implemented in COM for OpenVMS and TruUnix as well as other commercial versions which support Linux, Digital and HP Unix. Although COM is the most widely used approach to designing middleware components, it is not the only method available. One alternative comes from the Object Management Group who produced the Common Object Request Broker Architecture (CORBA) (OMG, 2000). In a similar nature to COM, CORBA invokes applications on remote machines but has a stronger emphasis on design using the object-oriented software model and communication between objects involving marshalling. More recently, Enterprise Java Beans (EJB) has become available with the Java language (Matena & Hapner, 1999). The local execution environment for Java applications on a particular machine is provided by a platform specific virtual machine (JVM); Enterprise Java extends applications to work and communicate across machine boundaries and the Internet. EJB is a component model based on the Enterprise Java API that promotes reusable, network-ready components.

Building Component-Based Software

The communication channels between one object and another are defined by a set of interfaces, which are supported by a component. A COM interface is a collection of related function prototypes (pure virtual in C++). For a component to support an interface it must implement that interface; that is, it must provide code to implement each function declared in the interface. There are two main interface types, custom and dispatch, which are described in more detail later in this section. Any component must meet two architecture requirements: first, the ability to link dynamically to other applications and components; secondly, the ability to hide (encapsulate) the implementation details. The ability to dynamically link at the binary level means that the process of updating software is greatly simplified. It is no longer necessary to rebuild and replace the entire application, but merely the component that has been updated. Software components can therefore be transparently updated such that no change is visible to the user (subject to there being no need to change the way in which it communicates).

Four main tasks must be completed when building components. The low-level COM API and the higher-level approaches that follow have to complete each of these stages. However, with the API each section is written from scratch with no help from macros or template libraries provided by a framework. The four tasks are:

- declare the interfaces;
- build the required COM objects;
- build the class factory to instantiate the COM objects;
- create the container module (DLL or EXE server).

Interfaces. Defining sensible interfaces is the key to developing a usable, and reusable, component. The interfaces should reflect the outcome of sufficient analysis and design having been carried out for a project. The interface defines how clients will access the component; what data can be transferred; what functions can be called; and how the functions are grouped together. Once published, that is, deployed outside the development environment, an interface should not be modified as any changes will render existing clients unusable until they have been rebuilt. An interface must be assigned an interface identifier (IID), which is a Globally Unique Identifier (GUID) represented by a 128-bit number, that can be used to access details of the interface in the system registry (Shepherd & Wingo, 1996). There is one mandatory interface that all components must implement: IUnknown. The methods exposed by IUnknown handle reference counting to manage the lifetime of an object and allow clients to query an object to determine whether a particular interface is supported.

COM Classes. To build the concrete COM object, a standard C++ class must be derived from the interfaces that provide the required functionality for the components. This type of class is often referred to as the COM class. Each of the pure virtual methods from the custom interfaces and the inherited IUnknown interface must be implemented inside the COM class. Apart from these exposed functions, the class behaves in the same way as any other C++ class and can use member variables and functions in the normal way.

Class Factories. Class factories are used to construct COM objects in a language independent manner. Inside the class factory, the creation process is dependent upon the implementation language; in C++, the new operator would be used

Table 1: Differences between in-process and out-of-process COM servers

In process server (DLL)	**Out of process server (EXE)**
Runs on local machine only	Runs on local or remote machine
Runs in address space of client application	Runs in its own address space
Will terminate if application terminates or crashes	Will continue running should application crash
Efficient function calls since inside same process	Slower function calls due to proxy-stub, RPC and marshalling over networks
Custom interfaces do not require extra standard marshalling code	Custom interfaces require standard marshalling to be provided using IDL
One instance of server resides inside one instance of client. Multiple clients require multiple servers.	Multiple clients can connect to, and share, a single out-of-process server.
Allows reuse through aggregation and containment	Allows reuse through containment only

to dynamically create an instance of the COM class. Once created, a pointer to the new object is returned to the client, who will be unaware of how the object was actually created. The call to the class factory is usually made indirectly via the COM API.

COM Servers. The final decision when building COM objects is what type of container will be used to house the components: in-process or out-of-process server. The advantages and disadvantages of these container types are listed in Table 1 although out-of-process (EXE) servers are mandatory for Distributed COM.

There are three main approaches to implementing COM objects in C++.

COM API. Developing software components using the COM API is the most verbose, and often most complex, of all the available approaches. However, it is also the most explicit and informative method that clearly shows what is happening at each stage of the objects' lifecycle; how clients interact with objects; how objects interact with COM and how COM interacts with the operating system. Unlike the methods that follow, programming the COM API, as with most APIs, means that the user has to do most of the work and there is no support from macros or other code-saving techniques found in application frameworks.

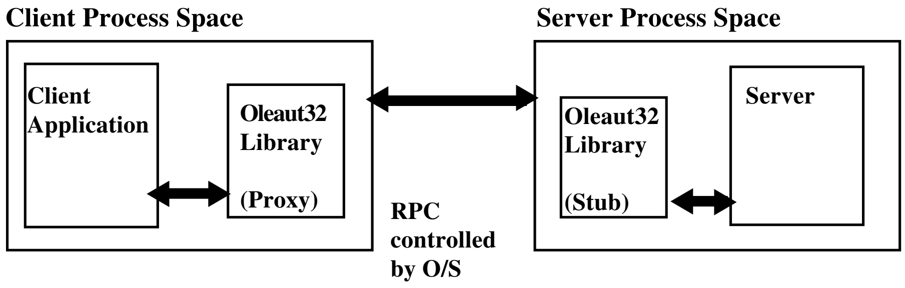
Microsoft Foundation Classes (MFC). The MFC framework provides classes and macros to wrap the COM API much the same as it wraps the Win32 API. In addition to adding a layer of abstraction, the MFC COM extensions simplify most of the repetitive and time-consuming tasks when defining interfaces and implementing COM classes, the most obvious example being writing repeated blocks of code to implement the IUnknown interface when is implemented by an MFC class.

Active Template Library (ATL). The ATL is the most recent collection of C++ classes that aid writing COM components. The ATL is a set of template-based classes designed specifically to create small, fast and extensible COM objects. As with the MFC extensions, ATL wraps the underlying COM API to provide a level of abstraction and simplified code for common procedures such as aggregation, dual interfaces (for OLE automation), connection points and enumerated types. Unlike MFC, the ATL does not depend on a large external library of routines that are also used for application and GUI design.

Distributed COM

Distributed COM (DCOM) is an infrastructure for software components to communicate over a network. DCOM components, commonly referred to as DCOM servers, must be implemented as out-of-process EXE servers as they will be running on remote machines and not loaded into the address space of a local client. One of the key benefits of DCOM is the transparency of the network layer as the client is unaware of the physical location of the server. Any data passing across a network has to undergo marshalling, a process where complex and customised data

Figure 1: Proxy/stub connection using the standard automation library



structures are broken down into simple types and sent between machines. The data transfer procedure uses a proxy/stub connection which is illustrated in Figure 1. The data transfer takes place between a proxy on the client and a stub on the server; for standard data types the OLE automation library handles the proxy and stubs. At both client and server, the components appear to be talk directly to one another; the intervention of the automation library is transparent to the user.

Marshalling data across process boundaries obviously incurs an overhead and the interface definition language (described in a later section) goes some way to improving the efficiency of this process. Nevertheless, there are still significant overheads in making the function call via remote procedure calls (RPC), as shown in an experiment using null method invocation (no parameters and no work carried out by the method) (Box, 1997). With both client and server on the same machine and in the same process (in-process DLLs), it was possible to invoke a null method over 28 million times in one second. The corresponding value for out-of-process servers on different machines was just under 1,000. Although this figure seems quite alarming and a considerable drawback of distributing components for data mining, examining what happens during the search for rules reveals that it is not as bad as it seems. Most of the time associated with solving data mining problems is spent evaluating solutions, and therefore the number of methods invoked would more likely be dependent upon the number of evaluations, which are expensive compared to the overheads of calling the method out-of-process.

COM Automation

Interfaces can be defined to be custom interfaces, dispatch interfaces, or both. Custom interfaces usually inherit directly from IUnknown and are accessed through v-tables, these interfaces are used in languages such as C++ as this is the most efficient approach of method invocation. The alternative, dispatch interfaces allow methods and properties to be accessed using their names and a dispatch ID; accessing methods and properties in this way is also known as (OLE) automation. Automation exposes the functionality of a component to applications and languages that do not support the use of v-tables or are not strongly typed. Scripting languages such as Javascript and VBScript are examples of such languages commonly used to access COM objects in web pages, i.e. ActiveX objects. Components can also be designed to have dual interfaces, that is, they support access through custom interfaces and automation. Accessing functions via their names introduces a significant performance hit; however, in some cases such as using scripting languages, it is the only method available. It is therefore important when designing the component to identify how the component is likely to be used, on what platforms, and through which clients. Designing for access by multiple clients in multiple languages (using a dual interface) will result in the most flexible component but require slightly longer development time and will be most efficient when accessed through the custom interface.

IDL and Type Libraries

The interface definition language, or IDL, is a way of describing the functionality provided by interfaces (Hludzinski, 1998). On the Windows platform, IDL source is compiled into a type library using the MIDL compiler; MIDL also generates proxy and stub code used to marshal parameters between process boundaries. The type library is a compiled version of the IDL code and contains meta-data describing the component server such as the logical definitions for the dispatch interfaces and v-table descriptions for custom interfaces. MIDL uses attributes to determine the data transfer direction of parameters during marshalling. By knowing that a parameter is not modified by the server function, the marshaller does not have to copy the parameter back to the client when the function has ended, thus reducing the amount of work involved.

Component Reuse

Recent programming languages and development frameworks all promote code and software reuse; COM is no exception, especially with its design goal to promote reuse at the binary level. Until recently, the most common form of reuse was code reuse through inheritance; however, the language independence nature of COM complicates this issue. There are two distinct types of items that can be reused and take part in inheritance relationships: interfaces (code reuse) and implementations (software, or binary reuse). Implementation inheritance is extremely important in the development of small reusable components. It is implemented using one of two approaches, aggregation and containment, which logically embed one component inside another.

DESIGNING COMPONENT-BASED HEURISTIC SEARCH ENGINES

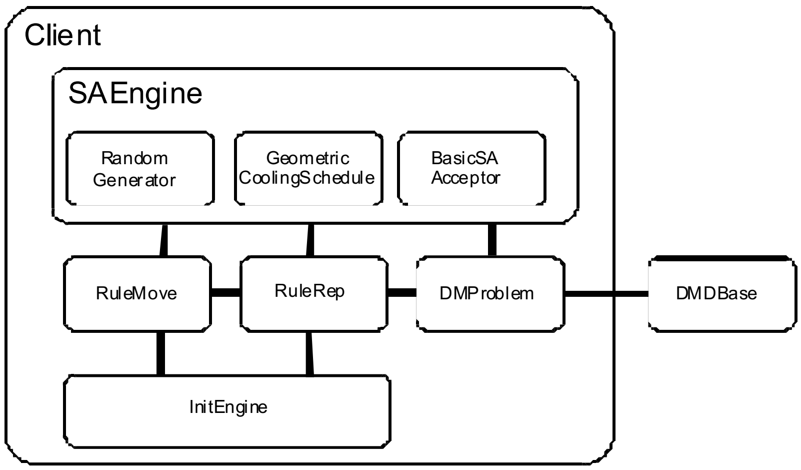
Application Design Using Static Libraries

The Templar framework (Jones, McKeown & Rayward-Smith, 2001) was used to develop an early version of the toolkit that used simulated annealing for data mining. Templar is an object-oriented framework for the implementation of search techniques and provides a collection of C++ classes. Three main components—the problem, representation and engine—make up a software solution suitable for solving an optimisation problem, in this case finding an optimal rule describing a class of records within a database. The framework provides full code for search engines, such as hill climbers and simulated annealing, which work on problem classes and representation classes. Standard representations, including bitstrings, integer arrays and permutations, are part of the basic framework as are a number of sample problem classes.

Rather than encode a rule as a standard representation type, a class was derived from the representation base class to model solutions more logically. The class consists of a data structure to store the conditions in a rule and a number of neighbourhood move operators used by the engine to generate new solutions. For the data mining problem, a class was derived from the base problem class provided by the framework. The primary requirement of this class is to implement an evaluation function that returns a fitness value for a given representation. Functions in the problem class use database and rule evaluation routines from a dynamic linked library to perform the evaluation. Once these classes have been written, all of the code is compiled and linked into the main client application (with the exception of the database DLL), as shown in Figure 2.

It is clear from the above illustration that any changes made to classes used by the SAEngine or the client will force recompilation and/or relinking of dependent modules. Therefore, changing the evaluation function in the problem class or modifying the cooling schedule would mean the client application, and SAEngine, would need to be rebuilt. Similarly, if the engine is to be used with a different problem, possibly inside a different client application, it is necessary to rebuild the program or maintain two versions of the source code. One possible solution, which is more elegant for cooling schedules than problem classes, is to embed multiple

Figure 2: Traditional application design using mostly static libraries



approaches inside the engine/client and make the selection at runtime by means of a large conditional statement inside the code.

Benefits of a Component-Based Approach

The decision to move to component-based design using COM was based on the following reasons that have been subdivided into programming and design factors; maintenance and future development; and other commercial benefits and influences.

Programming and Design Factors

Improved Project Management. Large projects are generally undertaken by development teams that work on many different parts of the application at the same time. Dividing the project up into smaller components makes it easier to assign jobs and handle the source code for the project. From a developer's point of view, a significant amount of time is spent waiting for machines to recompile and link projects after changes have been made. As the dependencies between components are defined by their interfaces, only changes to the interface structure would require rebuilding other components to take advantage of the changes.

Support for Multiple Languages. Components can be developed in and used by many languages supporting the COM standard. Clients can be written in languages including C++, Visual Basic and Java as well as scripting languages, such as VBScript and JavaScript, which in turn can be used in web pages through Active Server Pages (ASP). Another useful client environment is Visual Basic inside the Microsoft Excel and Access office applications which can be used to control the components.

Maintenance and Future Development

Improved Maintenance Paths. With the client application and search engine split up into a number of smaller components, the system is easier to maintain as fixes and updates will only apply to the component in question. The smaller size of the update increases the number of potential channels for deployment to users. Small updates can easily and relatively quickly be downloaded from support Web sites or even attached to emails. The cost of issuing updates over the Internet is significantly cheaper than distributing diskettes or CD-ROMs by mail. Updates can also occur on a more regular basis making it easier for users to keep up to date by having the latest version of the software.

Development of New Components. The ability to effortlessly add and remove components from engines makes developing new and improved techniques simpler to manage. COM's vendor independence goal allows one component to replace another, providing it supports the same interface. An example would be the development of a new cooling schedule for a simulated annealing engine. Providing the new component supports the same interfaces as the original, existing engines and client applications can use the new technique without making changes to the source code or needing rebuilding. Similarly, new problem classes can be developed to solve other optimisation problems without having to have or see source code from the engines. Developers require only the type libraries of the components they need to reference, hence hiding much of the complexity of the engines internal routines. This benefit makes it easier for software developers to design and experiment with new algorithms to solve data mining problems.

Commercial Benefits and Influences

Emerging Technology. The use of COM in software development has grown rapidly since its launch (when it was originally known as OLE). The majority of modern applications make heavy use of its technology and the ability to easily integrate third-party components. COM is used as a mechanism for communicating with current software libraries such as user interface objects, web browser controls and database access controls. Connecting to database servers was traditionally made through proprietary clients provided by the database vendor although the development of the Open Database Connectivity standard (ODBC) was the first step in having a universal method of talking to databases. The advent of COM has brought a new standard: ActiveX Data Objects (ADO). ADO is a set of components that a client uses to connect to and query any data source that can be represented in tabular format. ADO is the primary method of database connectivity used in Web site design.

Reusing components in software design is still on the increase, and emerging technologies such as XML (Bray, Paoli, & Sperberg-McQueen, 1998) and SOAP (Box, 2000) are good examples. The Extensible Mark-up Language (XML) is a method for storing structured data in text files. Because it is platform independent and well supported, XML is becoming a popular method for data storage in both desktop and Web-based applications. Access to XML documents is usually made through a Document Object Model (DOM) either from a client application or web browser. The XML DOM is itself a COM object and exposes a number of interfaces for creating, modifying and traversing XML documents. The Predictive Model Markup Language (PMML) is an emerging standard for representing data mining models and rules using XML (DMG, 2001). The Simple Object Access Protocol (SOAP) is an alternative to standard RPC as a method of interoperations between components running on remote machines. SOAP uses HTTP as a transport layer and XML to describe the data being sent and received; the approach overcomes many of the shortcomings of both DCOM and CORBA.

Reusable Across Multiple Projects. Developing reusable components is a necessity in business. For a company working with many related products and projects, it is a great benefit to have a collection of reusable components that can readily be used to solve a new problem. Approaching projects in this way provides a more rapid solution to writing custom and prototype software and leads to an increased return on the original investment spent developing the components. Components that model data mining algorithms can also be used to develop software for niche markets, such as customer relationship management and fraud detection, and can be embedded in other packages. For example, data mining components can now be linked into SQL Server 2000 and models can be constructed using extensions to the standard SQL language. For a component to be made compatible, it must support a number of interfaces defined in the OLE-DB for Data Mining specification (Microsoft, 2000).

Improved Test Environment. The testing phase of the software development lifecycle is a key phase for any commercial vendor. Test plans are created to analyse the stability and efficiency of programs. For a data mining toolkit, testing the algorithm is a separate task to testing the interface. The use of scripting languages to access the algorithm components directly, without the need or overhead of the user interface, creates a more workable testing environment. Once the tests have been completed and any problems have been fixed, providing the interfaces have not been modified, the same script can be used to retest and validate the component.

Software Development Kits. As this chapter shows, component-based software such as the heuristic search engines and data mining problems can be embedded inside full commercial products such as a data mining toolkit. The previous sections have also described how these components can be reused within the organisation to solve other optimisation problems and rapidly prototype and develop custom applications. Furthermore, it is possible for a company to market their collection of components as a software development kit (SDK). SDKs allow other users to embed the techniques into their own applications, develop new problems and add their own user interface. In the case of the components described here, there is potential to market a set of generic components to solve optimisation problems or an SDK for rule discovery. Some data mining SDKs are already available and are listed at KDNuggets.

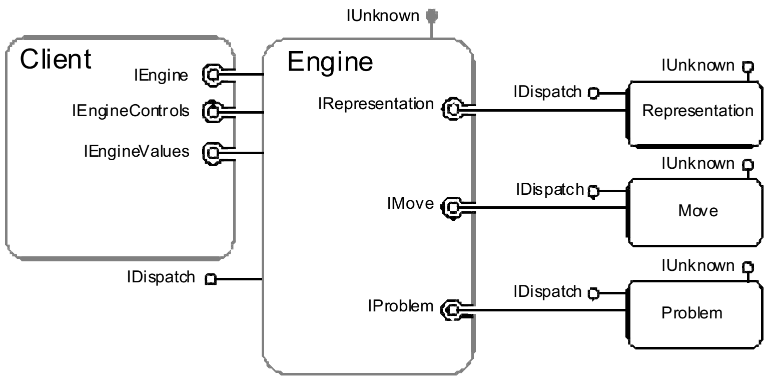
Implementing Heuristic Search Engines Using COM

The original Templar framework design used engines, problems and representations as a method of implementing solutions to optimisation problems. These core modules have been retained in the move from static libraries to components, as the framework is stable, well documented and justified (Jones, 2000). The original design uses C++ base classes as a generic method of accessing classes. This is a framework for code reuse at the source code level and allows different problems to be recompiled into the application with minimal changes to the source code. The COM design takes this a stage further. By having each core component support a generic interface, the components can be interchanged within the application at runtime. Components supporting a similar set of interfaces can be reused at the binary level without the need to recompile or rebuild the client application. Figure 3 shows how components can be snapped into an engine, and likewise the engine snaps into the client application. As an example of the generic nature, the engine can work with any problem component as long as it implements the IProblem interface, the method of communication used between engines and problems.

In the same way that a problem exposes a generic interface, the engine exposes a number of generic interfaces that can be used by client applications. In the case of the engine in Figure 3, the client can communicate with the engine through three generic interfaces. The design allows the client to work with different heuristic search engines requiring minimal changes to the internal code. The above design works well for developing different problem and representation components that can be embedded inside commercial software. However, from an algorithm research and development perspective, internal changes to the engines would still mean the entire component has to be rebuilt.

The next stage in the design process was to isolate key areas of the search algorithms that could be abstracted into snap-in components; two areas were chosen: the section where moves are generated and the section where they are accepted. A number of methods common to each section were identified and grouped together to form generic interfaces. Acceptor components would support IAcceptor and implement a method that determines whether or not a move is accepted and return the decision to the engine. Similarly, a generator would implement IGenerator and methods to generate new solutions. For example, a hill climber engine would usually have an acceptor based simply on whether one fitness value is an improvement of another, and a generator that builds moves for the entire neighbourhood of the current solution. For modern heuristic algorithms, such as simulated annealing, further components can be created. The cooling schedule is central to the design of a SA engine and numerous techniques exist to control the cooling rate of the temperature. Cooling schedules can therefore be implemented as components and snapped into the engine at runtime, eliminating the need to store all available schedules inside the engine. In all of the above areas, it is important to remember two major benefits of using COM from a research and development point of view. First, components can be added and removed without changing the engine, making it very easy to experiment with different approaches, especially batch runs controlled by a script. Secondly, when developing and modifying algorithms only the component being updated has to be rebuilt; the engine, client application and test

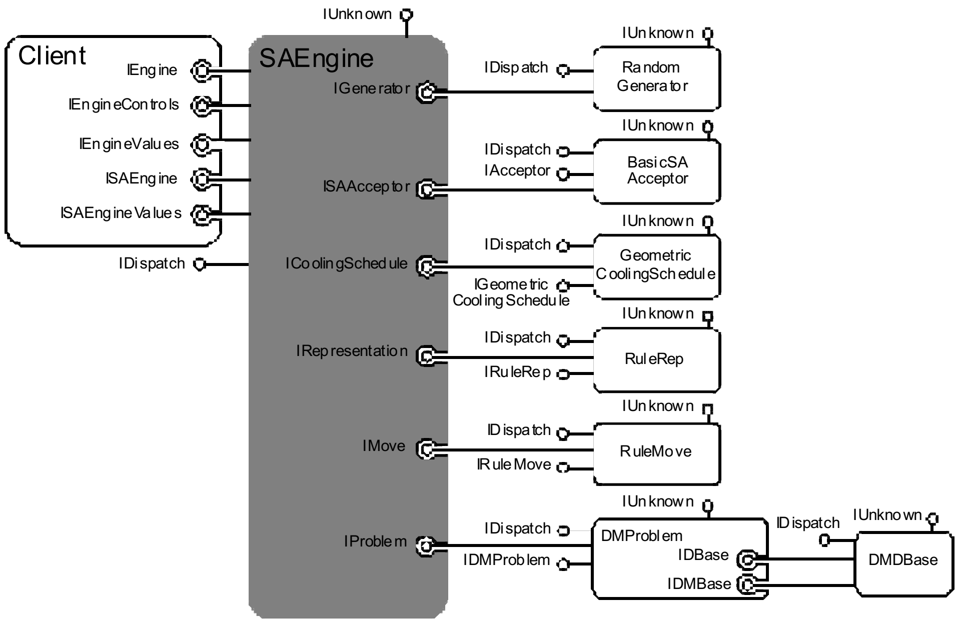
Figure 3: Application structure using COM-based search engine components



environment will require no modifications. Figure 4 shows how a simulated annealing search engine (described in the next section) has a number of snap-in components that communicate through generic interfaces. The full design describing the snap-in components as well as detailed interface definitions and engine components for hill climbers, SA and tabu search can be found in Howard (2001).

Of the three possible implementation methods, the ActiveX Template Library (ATL) was chosen because of the support for automating much of the repetitive tasks required in building COM objects and the ability to generate small and efficient components. The ATL is the most recent C++ library for developing COM objects and was designed to build lightweight and efficient components with minimal dependencies on other libraries unlike MFC. Many of the recurring tasks, such as writing IUnknown and dispatch methods, managing inheritance of multiple interfaces and standard marshalling are provided automatically through using the ATL. By removing most of the overheads of COM programming, the programmer is left to concentrate on the design and implementation details of the custom components. IDL was used to generate the type libraries so that the engine could easily reference and use the snap-in components, and similarly, the client can reference the engine. The current design has all component servers implemented as in-process DLLs as this is the most efficient method of communication between objects. As a result of this decision, proxy and stub marshalling code is not required as all of the components will be running within the same process on the same machine.

Figure 4: Application design using the SAEngine and snap-in components



A SIMULATED ANNEALING ENGINE SERVER

The SA engine server is an in-process (DLL) component server that contains the COM class for the simulated annealing search engine shown in Figure 4. The SA engine is more complex than the basic hill climber but still uses the same basic algorithm structure and modules. Throughout the following section, *m* refers to a move structure; *s* refers to a solution; and *f* refers to the fitness of a solution. Three suffixes are also used: *p* for proposed solutions; *c* for current solutions; and *b* for best solutions.

```
$s_c$ = Generate initial solution
$f_c$ = Evaluate($s_c$)
$s_b = s_c$
$f_b = f_b$
terminated = false
t = CoolingSchedule.InitialTemp
while terminated = false
    Perform Iteration
    t = CoolingSchedule.CalculateNextTemp
    if iteration >= iteration limit
            terminated = true
    end if
end while
```

An iteration of the SA engine is defined to be:

```
n = number of proposed moves per iteration
while n <= proposed moves limit
    $m_p$ = GenerateNextMove($s_c$)
    $s_p$ = PerformMove($m_p$, $s_c$)
    $f_p$ = Evaluate($s_p$)
    if Accepted($f_c$, $f_p$, t)
            $s_c = s_p$
            $f_c = f_p$
            if $f_c < f_b$
                    $s_b = s_c$
                    $f_b = f_c$
            end if
    end if
    increment n
end while
```

SAEngine Component

The SAEngine COM class supports the four generic interfaces that must be supported by all heuristic search engines but also has two engine-specific interfaces: ISAEngine and ISAEngineValues. During each iteration, the engine uses interface pointers to access the snap-in components RandomGenerator, BasicSAAcceptor and GeomtricCoolingSchedule. The SAEngine also has a number of functions that are local to the COM class. One of the important tasks during the initialisation stage of the engine is to create the internal solutions and components. Representations for current, best and proposed solutions are created using the ProgID for the RuleRep class. The client is required to call set the representation type through the IEngine interface to define this value.

RandomGenerator Component

Move generators are one collection of snap-in components used to supply the engine with a move structure capable of generating the next solution. The RandomGenerator component exposes its functionality through the IGenerator interface. The generator selects at random one of the move operators from the representation as well as a random position at which it will be applied. This information is returned to the engine and used to describe and perform the next move.

BasicSAAcceptor Component

Acceptors form another group of snap-in components for the engines. Acceptors are used to determine whether or not a solution should be accepted. The simplest acceptor would be found in a hill-climber engine and based simply on whether one fitness value was an improvement of another. The SA algorithm is based on the concept of probabilistically allowing worsening moves to be accepted during the search. The probability is calculated $e^{-\delta/t}$ using where δ is the difference between current and proposed fitness values and t is the current temperature. BasicSAAcceptor uses a method from the generic IAcceptor interface to calculate whether a move should be accepted. The ISAAceptor interface is used to pass the current temperature value from the engine to the acceptor.

GeometricCoolingSchedule Component

Modifying the initial temperature and selecting an alternative cooling schedule is a common method of trying to find better solutions using a SA engine. By making the cooling schedule a snap-in component of the engine, the schedule can easily be changed by the client to perform further experimentation. Components that support ICoolingSchedule could be based on techniques such as geometric or arithmetic progressions, or the Lundy and Mees (1986) cooling schedule. Some approaches will require additional parameters and thus require additional interfaces to set these values. For example, the Lundy and Mees calculation requires the value of β to be set, whereas the ratio will be necessary for a geometric cooling schedule.

THE DATABASE AND DATA MINING COMPONENT SERVER

The DMEngSvr server is comprised of three core components which are central to solving data mining problems and do not form part of the generic set of engine components. The three components are: DMDBase, the database server; DMProblem, the module which defines how rules are evaluated and scored; and RuleRep, which represents a rule as a solution to an optimisation problem.

DMDBase Component

The DMEngine library (Howard, 1999b) is a collection of C++ classes that provide database access and data mining algorithms. The DMDBase component wraps the functionality provided by DMEngine with COM interfaces. The existing classes are statically linked into the component which only exposes a limited number of methods and properties through two interfaces: IDBase and IDMDBase. As the internal operations of the component are hidden to the client, the database implementation can be changed and improved at any time without the need to rebuild the client. IDBase is an interface for setting parameters and accessing information about the underlying database. The implementation details of the database are hidden to the client, who would be unaware of whether the database was being held entirely in memory using C-style arrays or stored in a DBMS on a remote server.

DMProblem Component

The DMProblem component is similar to the DataMine class used with the search engines from the Templar framework. The problem component encapsulates the data, evaluation routines and parameters associated with solving data mining optimisation problems. The component supports two custom interfaces: IProblem and IDMProblem. The IProblem interface is used to initialise the problem and provide the engine with fitness values by evaluating solutions. The generic nature of this interface allows different problems to be added and removed from the engine at run-time. Unlike the generic IProblem interface, IDMProblem exposes properties and methods specific to the data mining class of problems. Parameters include the coefficient used to bias rule accuracy or coverage and a value that controls the complexity of the rules generated.

RuleRep Component

The RuleRep component represents solutions in the form of rules. The RuleRep component supports two custom interfaces. IRepresentation is the generic interface that must be provided by all representation components that are capable of interacting with search engines. Generic representation types usually include arrays of integers or doubles, bitstrings and permutations; the RuleRep class described here is a more specialised representation used for rule discovery. IRepresentation exposes methods to perform and undo moves on a representation; selects a initialisation routine; and generates neighbourhood moves using a number of operators. The IRuleRep interface exposes the additional methods to get and set clauses in a rule representation object as well as a single property, Length, to set the length of the solution.

RuleMove Component

To perform a move on a representation the engine could simply request that a change be made to an existing solution. However, a two-phase process of performing a move using a structure (in this case, another component) to store the attributes of the representation to be changed has a number of benefits. First, as the component can also store a set of previous values, a move can be undone if it has been rejected by the engine; in most cases the undo process will be more efficient than cloning the previous solution. Secondly, as information is stored about which attributes of the solution have been changed, this data can be in attribute-based memory structures such as those used in Tabu Search (Glover & Laguna, 1995).

CONCLUSIONS

This chapter justifies and describes the movement of libraries for solving optimisation problems from the object-oriented paradigm to the component-based paradigm. In doing so, a number of important commercial and research development issues have been addressed including how a component-based approach would benefit the development of data mining software. The design uses a COM approach to build applications with embedded search engines using many small components that can readily be interchanged at run-time. Constituent parts of the core search algorithms have been identified and modelled with snap-in components; by doing so, the overall development of a large application becomes easier, as is designing and testing new algorithms. A data mining problem component, representation and move structure have been written to solve data mining rule induction problems. Heuristic search engines are used to find rules describing one particular class of records in a database, a technique sometimes referred to as nugget discovery.

There is wide support for COM in the industry and developing software in this way is becoming normal practice. The major benefits of components for software vendors are:

- reuse between projects;
- useful for rapidly developing prototype applications, e.g., for consultancy services;
- could form part of a software development kit that could be marketed for users to develop their own applications.

FUTURE WORK

The work described here focuses on a collection of components designed and developed to solve optimisation problems. The movement to a COM-based approach acts as a proof of concept as a number of working search engines have been written, tested and exploited in the area of data mining. Two sections of the algorithms, namely acceptors and generators, were identified as areas that could be developed into components. Other sections, such as initialisation routines and iterations could similarly be developed but have more of an impact on the overall design. If enough areas can be factored out, it becomes possible to implement a generic engine component that can be transformed into a particular engine based on the components that are snapped into it. A side effect of generalising too far is that it becomes more difficult to set algorithm specific parameters. Consider a GUI based application that allows users to change which engine they use to solve data mining problems. The user interface must allow users to set parameters on all components associated with the system. But, as components are updated and new parameters are added, the user interface needs to adapt to show the correct configuration options. This means that the GUI needs to be built dynamically using a list of parameters that can be retrieved from the component at run-time. One solution may be for components to have an IParameterList interface which can be queried to dynamically build a list of parameters. Finally, using DCOM, the components can be configured to work over remote systems with the engines running on one machine and the problem component (linked to a database server) running on another. However, in data mining problems, the majority of the work performed by the system is on evaluating the rules. This would still take place on a single machine, as the database server is the only component that can evaluate rules and provide the problem with enough information to calculate a fitness value. To overcome this, the database problem (and possibly database) needs to be split up into smaller problems and distributed across a number of machines, which means looking at the area of distributed data mining (Chattratichat, Darlington & Ghanem, 1997).

REFERENCES

Allen, P. & Frost, S. (1998). *Component-based development for enterprise systems.* New York: Cambridge University Press.

Box, D. (1997). Q&A ActiveX/COM. *Microsoft Systems Journal, 12(5),* 93-108.

Box, D. (2000). A young person's guide to the simple object access protocol. *MSDN Magazine, 15 (3)*, 67-81.

Bray, T., Paoli, J. & Sperberg-McQueen, C. M. (1998). *Extensible markup language (XML) 1.0.* Retrieved April 17, 2001 from the World Wide Web: http://www.w3.org/xml.

Chattratichat, J., Darlington, J., Ghanem et al. (1997). Large scale data mining: challenges and responses. In *Proceedings of the KDD '97 Conference.* California: AAAI Press, 143-146.

DMG (2001). *PMML 1.1: Predictive Model Markup Language.* Retrieved April 17, 2001 from the World Wide Web: http://www.dmg.org

Glover, F. & Laguna, M. (1995). Tabu search. In C. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems* (pp. 70-150). London: McGraw-Hill.

Howard, C. M. (1999). *Datalamp version 1 requirements document* (Lanner Group Report). Redditch, UK: Lanner Group.

Howard, C. M. (1999). *DMEngine class reference* (Technical Report SYS–C99–03). Norwich, UK: University of East Anglia.

Howard, C. M. (2001). Tools and techniques for knowledge discovery. *PhD Thesis to be submitted 2001.* Norwich, UK: University of East Anglia.

Hludzinski, B. (1998). Understanding interface definition language: A developer's survival guide. *Microsoft Systems Journal, 13 (8)*, 51-66.

Jones, M. S. (2000). An object-oriented framework for the implementation of search techniques. *PhD Thesis*. Norwich, UK: University of East Anglia, Norwich.

Jones, M. S., McKeown, G. P. & Rayward-Smith, V. J. (2001). Distribution, cooperation and hybridisation for combinatorial optimization. In S. Voss & D. L. Woodruff (Eds.), *Optimization Class Libraries*. Kluwer Academic Publishers (in press).

KDNuggets (2000). Retrieved April 17, 2001 from the World Wide Web: http://www.kdnuggets.com

Lundy, M. & Mees, A. (1986). Convergence of an annealing algorithm. *Mathematical Programming, 34*, 111-124.

Matena, V. & Hapner, M. (1999). Enterprise JavaBeans specification version 1.1. Retrieved April 17, 2001 from the World Wide Web: http://java.sun.com.

Microsoft Corporation (1999). *Mastering distributed application design and development using Visual Studio 6* (MS1298A). Redmond: Microsoft Corporation.

Microsoft Corporation (1999). *Component development using the Active Template Library 3.0* (MS1304A). Redmond: Microsoft Corporation.

Microsoft Corporation (2000). *OLE-DB for Data Mining specification v1.0.* Retrieved April 17, 2001 from the World Wide Web: http://www.microsoft.com/data/oledb/dm

Object Management Group (2000). The Common Object Request Broker: Architecture and specification rev. 2.4. Retrieved April 17, 2001 from the World Wide Web: http://www.omg.org

Rayward-Smith, V. J., Debuse, J. C. W., & de la Iglesia, B. (1996). Discovering knowledge in commercial databases using modern heuristic techniques. In *Proceedings of the KDD '96 Conference.* California: AAAI Press, 44-49.

Shepherd, G. & Wingo, S. (1996). *MFC internals.* Massachusetts: Addison Wesley Developers Press.

Sherlock, T. P. & Cronin, G. (2000). COM beyond Microsoft. Boston: Digital Press.

AUTHOR NOTES

This work was undertaken within a TCS Programme (no. 2552) between the School of Information Systems at the University of East Anglia and Lanner Group Limited.

Chapter VI

Clustering Mixed Incomplete Data

José Ruiz-Shulcloper
University of Tennessee-Knoxville, USA
Institute of Cybernetics, Mathematics and Physics, Cuba

Guillermo Sánchez-Díaz
Autonomous University of the Hidalgo State, Mexico

Mongi A. Abidi
University of Tennessee-Knoxville, USA

In this chapter, we expose the possibilities of the Logical Combinatorial Pattern Recognition (LCPR) tools for Clustering Large and Very Large Mixed Incomplete Data (MID) Sets. We start from the real existence of a number of complex structures of large or very large data sets. Our research is directed towards the application of methods, techniques and in general, the philosophy of the LCPR to the solution of supervised and unsupervised classification problems. In this chapter, we introduce the GLC and DGLC clustering algorithms and the GLC+ clustering method in order to process large and very large mixed incomplete data sets.

CLUSTERING MIXED INCOMPLETE DATA

In the process of Knowledge Discovery from Data (KDD), one of the most important tasks is to classify data. It is well known that one of the most powerful tools to process data in order to extract knowledge is the class of clustering algorithms, whose purpose is (in the KDD context) to solve the following problem. Given a similarity measure Γ (not necessarily a distance function) between pairs of object descriptions in some representation space and a collection of object descriptions in that space, find a structuralization of this collection. These sets could form hard or fuzzy cover or partition (Martínez-Trinidad, Ruiz-Shulcloper, & Lazo-Cortés, 2000a; Ruiz-Shulcloper, & Montellano-Ballesteros, 1995) of the data set. In other words, finding the similarity relationship between any pair of objects under a certain clustering criterion without utilizing a priori knowledge about the data and with the following additional constraints: i) the use of computing resources must be minimized and ii) the data set could be *large* or *very large*.

Also, it is well known today that in some areas such as finance, banking, marketing, retail, virtual libraries, healthcare, engineering and in diagnostic problems in several environments like geosciences and medicine among many others, the amount of stored data has had an explosive increase (Fayyad, Piatetsky-Shapiro, Smyth, & Uthurusamy, 1996). In these areas, there are many instances where the *description of the objects is nonclassical*, that is, the features are not exclusively numerical or categorical. Both kinds of values can appear simultaneously, and sometimes, even a special symbol is necessary to denote the absence of values (missing values). A *mixed and incomplete* description of objects should be used in this case. Mixed in the sense that there are simultaneously categorical and numerical features; incomplete because there are missing values in the object descriptions.

The study of the similarity relationships with mixed incomplete descriptions of objects is the principal aim of LCPR (Martínez-Trinidad et al., 2000a; Ruiz-Shulcloper, & Montellano-Ballesteros, 1995; Dmitriev, Zhuravlev, & Krendelev, 1966).

In order to gain clarity and understanding, we will establish conventional differences between *Data Set* (DS), *Large Data Set* (LDS) and *Very Large Data Set* (VLDS). In a mining clustering (also in a supervised classification) process DS will be understood to mean a collection of object descriptions where the size of the set of descriptions together with the size of the result of the comparison of all pair wise object descriptions, that is, the *similarity matrix*, does not exceed the available memory size. LDS will mean the case where only the size of the set of descriptions does not exceed the available memory size, and VLDS will mean the case where both sizes exceed the available memory size.

In addition, we propose conventional differences between the *Very Large Data Set Clustering Algorithm* (VLDSCA), the *Large Data Set Clustering Algorithm* (LDSCA), and the *Data Set Clustering Algorithm* (DSCA). If we denote OT_A as the run-time complexity, and OE_A as the space complexity of a clustering algorithm CA, then we have a VLDSCA iff $OT_A < O(m^2)$ and $OE_A < O(m)$. We have a LDSCA iff

$OT_A \leq O(m^2)$ and $O(m) \leq OE_A < O(m^2)$. Otherwise, the CA will be a DSCA.

In this chapter we will discuss how the ideas of LCPR could be extended to KDD, what is done today, and what is possible and necessary to do in the near future. First of all, several necessary concepts and results from LCPR will be introduced in order to have a clearer understanding about the ideas and possibilities of the LCPR in KDD. Some related works are considered and their contribution to the solution of the formulated problem are discussed. After that, three *mixed incomplete data* (MID) *clustering algorithms* are presented and several experimental results are analyzed. Finally, a brief projection of future ideas and work are commented.

BACKGROUND

First of all, it is important to be reminded of the principal differences and connections between the two concepts: distance and similarity functions. Sometimes the term *distance* is intended as a synonym of *metric* (*measure*), and *s*ometimes it is a particular case of a metric. The latter is our preference. Both distance and similarity functions are particular cases of metrics. There are many types of distance functions or dissimilarity measures (pseudo-distance, semi-distance, distance, pre-distance, ultra-distance, etc.) depending on the presence of certain properties. In general, when we are talking about distance, it is assumed to be a symmetric function with two additional important properties in the case of the distance function: the triangular inequality and the requirement to be positive-definite, that is, the function can take the value zero (minimal value of the image of the distance function) only when comparing an element with itself. Only in this case is the function zero. Furthermore, it is true that the dual of each of all these particular cases of metrics are similarity functions. Following Goldfarb (1985) let us see these definitions in more detail.

Definition 1. By a *pseudo-distance function* (*dissimilarity coefficient*) we mean a non-negative real-valued function π defined under the Cartesian product of a set P, satisfying the following two conditions:

a) $\forall p_1,p_2 \in P\ \pi(p_1,p_2)=\pi(p_2,p_1)$ (symmetry)

b) $\forall p \in P\ \pi(p,p)=0$ (reflexivity)

Definition 2. A pseudo-distance function which satisfies the following additional property

c) $\forall p_1,p_2 \in P\ \pi(p_1,p_2)=0 \Rightarrow p_1=p_2$ (definiteness)

is called a *semi-distance function* (*definite dissimilarity coefficient*).

Finally,

Definition 3. A semi-distance function which satisfies the following condition

d) $\forall p_1,p_2,p_3 \in P\ \pi(p_1,p_3) \leq \pi(p_1,p_2)+\pi(p_2,p_3)$ (triangle inequality)

is called a *distance function* or simply a *distance*.

Although the similarity function is responsible for the very origin of pattern recognition (Goldfarb, 1985), in general, it has been considered practically as an equivalent to a dissimilarity function. That is why, for many authors, it is very easy to think that if we have a distance function (dissimilarity measure) we can get a similarity function by using the opposite (or the inverse) of this function. And that is right: the opposite or the inverse of a distance function is a similarity function, but there are many other similarity functions for which there is no distance function from which they can be obtained. In general, it is not true that a similarity function is the dual function of a distance.

In the literature, the similarity functions have not received the same attention that the distance functions have received. We can find a simple definition of a similarity function in Jain & Dubes (1998) or a more complex one in Martínez-Trinidad et al. (2000) and Alba-Cabrera (1998). The necessity to work with a more flexible function than a distance function, even with other similarity functions that are not dual of some distance, is a requirement imposed by real-world practice. There are many practical problems in which this flexibility is necessary, see for example Gómez-Herrera et al., 1994; Ortíz-Posadas, Martínez-Trinidad, & Ruiz-Shulcloper, 1996; Martínez-Trinidad, Velasco-Sánchez, & Contreras-Arévalo, 2000b).

Now we will discuss the principal concepts and results of LCPR and the possibilities of the use of its tools for KDD using mixed incomplete (large, very large) data sets.

LCPR originated in the Soviet Union, with the publication by Dmitriev, Zhuravlev and Krendelev (1966). The principal task of that paper was to find a solution to the prognosis of mineral resources. Later on, other works appeared in which features of different natures were considered, that did not involve first Boolean alone or qualitative features alone without transformation (Gómez-Herrera, et al. 1994; Ortíz-Posadas, et al. 1996; Martínez-Trinidad et al., 2000b).

SOME NECESSARY CONCEPTS

Let U be a universe of objects, which could be structured in K_1,..., K_r classes (where $r \geq 2$), and described in terms of a finite set of features $R = \{x_1,...,x_n\}$ used to study these objects. Each of these features has associated with them a set of admissible values M_i, i=1,...,n. For M_i no algebraic, topologic or logic structure is assumed. These sets of values, unlike the previous approaches, could be of any nature: quantitative and qualitative simultaneously. They could be a subset of real numbers; spatial points; terms of some dictionary; propositions or predicates of some artificial or natural language; functions; matrices; and so on. Each of these sets contains a special symbol "*" denoting the absence of a value for the feature x_i in the description of an object O (missing data).

Definition 4. By *mixed incomplete description of object* O, an n-ary tuple of nominal, ordinal and/or numerical values can be defined as $I(O)=(x_1(O),...,x_n(O))$, where x_i:U→M_i, $x_i(O) \in M_i$; i=1,...,n. Let T be a subset of R, $I|_T(O)$ will denote a *sub description* of O exclusively in terms of the features of T.

From here on consider that $U=M_1 x \ldots x M_n$, the Cartesian product of the admissible value sets of features of R and I(O)=O, in order to simplify the notation in all the cases that will not necessarily specify the type of description.

Let $M=\{O_1,...,O_m\}\subseteq U$, which in the case of feature selection and supervised classification problems will be the union of the sets $K_1',..., K_r'$, K_i', i = 1,...,r, which are not necessarily disjoint sets and not necessarily crisp sets.

Similarity Measures

Definition 5. A *comparison criterion* $\varphi_i:M_i\times M_i \longrightarrow L_i$ is associated to each x_i (i = 1,...,n), where: a) $\varphi_i\ (x_i(O),x_i(O)) = \min\{y|\ y\in L_i\}$ if φ_i is a *dissimilarity* comparison criterion between values of variable x_i or b) $\varphi_i\ (x_i(O),x_i(O)) = \max\{y|\ y\in L_i\}$ if φ_i is a *similarity* comparison criterion between values of variable x_i for i = 1,…n. φ_i is an evaluation of the degree of similarity or dissimilarity between any two values of the variable x_i when L_i is a totally ordered set, i=1,…,m.

A magnitude may be calculated for each pair of objects in U. This magnitude is obtained by applying a function, which will be called a *similarity function.* This similarity function can be defined for any subset of R.

Definition 6 (Martínez-Trinidad et al., 2000a). Let a function $\Gamma: \cup_{T\subseteq R} (M_{i1}x...xM_{ip})^2 \rightarrow L$, where L is a totally ordered set; $T=\{x_{i1},...,x_{ip}\}\subseteq R$, $p\geq 1$, and where Γ satisfies the following two conditions:

1) Let $T_1,\ldots,T_s$ nonempty disjoint subsets of R, $\lhd$ is the order in L and $T= \cup\ T_i$, i=1,...,s, then we have:
   if for all h=1,…,s $\Gamma(I|_{Th}(O_i),I|_{Th}(O_j))\lhd\Gamma(I|_{Th}(O_f),I|_{Th}(O_g))$, then
   $\Gamma(I|_T(O_i),I|_T(O_j))\lhd\ \Gamma(I|_T(O_f),I|_T(O_g))$
2) For all sub-description in $\cup_{T\subseteq R}\ (M_{i1}x...xM_{ip})^2$ we have
   a) $\max\{\Gamma(I|_T(O_i),I|_T(O_j))\}=\{\Gamma(I|_T(O_i),I|_T(O_j))\}$, $O_j\in M$
   b) $\Gamma(I|_{Ti}(O),I|_{Ti}(O)) = \Gamma(I|_{Tj}(O),I|_{Tj}(O))$
   c) $\Gamma(I|_T(O_i),I|_T(O_j)) = \Gamma(I|_T(O_j),I|_T(O_i))$

Γ denotes a *similarity function* and it is an evaluation of the degree of similarity between any two descriptions of objects belonging to U.

Similarity Matrices

In LCPR, clustering criteria are essentially based on *topological* (not necessarily metrical) relationships between objects. This approach responds to the following idea: given a set of object descriptions, find or generate a *natural* structuralization of these objects in the representation space.

Any structuralization of a representation space depends on a similarity function Γ (also on comparison feature values criteria) and a clustering criterion that expresses the way in which we can use Γ.

In general, the clustering criteria have as parameters: a symmetric matrix (if Γ is a symmetric function) $|\Gamma_{ij}|_{mxm}$ denominated *similarity matrix* for which each $\Gamma_{ij}=\Gamma(O_i,O_j)\in L$; a property Π, which establishes the way that Γ may be used, and a *threshold* $\beta_0\in L$.

Definition 7. O_i, $O_j\in M$ are β_0-*similar objects* if $\Gamma(O_i,O_j)\geq\beta_0$. In the same way

$O_i \in M$ is β_0-*isolated object* if $\forall O_j \neq O_i \in M \ \Gamma(O_i,O_j)<\beta_0$.

The β_0-threshold value can be used to control how close a pair of objects must be in order to considered similar. Depending on the desired closeness, an appropriate value of β_0 may be chosen by the expert/user.

The determination of this threshold is not easy, but it is not an impossible process. β_0 could be obtained by expert criteria. Also, it is possible to conduct some numerical experiments.

Definition 8 (Martínez- Trinidad et al., 2000a). For a *crisp clustering criterion* $\Pi(M,\Gamma,\beta_0)$ we mean a set of propositions with parameters M, Γ and β_0 such that:

a) It generates a family $\tau=\{NU_1,...,NU_c\}$ of subsets (called *nucleus*) of M (crisp clusters) that:

I)$\forall$ $NU \in \tau$ $[NU \neq \emptyset]$;

II) $\cup NU = M$, $NU \in \tau$

III)$\neg\exists$ NU_r, $NU_{j1},..., NU_{jk} \in \tau$ $[NU_r \subseteq \cup NU_{jt}$, $t=1,...,k$; $jt \neq r]$; and

b) it defines a relationship $R_\Pi \subseteq M \times M \times 2^M$ (where 2^M denotes the power set of M) such that:

IV) $\forall O_i,O_j \in M$ $[\exists NU \in \tau \ \exists S \subseteq M[O_i,O_j \in NU \Leftrightarrow (O_i,O_j,S) \in R_\Pi]$.

Definition 9. We say that a subset $NU_r \neq \emptyset$ of M is a β_0-*connected nucleus* (*component*) with respect to Γ and β_0 if

a)$\forall O_i,O_j \in NU_r \exists \in O_{i1},...,O_{iq} \in NU_r[O_i=O_{i1} \wedge O_j=O_{iq} \wedge \forall p \in \{1,...,q\text{-}1\} \ \Gamma(O_{ip},O_{ip-1}) \geq \beta_0]$

b) $\forall O_i \in M$ $[(O_j \in NU_r \wedge \Gamma(O_i,O_j) \geq \beta_0) \Rightarrow O_i \in NU_r]$, and

c) Any β_0-isolated element is a -*connected nucleus* (*degenerated*).

ALGORITHMS FOR CLUSTERING MIXED INCOMPLETE DATA

Several techniques have been developed to cluster large and very large data sets. CLARANS (Ng & Han, 1994), DBSCAN (Ester, Kriegel, Sander & Xu, 1996), GDBSCAN (Sander, Ester, Kriegel & Xu, 1998), BIRCH (Zhang, Ramakrishnan & Livny, 1996) and IncDBSCAN (Ester, Kriegel, Sander, Wimmer & Xu, 1998) have given solutions, with higher or lower efficiencies. The objects handled by these algorithms are described in metric spaces, and these algorithms use a distance D to compare two objects of the data set. That is, D is a numerical symmetric function, which fulfills the triangle inequality. One of the drawbacks of the mentioned algorithms is that they were not developed to cluster data sets defined in spaces in which the similarity relationships do not satisfy the triangle inequality and in some cases are not symmetric.

Recently, algorithms for processing qualitative data and also mixed incomplete large and very large data sets have started to develop. ROCK (Guha, Rastogi & Shim, 1999), Chameleon (Karypis, Han & Kumar, 1999), GLC (Sánchez-Díaz & Ruiz-Shulcloper, 2000), DGLC (Ruiz-Shulcloper et al., 2000), and GLC+ (Sánchez-Díaz, Ruiz-Shulcloper, 2001) are the most representative instances in this vein. These techniques handle objects defined in nonmetric spaces and use a similarity function S to compare two objects of the data sets.

ROCK is a hierarchical algorithm developed for clustering data sets with only categorical attributes with missing values, but not for mixed incomplete data. This algorithm uses the concept of *link* in order to measure the similarity between two objects. The concept of *neighbor* used in this algorithm is the same as the previously mentioned concept of β_0-similar. In this algorithm, the threshold is a user-defined parameter. The number of links between two objects is the number of common neighbors of the objects. The number of links will determine the membership of an object to a cluster. This algorithm uses a criterion that maximizes the sum of links between all pairs of objects that belong to the same cluster, while minimizing the sum of links of any two objects in different clusters. The whole process is supported by the determination of the neighbors of an object that is employed in the similarity function. The authors suggested for the treatment of the categorical attributes to use the Jaccard coefficient.

ROCK calculates the similarity matrix (pair-wise object similarities), so if the number of objects is very large, it is ineffective to cluster these data sets. The authors' proposal is to take only a few samples of the original data set in order to make the algorithm scalable for large data sets. After that, each of the remaining objects is assigned to the cluster in which the object has the maximum number of links. But, it is clear that the results are dependent on the order. Besides, ROCK is unworkable for clustering large and very large mixed incomplete data sets.

Another important attempt to solve a problem resembling the mixed incomplete data clustering is Chameleon. Chameleon is also an agglomerative hierarchical algorithm proposed for clustering data sets not necessarily described in a metric space. Chameleon considers both the *interconnectivity* and the *closeness* between clusters to identify the most similar pair of clusters. These values are obtained from the k-nearest-neighbor graph associated with the assumed given matrix of the similarities between data. So, Chameleon is applicable to any data set for which a similarity matrix is available (or can be obtained). The assumption that a similarity matrix is given avoids the problem of treating the mixed incomplete data, so this problem is assumed resolved. When this matrix is not given, the problem is obviously associated with the similarity function. In the Chameleon case it is not clear how to deal with mixed incomplete data.

The main limitation of Chameleon is that the runtime complexity to generate the similarity matrix is quadratic, making it inoperative for clustering large and very large data sets.

Other efficient algorithms for clustering data sets have been developed in artificial intelligence, such as Kohonen's self organizing feature maps (Kohonen, 1982), and learning vector quantizer (Kohonen, 1986). However, none of these models process input objects described by quantitative and qualitative attributes (mixed data). In addition, these techniques were not developed to address clustering large or very large data sets.

Starting from the real existence of a number of complex data sets and the incapability of the previously developed clustering algorithms to address mixed incomplete data processing, several other approaches in which the methods, the techniques and in general, the philosophy of the LCPR were applied to the solution of supervised and unsupervised classification problems, using huge data sets containing merged quantitative and qualitative data, have recently appeared. These approaches constitute the conceptual base and have produced the first results in an emerging new research area in Data Mining that we called Mixed Incomplete Data Mining (MID Mining).

Global Logical-Combinatorial (GLC) Clustering Algorithm

There are several traditional methods for calculating connected components in a mixed incomplete data set. In Martínez-Trinidad et al. (2000a), a methodology is described that performs this calculation as a part of the solution to other problems. In general, these techniques have two fundamental phases: the calculation of the similarity matrix of the objects and the generation of the clustering following some clustering criterion. The main drawback that these algorithms show is the necessity to calculate and store the similarity matrix, and when the number of objects in the data set grows considerably, then it becomes practically inefficient for their application (in size and time).

GLC is an incremental clustering algorithm for large mixed incomplete data sets that obtains a partition in connected components. This algorithm is based on LCPR.

GLC handles data in any space (not necessarily metric) and it is based on any similarity function G that compares two objects. This algorithm uses the *connected component* criterion to generate clusters from the data set. This algorithm generates the only clustering of the data set, without relying on the input order of data. The GLC algorithm detects all the connected components of the data set. Also, GLC neither calculates nor stores the similarity matrix. This is the main improvement and advantage of the GLC algorithm over the other traditional methods.

The GLC algorithm generates (incrementally) all the connected components while it is reading the objects of the data set. It uses only the necessary memory to store the clusters that are created by the algorithm. It is important to point out that the algorithm never compares two objects more than once.

The GLC algorithm proceeds in the following way: upon the arrival of a new object O_i, it is verified for some cluster G_k determined by the condition $\Gamma(O_i,O_j)\geq\beta_0$, for an object $O_j\in G_k$ (similarity condition). If no cluster fulfills the similarity condition, then O_i builds up a new cluster. Otherwise, the object O_i is added to the cluster G_k that contains the object O_j that was similar to it. At this time, further comparisons with the residual objects of the cluster G_k are not necessary.

Object O_i is then compared with the objects of the clusters G_h, with $h\neq k$, and if $\Gamma(O_i,O_j)\geq\beta_0$, for some $O_j\in G_h$, no additional comparisons with the remaining objects of G_h are needed and clusters G_k and G_h are merged to form a unique one.

This procedure is carried out with all the existent clusters. The previous process shall be performed with each object of the cluster in the data set. The GLC algorithm is shown in detail in Sánchez-Díaz and Ruiz-Shulcloper (2000).

Density-Based Global Logical Combinatorial (DGLC) Clustering Algorithm

The DGLC clustering algorithm for discovering β_0-density connected components from large mixed incomplete data sets combines the ideas of LCPR with the *density-based* (Ester et al., 1996) *notion of cluster*.

Let a similarity function Γ, a similarity threshold β_0, a natural number *MinPts*, and a large MID $M=\{O_1,...,O_m,...\}\subseteq U$ be given. Let U be a dynamical universe of objects described in terms of several kinds of features x_i, with M_i as an admissible values set for i=1,...,n.

Definition 10. An object $O\in \Pi M_i$, i=1,...,n has a *β_0-dense neighborhood* with respect to Γ, *β_0*, and *MinPts*, iff $|V_{\beta 0}| \geq$ *MinPts*, where= $V_{\beta 0}(O)=\{O_j\in M \,|\Gamma(O,O_j) \geq \beta_0\}$. We say that O is a *dense point*.

Definition 11. A non–empty set $C=\{O_1,...,O_s\}$ is named a *β_0-chain* with respect to Γ, β_0, iff for all $O_j\in C$, $\Gamma(O_j,O_{j+1}) \geq \beta_0$. In other words, $C=\{O_1,...,O_s|$ for j=1,...,s-1 $O_{j+1\in} V_{\beta 0}(O_j)\}$.

Definition 12. A β_0- chain with respect to Γ, β_0, and *MinPts* $C=\{O_1,...,O_s\}$ is named a β_0-*dense chain* with respect to Γ, β_0, *MinPts*, iff for all $O_j\in C$, O_j is a dense point. In other words

$C=\{O_1,...,O_s|$ [for j=1,...,s-1,$O_{j+1}\in V_{\beta 0}(O_j)]\wedge|V_{\beta 0}(O_j)| \geq$*MinPts*, j=1,...,s}.

Definition 13. A non–empty set $NK=\{O_1,...,O_m\}\subseteq K\subseteq M$ is named *nucleus of the β_0-dense connected component* K with respect to Γ, β_0, and *MinPts*, iff for all $O_j\in NK$ and for all $O\in M$ holds: $O\in NK$ iff there is $C=\{O_{i1},...,O_{is}\}\subseteq M$, a β_0-dense chain such that $O_j=, O_{i1},O = O_{is}, O_{it}\in NK$, for t=1,...,s-1.

Definition 14. A non–empty set $BK=\{O_1,...,O_m\}\subseteq M$ is named *border of the β_0-dense connected component* K with respect to Γ, β_0, and *MinPts*, iff for all $O_j\in BK$ there is $V_{\beta 0}(O_j)$, $0<|V_{\beta 0}(O_j)|<$*MinPts* and there is $O\in NK$ such that $O\in V_{\beta 0}(O_j)$.

Definition 15. A non-empty set $K=\{O_1,...,O_m,...\}\subseteq M$ is named a β_0-*dense connected component* with respect to Γ, β_0, and *MinPts* iff $K=NK\cup BK$.

The DGLC algorithm is shown in detail in Ruiz-Shulcloper et al. (2000).

A Method for Cluster Generation in Very Large Mixed Data Sets

The GLC+ method (*Global Logical-Combinatorial Clustering* method plus) has been introduced for clustering very large, mixed and incomplete data sets.

This is a new incremental clustering method for very large mixed incomplete data sets. This method uses the connected set criterion to generate clusters from the data set. The GLC+ method does not store all objects in the available memory. Instead, GLC+ only keeps the subset of these objects, which have a large number of similarity objects, named *skeleton objects*. Instead of comparing a new object with all of the objects in a cluster, this method only compares the new objects with the cluster's skeleton objects. In this way, GLC+ combines local and semi-global criteria to generate the clusters. Also, GLC+ neither calculates nor stores the similarity matrix.

Definition 16. Let $S \subseteq O$, $S \neq \emptyset$, S is a *b_0-connected set* with respect to Γ and β_0 if:

$\forall O_i, O_j \in S, \exists \{O_{s1}, O_{s2}, ..., O_{st}\}$ such that $O_i = O_{s1}$, $O_{st} = O_j$ and $\Gamma(O_{si-1}, O_{si}) \geq \beta_0$, for all i=2,...,t.

Each β_0-isolated object is a β_0-connected set with respect to Γ.

Definition 17. Let $\beta_0 \in V$ and S be a β_0-connected set. We say that $E=\{O_1,...,O_t\}$, $E \subseteq S$, is a *skeleton* of S if $\forall O_i \in S$, $\exists O_j \in E$, j=1,...,t, such that $\Gamma(O_i, O_j) \geq \beta_0$.

Definition 18. Let S be a β_0-connected set and E_i be all possible skeletons of S. E_{min} is a *minimal skeleton* of S if $\forall E_i \subseteq S$, $|E_{min}| \leq |E_i|$, where $|A|$ is as before, the number of elements in A.

Note that neither skeletons nor minimal skeletons are unique for a given β_0-connected set.

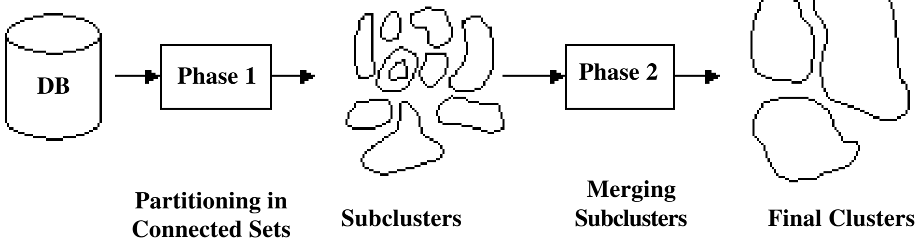
The GLC+ method has two phases. In the first phase, GLC+ clusters the objects in subclusters, which will be connected sets. In the second phase, GLC+ merges these generated subclusters, giving the final clustering of connected sets. The GLC+ method phases are shown in Figure 1.

Description of the GLC+ Clustering Method

The GLC+ method uses the GLC incremental algorithm and the SCS (Skeleton of a Connected Set, illustrated below). Like GLC, DGLC, ROCK and other clustering algorithms, GLC+ utilizes similarity threshold values. Usually, GLC calculates all connected components of the data set (we can see a connected component as a maximal connected set, see Definition 9), but here it is used in order to calculate only connected sets. From each connected set, the SCS algorithm generates a minimal skeleton or approximates a minimal skeleton. The idea of the GLC+ method is to calculate a partition of the data set in connected sets, which approximates a partition in connected components. This approximation is due to the size of the available memory.

The GLC+ method works in the following way: in the first phase, all possible

Figure 1: GLC+ method phases



connected sets are calculated until the given percentage of objects is reached or the available memory is full. In this process, an adjacent object list is created for each object O_i in which all of its β_0-similar objects in its cluster appear. In fact, at this stage we only have β_0-connected sets because not all the objects have been considered. After that, the SCS algorithm processes these sets in order to find the skeleton of each one in its corresponding cluster. Currently, we find the minimal skeletons. By doing that, we keep enough available memory in order to continue the clustering process, and at the same time, all of the remaining objects not belonging to the skeletons are kept also in their clusters, but not in the available memory. These objects are marked as border objects, and will be used in the second phase of GLC+. The SCS algorithm is called while the available memory is not fully used. The remaining objects are clustered, but they will be compared only with the objects stored in the available memory. This procedure is applied until all the objects have been clustered. In the second phase, GLC+ compares the border objects $O_b \in G_h$ and the skeleton objects $O_s \in G_l$, $h \neq l$. If O_b and O_s are similar, then the G_h and G_l clusters are merged. Both phases of GLC+ are incremental.

The GLC+ method proposed in this work is the following:

Phase 1.

Input: $O=\{O_1,\ldots,O_m\}$(data set), β_0 (similarity threshold value), % obj (percentage of objects to reach, optional).
Output: cCS (β_0-Connected Set Sub-clustering), SO and BO (skeleton objects and border objects)

1. Generate the β_0-connected sets until the %obj value is reached or the memory is still available (call GLC algorithm). Create an adjacent object list for each object in each cluster.
2. Extract the (minimal) skeletons from the β_0-connected sets formed in the previous step (call SCS algorithm).
3. Cluster new incoming objects as follows:
   - 3.1. If the object belongs to one skeleton, then tag it with the label of the skeleton, and mark it like a border object. If the object belongs to more than one skeleton, then these skeletons are merged. The object in the skeleton is not stored.
   - 3.2. If the object does not belong to any skeleton, then check if it belongs to any β_0-connected set. In the affirmative case, add the object to this β_0-connected set, and check if it belongs to any other β_0-connected sets in order to merge them in the same way that the skeletons merged. Otherwise, the object will form a new β_0-connected set.
4. If all objects have been clustered, then the algorithm finishes. In the other case, verify if the available memory is exhausted. In the affirmative case, apply the SCS algorithm to the β_0-connected sets already computed. Go to 3.

Phase 2.

```
Input: cCS ($\beta_0$-connected set sub-clustering), SO, BO (skeleton objects and border objects) and $\beta_0$ (similarity threshold value).
Output: Final $\beta_0$-connected sets clustering.
Repeat for i=1:m
    If %obj have been reached then Obtain_new_objects()
    If $O_i$ is border object, and $O_i \in G_k$, then
        If $\Gamma(O_i,O_j) \geq \beta_0$, $O_j$ is skeleton object, and $O_j \in G_h$, $h \neq k$ then $G_h \leftarrow G_h \cup G_k$
    Otherwise
        If $O_i$ is skeleton object, and $O_i \in G_k$, then
        If $\Gamma(O_i,O_j) \geq \beta_0$, $O_j$ is border object, and $O_j \in G_h$, $h \neq k$ then $G_k \leftarrow G_k \cup G_h$
```

SCS Algorithm for Finding the Skeletons of the β_0-Connected Sets

This algorithm calculates a minimal skeleton E_{min} (or *near* to it) of a β_0-connected set S. The objects that will be in the skeleton E_{min} are those objects $O_i \in S$ with the largest number of adjacent objects, excluding the remaining objects from S.

An object O_i, with the largest number of adjacent objects, forms a β_0-connected set with its neighbors and has more possibilities to be β_0-similar than other new incoming objects. That is why we calculate these objects.

The SCS algorithm works in the following way:

```
    Input: $G_k=\{O_{k1},...,O_{ks}\}$ ($\beta_0$-connected set), AL (adjacency list of the objects in $G_k$).
    Output: $E_k$ (skeleton of the connected set $G_k$).
1. Repeat for i=1:s
    Tag($O_{ki}$)←"SKELETON"
    Valence($O_{ki}$)←Adjacency_Grade(AL,i)
2. Repeat
    If Valence($O_{ki}$)≥Valence($O_{kj}$), i≠j, and Valence($O_{ki}$)>0,
    and Tag($O_{ki}$)←"SKELETON" then
        Tag($O_{kp}$)←"NOT_SKELETON", $O_{kp}$ adjacent to $O_{ki}$
    Until Valence($O_{ki}$)≤0
3. If Tag($O_{kq}$)←"NOT_SKELETON", q=1,...,s then Delete($G_k$,$O_{kq}$) else $E_k \leftarrow E_k \cup \{O_{kq}\}$
```

RUNTIME COMPLEXITY OF GLC, DGLC AND GLC+

In the worst case, the runtime complexity of the GLC and DGLC algorithms is quadratic. And, in the best case, the runtime complexity of these algorithms is lineal with respect to the number of objects. For the GLC+ method, the runtime complexity is lineal, with respect to the number of objects.

In general, the GLC and the DGLC algorithms do not use any index structure support because they handled quantitative and qualitative attributes simultaneously, defined not necessarily in a metric space. The concept of index structure to handle spatial databases, numerical data sets and log databases (Sander, et al., 1998; Ciaccia et al., 1997) is defined in metric spaces. This important tool should be extended to the case of mixed incomplete data. To do the same as GDBSCAN and IncDBSCAN, the runtime complexity of GLC and DGLC without index structure support is quadratic.

PERFORMANCE OF THE ALGORITHMS

In this section, two examples of applications of the GLC algorithm to data sets are presented. The first data set shows graphically an example where this method could be applied (synthetic data). The second data set shows the behavior of GLC running real-life data. Also, we used a data set in order to show the fulfillment in time of GLC when it is applied to a large mixed real-life data set.

Synthetic Data Sets

We carried out a 2D projection (a LDS) of the Covtype data set (Blackard, Dean & Anderson, 1998). This 2D projection was made taking the attributes X_6 and X_8 of the Covtype data set. This 2D data set is shown in Figure 2(a). Figures 2(b) and 2(c) show the clusters generated by a traditional hierarchical algorithm and CLARANS, respectively. The clusters discovered by GLC, DGLC and GLC+ are shown in Figures 2(d), 2(e) and 2(f), respectively. Points in different clusters are represented by using several shades. In Figures 2(d), 2(e) and 2(f), we present the low cardinal clusters using the same shade, because we consider these clusters to be noisy.

Comparison of the Results Obtained of Running the 2D Projection

In this same figure, we show that the 2D projection contains one cluster with high density and cardinality, formed by the majority of the objects in the data set. The remaining clusters have low cardinality (i.e., they are formed by one, two, three, etc. objects). In this sense, the clusters discovered by GLC, DGLC and GLC+ are genuine, because these techniques generate one cluster with high density and cardinality, and the remaining clusters obtained by GLC, DGLC and GLC+, have low cardinality. The traditional hierarchical algorithm and CLARANS obtain four clusters, with high density and cardinality. These algorithms split (into four clusters) the cluster with high density and cardinality. In this sense, the clustering generated by GLC, DGLC and GLC+ is better than the clustering obtained by the traditional hierarchical algorithm and CLARANS.

Real-Life Data Sets

The second data sets shown is a real-life database denominated "Mushroom" (Aha, 1988).

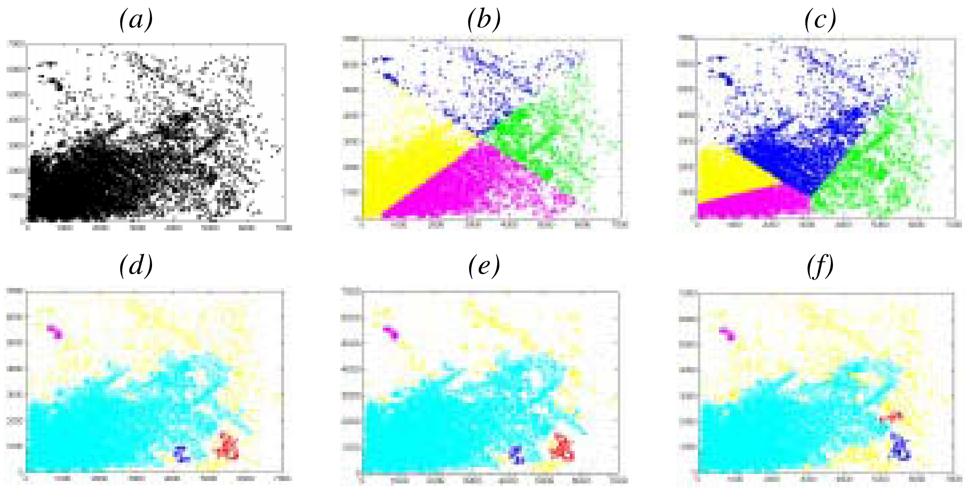
The mushroom data set (a LDS, according to our definitions) contains records with information that describes the physical characteristics of a single mushroom (e.g., color, odor, shape, etc.). This data set contains 8,124 records. All attributes are categorical, and contain missing values. Each record also contains a poisonous or edible label for the mushroom.

Table 1 contains the results of running the Mushroom data set, on the hierarchical algorithm and the ROCK algorithm. Table 2 contains the result of processing the Mushroom data set, on the GLC, DGLC algorithms, and GLC+ method. The notation handled in Tables 1 and 2 is as follows: CN denotes the cluster number; NE indicates the number of edible mushrooms, and, NP denotes the number of poisonous mushrooms.

Comparison of the Results Obtained of Running the Mushroom Data Set

Table 3 illustrates the result of clustering the mushroom database using the traditional hierarchical algorithm and the ROCK algorithm (Guha, et al.). For this example, the quality of the clusters generated by the traditional hierarchical algorithm was very poor. Table 3 indicates also that all except one (Cluster 15) of the clusters discovered by the ROCK algorithm are pure clusters, in the sense that mushrooms in every cluster were either all poisonous or all edible. In the cluster (15)

Figure 2. (a) Original data set; (b) clusters generated by a traditional hierarchical algorithm; (c) clustering obtained by CLARANS; (d) clusters discovered by GLC algorithm; (e) clusters generated by DGLC algorithm; and (f) clustering obtained by GLC+ method.



for poisonous mushrooms found by the ROCK algorithm, around 30% of the members are edible.

In this experimental result, all the clusters generated by all these techniques contain either all poisonous or all edible. Thus, the clusters discovered by GLC, DGLC, and GLC+ are, unlike ROCK, pure clusters, in the same sense previously mentioned.

EXECUTION TIME OF GLC, DGLC AND GLC+

We show the behavior of the run time of GLC with two mixed-incomplete-data sets. The first data set Adult was obtained from Kohavi and Becker (1996). This database contains records of the United States Census in 1994. This data set contains 32,561 records. The attributes are nominal and numerical simultaneously (age, education, capital-gain, etc.), with 7% of missing values. The Adult database is a

Table 1: Clustering result for mushroom data

Traditional Hierarchical Algorithm						ROCK Algorithm					
CN	NE	NP	CN	NE	NP	CN	NE	NP	CN	NE	NP
1	666	478	11	120	144	1	96	0	12	48	0
2	283	318	12	128	140	2	0	256	13	0	288
3	201	188	13	144	163	3	704	0	14	192	0
4	164	227	14	198	163	4	96	0	15	32	72
5	194	125	15	131	211	5	768	0	16	0	1728
6	207	150	16	201	156	6	0	192	17	288	0
7	233	238	17	151	140	7	1728	0	18	0	8
8	181	139	18	190	122	8	0	32	19	192	0
9	135	78	19	175	150	9	0	1296	20	16	0
10	172	217	20	168	206	10	0	8	21	0	36
						11	48	0			

Table 2: Clusters discovered by GLC, DGLC and GLC+, for mushroom data

GLC, DGLC algorithms, and GLC+ method											
CN	NE	NP	CN	NE	NP	CN	NE	NP	CN	NE	NP
1	0	512	7	0	192	13	0	16	19	72	0
2	0	768	8	0	48	14	256	0	20	32	0
3	0	96	9	0	48	15	1296	0	21	8	0
4	0	96	10	0	192	16	192	0	22	36	0
5	0	192	11	0	288	17	288	0	23	8	0
6	0	1728	12	0	32	18	1728	0			

Large Data Set (LDS), because the similarity matrix exceeds the available memory size, but the object descriptions do not exceed the available memory size.

The second data set is named covtype. It was obtained from Blackard et al. (1998), and contains records from cartographic data for forest prognosis. This database has 581,012 instances, without missing values. There are numerical and Boolean features simultaneously (hill, aspect, hydrological-horizontal-distance).

The Covtype database is a *Very Large Data Set (VLDS)*, because both the similarity matrix size and the objects descriptions size exceed the available memory size (i.e., the size of the objects descriptions is approximately 76 Megabytes).

The experiments were implemented in C language on a personal computer with the Pentium processor running at 350 mHz and 64 megabytes of RAM.

Figures 3(a) and 3(b) show the behavior of the runtime required by the GLC and DGLC algorithms respectively, in order to create the clusters of the Adult and Covtype databases. In addition, in Figures 3(a) and 3(b), are shown the run times required to calculate the similarity matrix of the objects by a traditional algorithm described in Martínez-Trinidad et al. (2000a). Figure 3(c) shows the behavior of the runtime required by GLC+ method in order to create the clusters of the Adult and Covtype databases. Finally, Figure 3(d) shows the runtime of the traditional algorithm, GLC, DGLC, and GLC+ techniques.

FUTURE TRENDS

Starting from the achieved results, the next steps in this area should be: The development of new methods based on other clustering criteria such as compact sets, strictly compact sets, among others (Martínez-Trinidad et al., 2000): the development of procedures to resolve the above mentioned problems, but in the case of unsupervised restricted classification, that is, when the number of clusters to be obtained is given.

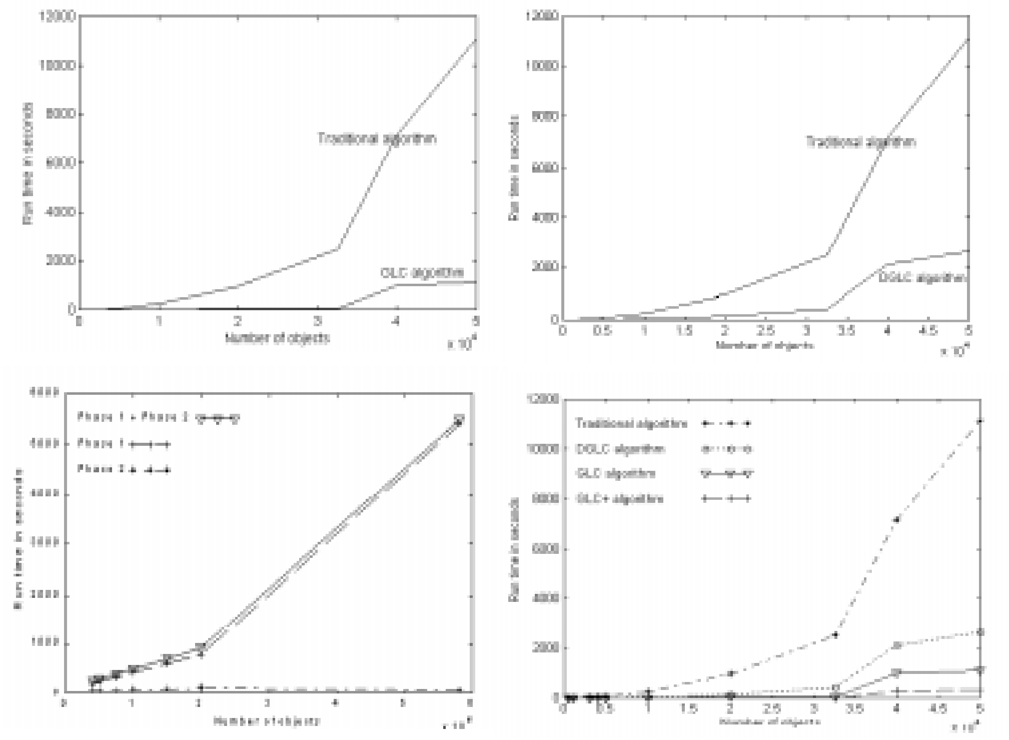
The development of methods and procedures based on fuzzy clustering criteria like fuzzy connected components, fuzzy compact sets; fuzzy strictly compact sets, among others (Martínez-Trinidad et al., 2000a).

Given the practical importance of the introduction to the index structure used in algorithms like DBSCAN, IncDBSCAN, and BIRCH, it is necessary to extend this idea to the mixed incomplete data case. Finally, it is worth mentioning the possibilities of all these results in the image processing and analysis fields. These are only some of the most important objectives that will be considered in the near future.

CONCLUSION

The existence of many mixed incomplete databases in different areas of real-world problems is a big motivation for the development of new methods for MID processing. In the cases of the three techniques discussed above we can conclude the

Figure 3. (a) Graphic objects-time required by GLC and a traditional algorithm; (b) behavior of the run time required by the DGLC algorithm; (c) run time growth of the GLC+ method; and (d) graphic objects-time required by the traditional algorithm, GLC, DGLC and GLC+ techniques.



following

The objects in the data set to be clustered may be described by quantitative and qualitative mixed attributes and with the presence of missing data. This eliminates the limitation that hinders some classical algorithms, which are capable of handling only one type of attribute. This improvement means an important step in order to solve practical real problems especially in the soft sciences, in which the descriptions of objects are almost never homogeneous in the sense of the kind of feature descriptions.

The GLC algorithm allows the generation of β_0-connected components from large mixed incomplete data sets. DGLC is a clustering algorithm, which works with large mixed incomplete data sets based on the density cluster notion introduced in (Ester et al., 1996, 1998). The GLC+ method allows the generation of β_0-connected sets from very large mixed incomplete data sets. GLC, DGLC and GLC+ do not establish any assumptions about the form, the size or cluster density characteristics of the resultant clusters. These algorithms could use any kind of distance, or symmetric similarity function. Nevertheless, these techniques have some restrictions. Indeed, they are still susceptible to outliers and artifacts.

REFERENCES

Aha, D.W. (1988). Mushroom database. [Internet] *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/mushroom.*

Alba-Cabrera, E. (1998). New extensions of the testor concept for different types of similarity functions. *PhD dissertation, Instituto de Cibernética, Matemática y Física, Cuba.*

Blackard, J.A., Dean, D.J., & Anderson, C.W. (1998). The Forest CoverType dataset. *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/cover-type.*

Ciaccia, P., Patella, M., & Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. *Proceedings of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, 426-435.

Dmitriev, A.N., Zhuravlev, Yu.I., & Krendelev, F.P. (1966). About the mathematical principles of objects and phenomena classification. *Sbornik Diskretnii Analisis 7*, 3-15. Novosibirsk.

Ester, M., Kriegel, H., Sander, J., Wimmer, M., & Xu, X. (1998). Incremental clustering for mining in a data warehousing environment. *Proceedings of the 24th VLDB*, New York, 323-333.

Ester, M., Kriegel, H.P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis, J. Han, U. Fayad (Eds.), *Proceedings of the Second International Conference on Knowledge Discovery & Data Mining*. 226-231.

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. (1996). *Advances in Knowledge Discovery in Databases.* Cambridge, MIT Press.

Goldfarb, L. (1985). A new approach to pattern recognition. In L. Kanal, & A. Rosenfeld (Eds.) *Progress in Machine Intelligence & Pattern Recognition.* vol. II

Gómez-Herrera, J.E., et al. (1994). Prognostic of gas-petroleum in the Cuban ophiolitic rocks association, applying mathematical modeling, *Geofísica Internacional, 33* (3), 447-467.

Guha, S., Rastogi, R. & Shim, K. (1999). ROCK: A robust clustering algorithm for categorical attributes. *Proceedings of the 15th International Conference on Data Engineering*, California, USA, 512-521.

Jain, K., & Dubes, R.C. (1998). *Algorithms for Clustering Data, Prentice Hall.*

Karypis, G., (Sam) Han, E.H., & Kumar, V. (1999). Chameleon: Hierarchical cluster using dynamic modeling. *Computer*, 68-75.

Kohavi, R. & Becker, B. (1996). Adult database. *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult.*

Kohonen T. (1986). *Learning Vector Quantization.* Helsinki University of Technology, Laboratory of Computer and Information Science, Report TKK-F-A-601.

Kohonen T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics, 43*, 59-69.

Martínez-Trinidad, J.F., Ruiz-Shulcloper, J., & Lazo-Cortés, M. (2000a). Structuralization of universes. *Fuzzy Sets & Systems, 112/3*, 485-500.

Martínez-Trinidad, J.F., Velasco-Sánchez, M., & Contreras-Arévalo, E. E. (2000b). Discovering differences in patients with uveitis through typical testors by class. In D.A. Zighed, J. Komorowski, & J. Zytkow. (Eds.), *Proceedings of the 4th European Confer-*

ence on Principles and Practice of Knowledge Discovery in Databases. Lecture notes in artificial intelligence 1910.

Ng, R., & Han, J. (1994). Efficient and effective clustering methods for spatial data mining. *Proceedings of the 20th Int. Conf. on VLDB*, Santiago, Chile, 144-155.

Ortíz-Posadas, M., Martínez-Trinidad, J.F., & Ruiz-Shulcloper, J. (1996). A new approach to differential diagnosis of diseases. *International Journal of Bio-medical Computing, 40*, 179-185.

Ruiz-Shulcloper, J., & Montellano-Ballesteros, J.J. (1995). A new model of fuzzy clustering algorithms. *Proceedings of the Third European Congress on Fuzzy and Intelligent Technologies and Soft Computing* (Aachen, Germany, 1995), 1484-1488.

Ruiz-Shulcloper, J., Alba-Cabrera, E., & Sánchez-Díaz, G. (2000). DGLC: A density-based global logical combinatorial clustering algorithm for large mixed incomplete data. [CD-ROM]. *Proceedings of the IGARSS 2000*, Honolulu, Hawaii.

Sánchez-Díaz, G., & Ruiz-Shulcloper, J. (2000). MID mining: a logical combinatorial pattern recognition approach to clustering in large data sets. Proceedings of the 5th Iberoamerican Symposium on Pattern Recognition, Lisbon, Portugal, September 475-483.

Sánchez-Díaz, G., Ruiz-Shulcloper, J. (2001). A new clustering method for very large mixed data sets. *PKDD'2001.*

Sander J., Ester, M., Kriegel, H., & Xu, X. (1998). Density-based clustering in spatial databases: the algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery an International Journal, 2(2),* 169-194,.

Schlimmer, J. (1987). 1984 United States congressional voting records database. *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/voting-records.*

Zhang, T., Ramakrishnan, R. & Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases. *Proceedings ACM SIGMOD International Conference on Management of Data*, ASM Press, 103-114.

SECTION THREE

STATISTICS AND DATA MINING

Chapter VII

Bayesian Learning

Paula Macrossan
University of New England, Australia

Kerrie Mengersen
University of Newcastle, Australia

Learning from the Bayesian perspective can be described simply as the modification of opinion based on experience. This is in contrast to the Classical or "frequentist" approach that begins with no prior opinion, and inferences are based strictly on information obtained from a random sample selected from the population. An Internet search will quickly provide evidence of the growing popularity of Bayesian methods for data mining in a plethora of subject areas, from agriculture to genetics, engineering, and finance, to name a few. However, despite acknowledged advantages of the Bayesian approach, it is not yet routinely used as a tool for knowledge development. This is, in part, due to a lack of awareness of the language, mechanisms and interpretation inherent in Bayesian modeling, particularly for those trained under a foreign paradigm. The aim of this chapter is to provide a gentle introduction to the topic from the KDD perspective. The concepts involved in Bayes' Theorem are introduced and reinforced through the application of the Bayesian framework to three traditional statistical and/or machine learning examples: a simple probability experiment involving coin tossing, Bayesian linear regression and Bayesian neural network learning. Some of the problems associated with the practical aspects of the implementation of Bayesian learning are then detailed, and various software freely available on the Internet is introduced. The advantages of the Bayesian approach to learning and inference, its impact on diverse scientific fields and its present applications are identified.

INTRODUCTION

Learning from the Bayesian perspective can be described very simply as the modification of opinion based on experience. The Bayesian approach to learning combines a prior subjective opinion with new information, in the form of data, to develop a revised opinion. This process can be repeated any number of times and occurs in a cyclical fashion, with the revised opinion becoming the prior opinion with the arrival of new data. This is in contrast to the classical, traditional or "frequentist" approach that begins with no prior opinion, and inferences are based strictly on information obtained from the data.

This chapter is intended to provide a gentle introduction of Bayesian methods by beginning with simple examples and explanations and later reviewing more complex applications in data mining. The concepts involved in Bayes' Theorem are established with the example case of a simple coin-tossing experiment, after which the typical hierarchical methods used in Bayesian modeling are described. Graphical models, in which the essential structures implicit in a model are defined, are introduced as a convenient method of breaking down a complex model into simple components that provide the basis for Bayesian computation. Bayesian linear regression is examined to illustrate the Bayesian approach to a well-known statistical technique. The application of Bayesian inference to hybrid methods of machine learning is illustrated with the discussion of Bayesian neural network learning. Some of the problems and pitfalls associated with the practical implementation of the Bayesian framework are discussed, after which the reader is then introduced to powerful software for Bayesian inference and diagnostics freely available on the Internet. In conclusion, the present and predicted future impact of the Bayesian approach to learning and inference on diverse scientific fields is discussed. Some texts are recommended for further reading.

BACKGROUND

The essential characteristic of Bayesian methods is their explicit use of probability for quantifying uncertainty of parameter estimates of interest in scientific analysis. Bayesian statistics are used to analyse the plausibility of alternative hypotheses that are represented by probabilities, and inference is performed by evaluating these probabilities and making consequent decisions or estimates.

There has been a long-standing debate between Bayesian and classical statisticians regarding the approach to data analysis and knowledge acquisition. The interested reader can pursue this largely philosophical discussion by referring to Press and Tanur (2001). Recently there has been a revival of interest, fuelled by pragmatic rather than philosophical concerns, in the Bayesian approach in all areas of scientific research. It is no coincidence that this revival has occurred in parallel with the burgeoning of computer technology, allowing computationally intensive algorithms to be implemented easily and inexpensively. The most far-reaching of these algorithms is the Markov Chain Monte Carlo (MCMC) approach for simulating draws from complex probability distributions (Besag, Green, Higdon & Mengersen, 1995).

Bayes' Theorem

We will begin our discussion with a description of the foundation of the Bayesian approach, namely Bayes' Theorem. Assume there are a number of plausible models, $H_1, H_2, ..., H_m$, which might account for the data to be gathered, where each model proposes a possible set of parameter values or model structure of interest. Our initial beliefs about the relative plausibility of these parameters or models can be quantified by assigning prior probabilities to each model, $P(H_1), P(H_2), ..., P(H_m)$, which must sum to one. Each model H_i can be used to make predictions in the form of a probability distribution, $P(D|H_i)$, about the probability of a dataset D if H_i is true. Bayes' Theorem then describes how these prior beliefs should be updated in view of the evidence given by the data. This posterior probability $P(H_i|D)$ is thus the plausibility of the model H_i given the observed data D, and is calculated by multiplying together two quantities: firstly $P(H_i)$, how plausible we believed the model to be before the data arrived; and secondly $P(D|H_i)$, how much the model predicted the data (MacKay, 1992). Equation 1 gives both a symbolic and informal expression of Bayes' Theorem.

$$P(H_i \mid D) = \frac{P(H_i)P(D \mid H_i)}{P(D)} \text{ i.e., } Posterior = \frac{prior \times likelihood}{evidence} \qquad (1)$$

The evidence, $P(D)$, which is the denominator of Equation 1, is the prior probability of the data, and serves as a normalising constant that makes the posterior probabilities, $P(H_i|D)$, add to one. In the discrete case, such as the coin-tossing experiment discussed in the following section, the evidence $P(D)$ can be summed. In the continuous case, $P(D)$ must be integrated. It is in the process of integration that the practical application of Bayes' Theorem may become intractable in all but the most simple cases, and simulation techniques such as MCMC are required to approximate the posterior distribution.

A Simple Example of Bayes' Theorem in Practice–A Coin-Tossing Experiment

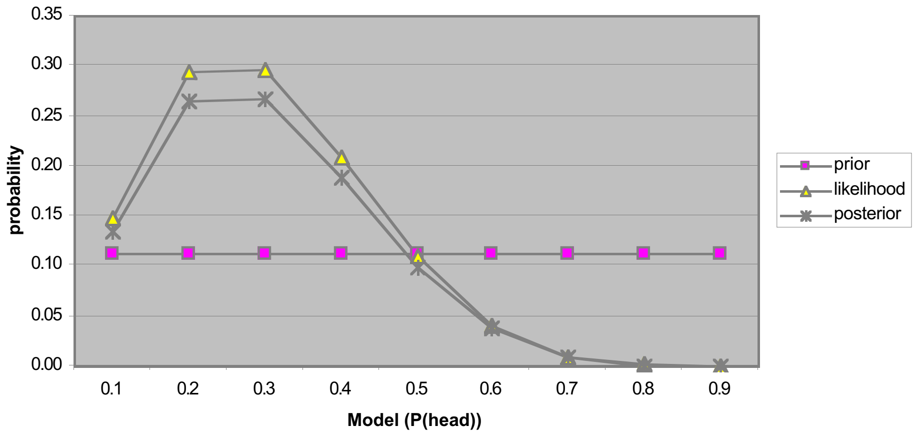
Since the intention of this chapter is to illustrate the power of the Bayesian approach as a tool for integrating knowledge, we detail here a simple example from which complex applications follow. Consider the illustration of guessing the outcome of a coin-tossing experiment. We could formalise this by saying that we wish to estimate the true probability of a head, i.e., $P(H)=\theta$, using our prior beliefs and some observed data. The discussion below is reflected in Table 1, which shows the various calculations made. Firstly, we need to formalise our prior belief. If there is prior information or belief that the coin is biased and more likely to land heads, a prior probability distribution can be modelled that assigns a higher probability to this latter event. Assume the case where there is no prior knowledge of the coin, except that the coin may be biased either way, and we wish to investigate nine different models, H_i, for the probability of throwing a head. Here, H_i represents the model that *P(Head)* = 0.1, H_2 represents the model that *P(Head)* = 0.2, and finally to H_9 which represents the model that *P(Head)* = 0.9. In this case the fairness of the coin may be represented as a uniform prior, with the prior probability partitioned among the nine models, so that each model is assigned a prior probability of 1/9. The next issue in Bayesian modelling concerns our data set. Suppose that the coin is tossed eight times resulting in the data D= (THTTHTTT), or two heads in a total of eight tosses. This then leads to the question of the likelihood of each model, $P(D|H_i)$, which is the probability of seeing the data D if that model H_i is correct. For example, under the first model H_i, *P(Head)* = 0.1, and the likelihood of seeing two heads from eight tosses is $8C_2$ x $(0.1)^2$ x $(0.9)^6$ = 0.1487. Similarly under the second model H_2, *P(Head)* = 0.2, and the likelihood is $8C_2$ x $(0.2)^2$ x $(0.8)^6$ = 0.2937, and so on for each model. Next, how do we calculate *P(D)*, the prior probability of the data? This is the sum of the likelihoods over all models. Armed with all the above calculations, the posterior probability for each model can be calculated using Equation 1, as shown in Table 1.

How is this posterior probability influenced by the prior probabilities and the data? Notice that the model with the largest posterior probability is H_3:*P(Head)* = 0.3, closely followed byH_2:*P(Head)* = 0.2, in accordance with the data that was

Table 1: The calculation of the likelihood and posterior probabilities for nine alternative models in the coin tossing experiment given a uniform prior probability for each model of 1/9 (0.11)

Proposed Model	Prior Probability	Likelihood	Prior x Likelihood	Posterior Probability
H_i:*P(Head)*	$P(H_i)$	$P(D\|H_i)$	$P(H_i).P(D\|H_i)$	$P(H_i\|D)$
H_1:0.1	0.11	0.1488	0.0165	0.1341
H_2:0.2	0.11	0.2936	0.0326	0.2646
H_3:0.3	0.11	0.2965	0.0329	0.2672
H_4:0.4	0.11	0.2090	0.0232	0.1883
H_5:0.5	0.11	0.1094	0.0121	0.0986
H_6:0.6	0.11	0.0413	0.0046	0.0372
H_7:0.7	0.11	0.0100	0.0011	0.0090
H_8:0.8	0.11	0.0012	0.0001	0.0010
H_9:0.9	0.11	0.0000	0.0000	0.0000
Sum	1.00		P(D)=0.1233	1.0000

Figure 1: The posterior probabilities for nine alternative models in the coin tossing experiment, based on the likelihood and a uniform prior probability for each model of 0.11



observed, since the prior probability does not lend much information. Figure 1 shows graphically the result of combining the prior with the likelihood function, leading to a posterior distribution that is concentrated in the vicinity of the value corresponding to the observed frequency of heads. Note that this convergence to the result anticipated from the data may be modified by the prior. As an extreme case, if the prior belief were that *P(Head)* =1, then no amount of data to the contrary can sway this opinion. In a less extreme scenario, if we believed the coin to be possibly biased towards heads, we could assign small prior probabilities for H_1 through H_5 (say, 0.04 each), describe H_6 and H_7 as much more likely (with priors of 0.3 each) and attach 10% probability to each of H_8 and H_9. In this case, the integration of this prior information with the data (two heads out of eight tosses) leads to posterior probabilities for H_1 through H_9 of 0.10, 0.20, 0.20, 0.14, 0.08, 0.21, 0.05, 0.00 and 0.00, respectively. Notice that the posterior probabilities for H_2 and H_6 are both similar and relatively large; in the case of H_2 this is due to a relatively large likelihood for the model, whilst in the case of H_6 it is the result of a relatively large prior probability on this model.

Prior and Posterior Probabilities

The prior probability in Bayesian statistics is used as a means of quantifying uncertainty about unknown quantities or model parameters. It is generally the case that the parameters being estimated have a precise meaning in the problem under study, and it is therefore likely that the researcher (or expert opinion) has some knowledge about their true values. The prior may therefore be a subjective probability that describes the researcher's personal view of what values the model parameters might have, and it is at this point that the Bayesian and classical approaches diverge. For a discussion on the aptness of incorporating this historic information into the analysis, see Efron (1986).

The concepts of prior and posterior are relative to the data being considered at the time. If a new toss of the coin y becomes available after observing the data D and obtaining a posterior, the posterior relative to D becomes the prior relative to y, and a new posterior is obtained through a re-application of Bayes' Theorem. This updating process can be continuously applied so that inference is a dynamic process in which new data are used to revise the current knowledge.

Conjugacy of Prior and Posterior Distributions

When the posterior distribution derives from the same family as the prior, the distributions are termed conjugate and the integration can be performed analytically. However, in the case of non-conjugacy, analytical integration is impossible. Non-conjugacy was an insuperable problem in Bayesian learning prior to the accessibility of computers and computer-intensive approximation methods such as MCMC (see later section).

BAYESIAN HIERARCHICAL MODELLING

Hierarchical models are traditionally used to describe the structure inherent in the experimental model. For example, the allocation of similar individuals or observations into groups is commonly known as *blocking*, and an agricultural experiment could be made up of three levels in a hierarchy, the upper level being farms, the middle level being blocks within farms, and the lower level being plots within blocks. Hierarchical models are extended in Bayesian analysis to describe levels of prior specification.

Consider an alternative prior for the coin-tossing experiment described earlier. As before, the likelihood of each model, $P(D|H_i)$ is modelled as a binomial distribution $r \sim Binomial(n, \theta)$ with r successes (or heads) in n independent trials (or eight tosses of the coin), which here becomes the first level in the hierarchy. Now θ is the probability of throwing a head, and in place of the discrete prior distributions for the H_i defined previously, we can instead assign continuous prior probabilities for θ using a $Beta(\alpha, \beta)$ distribution, which is defined over the entire range of 0 to 1. Manipulation of the values of α and β allow, this distribution to be highly skewed towards either end of the range, reflecting strong prior belief in the bias of the coin, or be fairly symmetric around 0.5. This prior introduces a second level in the hierarchy. Now if the parameters α and β are not able to be specified they may be assigned a vague (but proper) exponential prior, say, *Exponential(0.01)*, which is the third level in the hierarchy. Bayes' Theorem is applied simultaneously to the joint probability model for the entire set of parameters (θ, α, β) to obtain a joint posterior distribution (see Gilks, Richardson & Spiegelhalter, 1996).

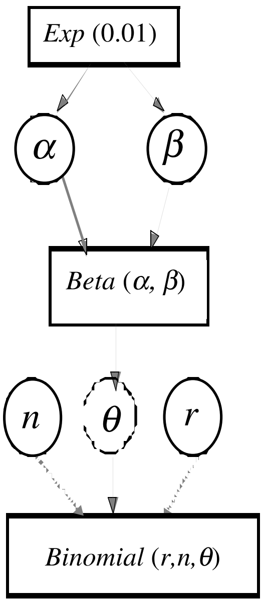
BAYESIAN GRAPHICAL MODELLING

Graphical models allow the essential structures implicit in a model to be defined, interpreted and communicated, and are a convenient method of breaking down a complex model into the simple components that provide the basis for Bayesian computation. They are an extension of the concept of the directed acyclic graph (DAG) built from a sequence of conditional independence assumptions, allowing factorization of the joint distribution. Figure 2 shows a graphical model for the coin-tossing example under the hierarchical model, in which circular shapes depict variables (data or parameters), boxes surround probability distributions, dashed arrows define deterministic relationships and solid arrows show stochastic relationships.

MARKOV CHAIN MONTE CARLO (MCMC) METHODS

As discussed earlier, when the prior and posterior distributions are non-conjugate or the quantity of interest is nonstandard (which represents the majority of cases in practice), the integration cannot be performed analytically and approximate solutions must suffice. This provides the motivation for MCMC methods in which parameter values are repeatedly drawn from their posterior distributions. The resultant sequence of values is a Markov chain, which has special statistical properties, particularly with regard to the true posterior distribution. The quandary of designing a Markov chain with the posterior distribution as its unique stationary distribution was solved by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953). Gibbs sampling (see Geman & Geman, 1984) is a special case of MCMC that generates a Markov chain by sampling from the full conditional distributions. Besag, Green, Higdon and Mengersen (1995) provide a complete discussion and worked examples of these algorithms.

Figure 2: A graphical model for the coin-tossing example with three hierarchical levels described as likelihood: r~Binomial(n,θ) ; prior on θ~Beta(α,β); prior on α, β~ Exponential(0.01).



THE APPLICATION OF BAYES' THEOREM TO THE MODELLING PROCESS

In order to demonstrate the wide applicability of Bayesian analysis, let us now examine three areas, familiar to both the KDD and statistical communities, that are more complex than the coin-tossing example.

Bayesian Networks

The graphical models described earlier to define the inherent structures in a model can be further extended to a powerful formal knowledge representation widely used in KDD, known as Bayesian Belief Networks (BBNs). BBNs describe associations between sets of variables based on probabilities and Bayes' Theorem, from which predictions about the value of a variable in a given situation can be made by computing the conditional probability distribution of the variable, given the values of a set of some other variables in the network. BBNs are initially constructed from prior knowledge, with Bayesian statistical methods being used to improve the network by learning both the parameters and the structure from the data. Some persuasive arguments, made by scientists working in commercial environments, for using BBNs over other KDD technologies such as traditional rule-based expert systems and neural networks can be viewed at (www.auai.org/BN-Testimonial).

Bayesian Linear Regression (BLR)

Under the classical framework, multiple linear regression is performed using the method of least squares or maximum likelihood. Under the Bayesian approach, posterior probabilities and expected values of the regression parameters are obtained by learning from both prior information and the data. In contrast to the coin-tossing experiment, the prior probabilities for the response and predictor variables in this example will be modelled using the common Normal distribution, a bell-shaped continuous distribution defined over the whole real line. The analytical approach taken is as follows. A linear regression model with *j* observations and *m* predictor variables $x_1, x_2, ..., x_m$ could be described by Equation 3, with α constant, the $B_i(i=1,...,m)$ representing the coefficients of the predictor variables of interest, and e_j representing the error or difference between the predicted and observed values.

$$y_j=\mu_j+e_j=\alpha+\beta_1 x_{1j} + \beta_2 x_{2j}+......+\beta_m x_{mj}+e_j \tag{3}$$

A Bayesian approach might model the response variable $y_j \sim \mathrm{N}(\mu_j,\sigma)$ so that $\hat{y}_j = \mu_j$ is the predicted response and $e_j \sim \mathrm{N}(0,\sigma)$. The little information known about the regression parameters can be reflected through the priors $\alpha \sim \mathrm{N}(0,100)$, $\beta_i \sim \mathrm{N}(0,100)$ and $1/\sigma^2 \sim Gamma(0.001, 0.001)$. MCMC methods are then used to carry out the necessary numerical integrations using simulation to obtain estimates of α and the β_i. Other quantities of interest may be, for example, the posterior probability that $\beta_i>0$ (i.e., a positive relationship between *y* and x_i) or predictive probabilities for unseen or missing observations.

Bayesian Neural Networks

Bayesian nonlinear regression can be performed using the hybrid machine learning approach of a Bayesian neural network (BNN), where a probabilistic interpretation is applied to the traditional ANN technique (Neal, 1996). In the Bayesian approach to ANN prediction, the objective is to use the training set of inputs and targets to calculate the predictive *distribution* for the target values in a new "test" case, given the inputs for that case. The Bayesian framework allows the objective evaluation of a number of issues involved in complex modelling including network architecture and the effective number of parameters used.

In a typical ANN with one hidden layer and one output unit, the output or target (y_n) might be computed from the inputs (x_i) as in Equation 4, where u_{ij} is the weight on the connection from input value i to hidden unit j; v_j is the weight on the connection from hidden unit j to the single output unit and the a_j and b are the biases of the hidden and output units respectively. These weights and biases are the parameters of the network. Thus, each output value, y_n, is a weighted sum of hidden unit values plus a bias. Each hidden unit computes a similar weighted sum of input values and then passes it through a nonlinear activation function.

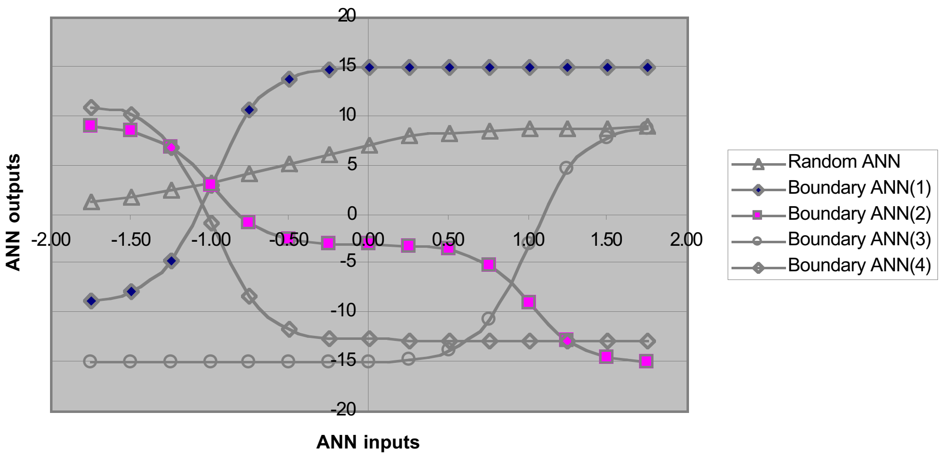
$$y_n = b_n + \sum_j v_j h_j(\tilde{x}_n) \;\; ; \text{ where } \; h_j(\tilde{x}_n) = \tanh(a_j + \sum_i u_{ij} x_{ni}) \quad (4)$$

Figure 3 provides an illustration of Bayesian inference for the ANN learning of a simple nonlinear regression model using Equation 4. The ANN architecture used consisted of a single input and output unit, representing a single predictor and response variable, and a single layer of 16 hidden units. The hidden weights and hidden and output biases were drawn at random from a N(0,1) distribution, and the output weights were drawn from a N(0,0.25) distribution. The functions were computed using the small data set consisting of six data points shown in Table 2. Figure 3 shows one such "random" ANN function computed. For each data point a probability is estimated that this calculated value came from a normal distribution with a mean of the response value and a standard deviation of 0.1 (describing the network "noise"). These six individual probabilities are combined (as in the coin-tossing experiment) to give a likelihood estimate for the model ANN. Large numbers of ANNs can be computed and compared through their likelihood functions to give the ANN with the maximum likelihood. Additionally, Figure 3 demonstrates the range of functions capable of being represented by the ANN. The four examples labelled "boundary" ANNs represent networks whose weights and biases were artificially assigned boundary or extreme values from the appropriate normal distribution (e.g., boundary values from a N(0,1) distribution might be ±3), as opposed to making random draws from the distribution. Using different combinations of such boundary values, these "boundary" ANNs can theoretically compute any function within the limitations set by the data and the chosen prior

Table 2: Predictor and associated response variables for the small data set used in the illustration of ANN learning

Predictor Variable	Response Variable
-0.85	2.10
-0.55	2.60
-0.45	3.55
1.10	4.21
1.20	4.25
1.30	4.41

Figure 3: An illustration of Bayesian inference for an ANN for a small data set of six input values. "Random" refers to an ANN with weights and biases being drawn from Normal distributions. "Boundary" refers to an ANN with extreme values from the Normal distributions used for the weights and biases.



distributions. The illustration could be taken a step further by allowing components of the ANN model architecture, such as the number of hidden units and layers, to be also drawn from sample distributions.

Although the above method is useful for illustration purposes, obviously a more efficient algorithm, namely, MCMC, is required to find the optimal set of weights and biases in the BNN to compute any useful function with a realistically sized data set. Some practical examples of the use of BNNs in KDD are Ma, Wang and Wu (2000), who demonstrate biological data mining using BNNs to carry out sequencing classification in DNA, and Macrossan, Abbass, Mengersen, Towsey and Finn (1999), who demonstrate the use of BNN learning in a data mining application concerning prediction of milk production in the Australian dairy industry.

PRACTICAL ISSUES IN THE IMPLEMENTATION OF BAYESIAN LEARNING

An area still to be addressed in Bayesian inference is the need for standards for analysis and reporting (Best, Marshall & Thomas, 2000). The results of Bayesian inference need to be justified to an outside world such as reviewers, regulatory bodies and the public, and such standards would assist in this regard. Additionally, in the public reporting of the results of Bayesian analysis, the assignment of prior probabilities must be explained and justified. Kass and Greenhouse (1989) advocate what they term "a community of priors" which undergo a sensitivity analysis to assess whether the current results will be convincing or otherwise to a broad spectrum of opinion. This community of priors includes "reference", "clinical", "sceptical" and "enthusiastic" priors. In the coin-tossing example given earlier, a uniform or reference prior was used to reproduce the maximum likelihood results as opposed to a clinical prior based on genuine clinical or exact opinion (e.g., *P(Head)* can only be greater than 0.5) or sceptical and enthusiastic priors (i.e., opinions can be either strongly or weakly biased).

Convergence of the MCMC algorithm to the target distribution is a problem that Bayesian computation has in common with many other numerical analysis methods. The advantage of MCMC is that it has mathematical results from Markov chain theory to verify the algorithms. However, convergence checking requires considerable care and should be validated both visually and through diagnostics (see next section).

SOFTWARE FOR BAYESIAN INFERENCE

First Bayes is a teaching package for elementary Bayesian statistics, available freely on the Internet (http://www.shef.ac.uk/~st1ao/1b.html). The emphasis is on obtaining an understanding of how the Bayesian approach works, using simple and standard statistical models. Binomial, gamma, Poisson and normal data may be analysed using an arbitrary mixture of distributions from the conjugate family, allowing the user to obtain an understanding of how the likelihood and prior distributions are combined using Bayes' Theorem to give a posterior distribution.

Bayesian inference using Gibbs sampling (WinBUGS) (Spiegelhalter, Thomas, Best & Gilks, 1995) assumes a full probability model in which all quantities, observed (the data) and unobserved (parameters and missing data), are treated as random variables. WinBUGS incorporates DoodleBUGS, a graphical interface for model specification, whilst GeoBUGS provides spatial modelling. All BUGS software may be freely obtained from the Internet site (http://www.mrc-bsu.cam.ac.uk/bugs).

Convergence diagnostics of BUGS output can be performed using CODA (Convergence Diagnostics and Output Analysis; Best, Cowles and Vines, 1995), which is written and maintained by the BUGS research team. Additionally, the convergence diagnostics of Raferty and Lewis (1992), *gibbsit*, and Gelman and Rubin (1992), *itsim*, are available from their respective authors at the WWW site (http://lib.stat.cmu.edu).

Neal (*www.cs.utoronto.ca/~radford/*) provides software that implements Bayesian models based on artificial neural networks, Gaussian processes and mixtures using MCMC methods, which is freely available on the Internet site.

AutoClass is an unsupervised Bayesian classification system that seeks a maximum posterior probability classification. AutoClass C is the public domain version available at *ic-www.arc.nasa.gov/ic/projects/bayes-group/group/autoclass*.

DISCUSSION AND CONCLUSIONS

Data mining is a burgeoning area of scientific interest involving the analysis of patterns in large volumes of raw data by integrating methods from such diverse fields as machine learning, statistics, pattern recognition, artificial intelligence and database systems. The Bayesian approach to data mining has a potentially significant role to play whenever complex modelling is required, by providing extremely powerful simulation techniques based in relatively unsophisticated and easily implemented statistical theory. However, this potential role will only be realized if researchers are prepared to embrace the paradigm shift required for achieving an understanding of the mechanisms inherent in Bayesian modeling. This chapter has attempted to provide the impetus for this, by providing a nonthreatening introduction to the Bayesian paradigm from the KDD perspective.

In conclusion, it is fitting to quote from a paper entitled "Quantitative Genetics in the Age of Genomics" (Walsh, in press). With the publicity and budget afforded the Human Genome Project it could be argued that the area of scientific endeavour receiving the most public attention at present and in the foreseeable future is that of genetics. The key to unfolding the mysteries of the human genome lies in the ability to discover patterns and meanings in large volumes of raw data, an impossible task prior to the computational revolution of recent times. This computational and data mining revolution which has been instrumental in ushering in the "Age of Genomics," and which has also been responsible for the development of MCMC methods prompts Walsh to predict a "Bayesian Future" for quantitative genetics. Such a strong prediction is typical of the type of growth being forecast in the popularity of Bayesian techniques in all areas of scientific endeavour.

FURTHER READING

The following introductory statistical texts, delving further into the theory of Bayesian statistics, are recommended to the interested reader: Carlin and Louis (1996), Gilks et al. (1996), Gelman, Carlin, Stern and Rubin (1995), and Gamerman (1997). A useful introductory text that presents Bayesian inference and learning from the KDD perspective is that of Ramoni and Sebastiani (1999). An informal discussion on practical aspects of MCMC methods can be found in Kass, Carlin, Gelman and Neal (1998).

REFERENCES

Besag, J., Green, P., Higdon, D. & Mengersen, K. (1995). Bayesian computation and stochastic systems. *Statistical Science, 10*, 3-66.

Best, N. G., Cowles, M. K., & Vines, S. K. (1995). *CODA: Convergence Diagnostics and Output Analysis Software for Gibbs Sampler Output: Version 0.3.* Technical report, Biostatistics Unit – MRC, Cambridge, U. K.

Best, N. G., Marshall, C., & Thomas, A. (2000). *Bayesian inference using WinBUGS.* Short Course: Brisbane, Nov 28-29, 2000.

Carlin, B. P. & Louis, T. (1996). *Bayes and empirical Bayes methods for data analysis.* London: Chapman & Hall.

Efron, B. (1986). Why isn't everyone a Bayesian? *American Statistician, 40*, 1-11.

Gamerman, D. (1997). *Markov chain Monte Carlo – Stochastic simulation for Bayesian inference*. Texts in Statistical Science. (1st ed.). London: Chapman & Hall.

Gelman, A., & Rubin, D. B. (1992). A single series from the Gibbs sampler provides a false sense of security. In J. M. Bernardo et al. (Eds.), *Bayesian Statistics 4* (pp. 625-631). Oxford: Oxford University Press.

Gelman, A., Carlin, J. B., Stern, H. S. & Rubin, D. B. (1995). *Bayesian Data Analysis.* Texts in Statistical Science. London: Chapman & Hall.

Geman, S. & Geman, D. (1984). Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence, 6*, 721-741.

Gilks, W. R., Richardson, S. & Spiegelhalter, D. J. (1996). *Markov chain Monte Carlo in practice*. London: Chapman & Hall.

Kass, R. E., Carlin, B. P., Gelman, A. & Neal, R. M. (1998). Markov Chain Monte Carlo in practice: A roundtable discussion. *The American Statistician, 52*, 93-100.

Kass, R. E. & Greenhouse, J. B. (1989). Comments on 'Investigating therapies of potentially great benefit: ECMO' (by J. H. Ware). *Statistical Science, 5*, 310-317.

Ma, Q., Wang, J.T.L. & Wu, C. (2000). Application of Bayesian Neural Networks to Biological Data Mining: A Case Study in DNA sequence classification. *Proceedings of the Twelfth International Conference on Software Engineering and Knowledge Engineering*, July 6-8, 2000, IL, USA, pp. 23-30.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation, 4*, 415-447.

Macrossan, P. E., Abbass, H. A., Mengersen, K., Towsey, M. & Finn, G. (1999). Bayesian Neural Network Learning for Prediction in the Australian Dairy Industry. *Lecture Notes in Computer Science LNCS1642*, pp. 395-406.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. (1953). Equation of state calculations by fast computing machine. *Journal of Chemical Physics, 21*, 1087-1091.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer, New York.

Press, S. J. & Tanur, J. M. (2001). *The Subjectivity of Scientists and the Bayesian Approach.* John Wiley & Sons, Inc., New York.

Ramoni, M. & Sebastiani, P. (1999). Bayesian methods. In M. Berthold and D. J. Hand

(Eds.) *Intelligent data analysis: An introduction*, (pp. 129-166) Berlin: Springer-Verlag.

Spiegelhalter, D. J., Thomas, A., Best, N. G., & Gilks, W. R. (1995). *BUGS: Bayesian Inference using Gibbs Sampling, Version 0. 50.* MRC Biostatistics Unit, Cambridge.

Walsh, J. B. (in press). Quantitative genetics in the Age of Genomics. *Theoretical Population Biology.*

Chapter VIII

How Size Matters: The Role of Sampling in Data Mining

Paul D. Scott
University of Essex, UK

This chapter addresses the question of how to decide how large a sample is necessary in order to apply a particular data mining procedure to a given data set. A brief review of the main results of basic sampling theory is followed by a detailed consideration and comparison of the impact of simple random sample size on two well-known data mining procedures: naïve Bayes classifiers and decision tree induction. It is shown that both the learning procedure and the data set have a major impact on the size of sample required but that the size of the data set itself has little effect. The next section introduces a more sophisticated form of sampling, disproportionate stratification, and shows how it may be used to make much more effective use of limited processing resources. This section also includes a discussion of dynamic and static sampling. An examination of the impact of target function complexity concludes that neither target function complexity nor size of the attribute tuple space need be considered explicitly in determining sample size. The chapter concludes with a summary of the major results, a consideration of their relevance for small data sets and some brief remarks on the role of sampling for other data mining procedures.

INTRODUCTION

When data mining emerged as a distinct field, it was plausibly claimed that the total quantity of information stored in databases doubled every 20 months (Frawley, Piatetsky-Shapiro & Matheus, 1991). The credo of the new discipline was that the effort expended on accumulating and storing this prodigious quantity of data should be regarded as an investment that had created a resource ripe for exploitation. Machine learning had produced a number of well-proven techniques for automatically discovering regularities and patterns in data sets. The idea of applying these techniques to find the untapped seams of useful information in these vast deposits of data was the starting point of the new discipline. In the subsequent decade, size appears to have undergone a seismic shift in status: very large databases are now regarded as problematic because it may not be possible to process them efficiently using standard machine learning procedures. The problem is particularly acute when the data is too large to fit into main memory.

There are three basic approaches to dealing with this problem: first, develop new algorithms with more modest space/time requirements; second, use existing algorithms but implement them on parallel hardware (see Freitas & Lavington, 1998 for review); and third, apply the learning procedures to an appropriate sample drawn from the data set.

Machine learning practitioners appear uncomfortable with the idea of sampling; for them, it is what John and Langley (1996) describe as “a scary prospect”. Why this should be is something of a puzzle, since sampling theory is a long-established area of study. Standard introductory texts on statistics (e.g. Wonnacott & Wonnacott, 1990) typically include a treatment of those basic aspects of the subject that bear directly on hypothesis testing; Mitchell (1997) covers similar material in a machine learning context. Sampling itself is usually treated in separate texts: Kalton (1983) provides a concise introduction, while Kish (1965) provides a more comprehensive and mathematically grounded coverage.

In this chapter I shall be concerned with one central question: how do you decide how large a sample you need in order to apply a particular data mining procedure to a given data set. In the next section I discuss why sampling is unavoidable and review the main results of basic sampling theory. The following section comprises a detailed examination of the impact of sample size on two well-known data mining procedures: naïve Bayes classifiers and decision tree induction. The next section introduces disproportionate stratification and shows how it may be used to make much more effective use of limited processing resources. This section also includes a discussion of dynamic and static sampling. This is followed by a section devoted to the impact of target function complexity on sample size. The final section provides a summary of the major results, a consideration of their relevance for small data sets and some brief remarks on the role of sampling for other data mining procedures.

SAMPLING

Many people reject the idea of sampling on intuitive grounds. Unfortunately, human intuition is often poor on matters concerning sampling; the erroneous belief that a larger population implies the need for a correspondingly larger sample is very common. Nevertheless, it is obvious that, all other things being equal, using the whole population will never be worse even if it is not significantly better. There are two major difficulties with this position. First, other things seldom are equal. In most practical situations, resources are limited and appropriate sampling strategies can lead to more effective use of limited resources. Second, the issue of sampling is usually inescapable. In data mining we are seldom given the whole of a population. Even very large data sets are usually samples of some larger population. An objection to sampling in principle is not a coherent position because data mining is almost always concerned with data sets that are themselves samples. A principled approach to data mining must therefore include a systematic consideration of the effect of sample size.

Confidence Limits on Proportion Estimates

The majority of machine learning procedures seek to discover associations between particular sets of attribute values. Most of them do this by counting, explicitly or implicitly, the occurrences of particular attribute value combinations in a training set and hence estimating the proportion of the population in which those combinations are found. These proportion estimates are often interpreted as probabilities. Consequently, the question of what effect sample size has on the accuracy of such proportion estimates is central to any examination of the effective and efficient mining of very large data sets.

The mathematical development of sampling theory assumes that a random sample has been taken from a larger population. A range of random sampling procedures are considered in the literature (Kish, 1965) but only a few of these are relevant to this discussion. A simple random sampling (SRS) procedure is defined as one that is equally likely to choose any of the possible subsets of the specified number of items from the population.

Suppose that we have an SRS, comprising *n* items selected from a population of size *N*, and that we wish to estimate the proportion of these that have some particular attribute value (or combination of attribute values) and to place confidence limits on our estimate. This problem is actually a special case of a more general one: estimating the population mean, $E(Y)$, of a numeric attribute, Y, from a simple random sample of *n* items. Common sense suggests that the best estimate of $E(Y)$ would be the sample mean, denoted $E(y)$, obtained by taking the average of the Y attribute values across the sample. In this case common sense is right; $E(y)$ is an unbiased estimator of $E(Y)$: if the population were sampled repeatedly, the mean of the resulting means would itself converge to $E(Y)$. The extent to which the sample means, $E(y)$, obtained through such repeated sampling, are scattered about their mean determines the confidence we can place in any one of them as an estimate of the population mean. The variance of such a sample mean, $V(E(y))$ is given by

$$V(E(y)) = \left(\frac{N-n}{N} \right) \frac{S^2}{n}$$

where N is the size of the population from which the sample was taken and S^2 is the population variance (see e.g., Kish, 1965, for derivation). It is convenient to define n/N as the *sampling fraction*, denoted by the symbol f. The equation may thus be rewritten:

$$V(E(y)) = (1-f)\frac{S^2}{n}$$

Provided the sample size is large enough to justify using a normal approximation, the 95% confidence interval (that is, the range within which all but 5% of values will fall) is given by

$$E(Y) = E(y) \pm 1.96\sqrt{(1-f)\frac{s^2}{n}}$$

where the coefficient 1.96 is $z_{0.025}$, the value cutting off the upper 2.5% of a standardised normal distribution, and s^2 is the sample variance.

This result can be used to determine the confidence limits when using sample proportions as estimators for population proportions. Suppose that an additional attribute is added to all the items; it takes a value of 1 when the item has the combination of attribute values under consideration and 0 otherwise. Estimating the proportion of items having that set of values is directly equivalent to estimating the mean of the new attribute. Since the latter has only two values the sample means will be distributed binomially with mean P and variance P (1-P). Hence the 95% confidence interval for a proportion estimate is

$$P = p \pm 1.96\sqrt{(1-f)\frac{p(1-p)}{n-1}} \qquad (1)$$

where p is the sample proportion and P is the estimated population proportion.

Equation (1) shows that the accuracy of such a proportion estimate depends on three factors: the sampling fraction f, the sample size n, and the proportion estimate itself, p. There is a widespread but incorrect intuition that a larger population implies the need for a significantly larger sample. The sampling fraction, f, will be very small if the population from which the sample is drawn is much larger than the

Table 1: The effect of sample size and proportion (P) on the 95% confidence interval for proportion estimate from a sample

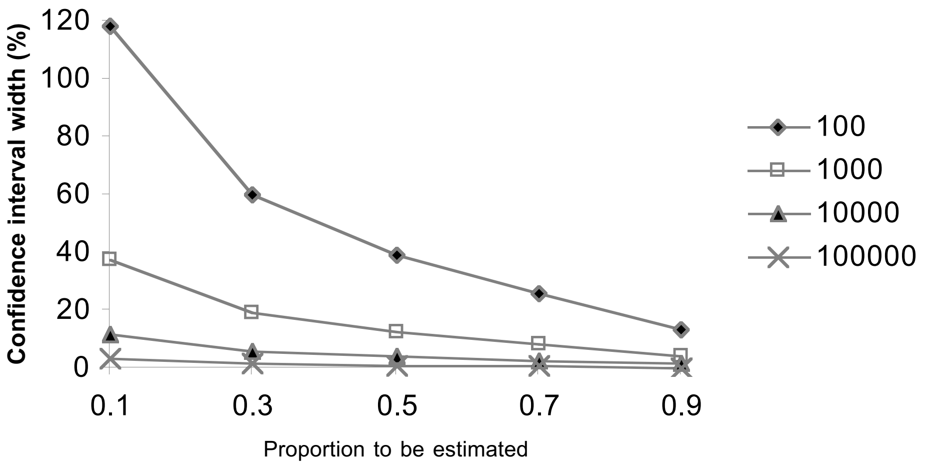
P	Sample Size			
	100	1000	10000	100000
0.1	±0.0591	±0.0186	±0.00588	±0.00186
0.3	±0.0903	±0.0284	±0.00898	±0.0284
0.5	±0.0985	±0.0310	±0.00980	±0.00310
0.7	±0.0903	±0.0284	±0.00898	±0.0284
0.9	±0.0591	±0.0186	±0.00588	±0.00186

sample; hence, the sample size required for a given precision is almost independent of population size.

Equation (1) has a maximum when *p* is 0.5, so the confidence interval will be widest when the proportion, *P*, to be estimated has this value. It is also clear that the width of the confidence interval will be inversely proportional to the square root of the sample size. Both these effects are demonstrated in Table 1, which shows the 95% confidence intervals for a range of proportions and samples of different sizes drawn from a population of 10^8 items.

This description of the effect of *P* on the confidence interval must be interpreted cautiously. Although the confidence intervals for *p* and (1-*p*) are necessarily the same, that interval may be a much larger fraction of one than of the other. The relative confidence interval, that is, the width of the confidence interval expressed as a percentage of the proportion to be estimated, is proportional to $\sqrt{((1-p)/p)}$. Hence, the sample needed to estimate a proportion of 0.1 is almost 100 times larger than that needed to estimate one of 0.9 with the same relative confidence interval. This effect is demonstrated in Figure 1 in which the data from Table 1 are presented as percentages of the proportions to be estimated. In most machine learning procedures, it is the relative rather than the absolute error in parameter estimates that is important, and hence small probabilities will play an important role in determining the necessary sample size.

Figure 1: The 95% confidence interval widths, given in Table 1, plotted as a percentage of the proportion to be estimated for sample sizes ranging from 100 to 100000



IMPACT OF SAMPLE SIZE ON MACHINE LEARNING PROCEDURES

These results identify the factors that influence the precision of proportion estimates and hence determine how large a sample is needed to estimate a probability with a specified confidence. In data mining many such estimates are typically compared or combined. In this section we investigate the impact of such estimates on the performance of machine learning procedures. Two well-known procedures, naïve Bayes classification and decision tree induction, are examined in detail. These particular procedures were chosen because they build very different models of the data and hence have very different sampling requirements.

Naïve Bayes Classifiers

We begin by considering naïve Bayes classifiers (Duda & Hart, 1973). Such classifiers are based on the assumption that all the attributes to be used for prediction are independent given the class to be predicted. Hence the probability of a particular combination of attribute values given a specific classification is

$$P(a_1 \wedge \ldots \wedge a_k \mid y_j) = \prod_{i=1}^{k} P(a_i \mid y_j)$$

where $a_1 \ldots a_k$ denote particular values of the predicting attributes and y_j is the classification. The classification most likely to give rise to this combination of attributes, y_{MAP}, can be found by applying Bayes Theorem:

$$y_{MAP} = \underset{y_j \in Y}{\arg\max} \left(P(y_j) \prod_{i=1}^{k} P(a_i \mid y_j) \right)$$

where Y is the set of possible classifications. Only a set of unconditional prior probabilities $P(y_j)$ and a set of conditional probabilities $P(a_i|y_j)$ are required to determine y_{MAP}. These will be estimated from proportions determined by counting appropriate items in the training set.

How large a sample is needed to construct a naïve Bayes classifier for a given data set? Clearly the sample should contain sufficient items to provide a good estimate of each of the probabilities. Since the naïve Bayesian method involves a comparison of the products of probability estimates, relative accuracy is important. As shown above, larger samples are needed to estimate the relative accuracies of smaller probabilities. Hence the size of sample needed will be determined by both the smallest non-zero probability to be estimated and the precision with which that estimate must be made.

How precisely should the conditional probabilities be estimated? If the attributes were all highly correlated given the class then it is very likely that all the probability estimates in a product would be too low (or too high) if any one of them is. In such circumstances, the relative errors would combine to produce a larger relative error in the product. However the assumption underlying the naïve Bayes approach is that the conditional probabilities are independent; under such circumstances some estimates will be too high while others are too low, and their accumulation as the product is formed will be much slower. In practice, a relative accuracy of ±10% is likely to be adequate.

In order to demonstrate how these ideas can be applied, we will consider a particular example. The data set chosen is a very small one used as a running expository example in Witten and Frank (2000). It comprises a set of five-tuples, which we will denote $<a,b,c,d,y>$. The task is to predict the value of the binary classification y from the values of a, b, c and d. a and b have three possible values; c and d are binary. The data set includes only 14 tuples; 9 have a classification y_1 while the remaining 5 have a classification y_2. The lowest non-zero probabilities are $P(b_3|y_2)$ and $P(c_2|y_2)$ which each have a value of 0.2. Applying Equation (1) implies a sample of about 1,530 would be needed to estimate these probabilities with ±10% accuracy. Since $P(y_2)$ is 0.36, a total sample of about 4,250 would be needed to produce enough examples of class y_2 and hence this is the size of training set that is needed.

The validity of this conclusion can be demonstrated by examining the behaviour of the confidence limits on the probability estimates as the sample size is increased. For the original set of only 14 items, applying Equation (1) gives the 95% confidence intervals for the unconditional prior probabilities as

$$P(y_1) = 0.643\pm0.261, \qquad P(y_2) = 0.357\pm0.261$$

Estimates for the 20 conditional probabilities needed and their associated confidence intervals can be computed in a similar way. The conditional probabilities given y_1 will be based on a sample of 9 while those given y_2 will be based on a sample of only 5. The results are shown in Table 2. (Confidence intervals derived from such small samples should be calculated using the binomial distribution rather than the much more convenient normal distribution. The results are accurate enough for present purposes for all the estimates except those based on zero counts, where the

Table 2: Conditional probability estimates and their 95% confidence intervals for a sample of 14 items (see text).

	$y = y_1$	$y = y_2$
$P(a_1\|y)$	0.222 ± 0.288	0.600 ± 0.480
$P(a_2\|y)$	0.444 ± 0.344	0.000 + 0.451
$P(a_3\|y)$	0.333 ± 0.327	0.400 ± 0.480
$P(b_1\|y)$	0.222 ± 0.288	0.400 ± 0.480
$P(b_2\|y)$	0.444 ± 0.344	0.400 ± 0.480
$P(b_3\|y)$	0.333 ± 0.327	0.200 ± 0.392
$P(c_1\|y)$	0.333 ± 0.327	0.800 ± 0.392
$P(c_2\|y)$	0.667 ± 0.327	0.200 ± 0.392
$P(d_1\|y)$	0.667 ± 0.327	0.400 ± 0.480
$P(d_2\|y)$	0.333 ± 0.327	0.600 ± 0.480

binomial distribution has been used). Note that most of the confidence interval widths exceed the magnitudes of the conditional probability estimates.

What is the impact of this high uncertainty on the behaviour of the naïve Bayes classifier? Witten & Frank (2000) use the tuple $<a_1,b_3,c_1,d_2>$ as an illustrative example. The probability estimates just derived produce values for $P(<a_1,b_3,c_1,d_2>)$ of 0.0053 if the class is y_1 and 0.0206 if the class is y_2. Hence the naïve Bayes classification would be y_2. However, this conclusion depends on the assumption that the ten probability estimates involved in the calculation are accurate. The worst case results, obtained when all the estimates are at either the lowest or highest bounds of their confidence intervals, are shown in the first column of Table 3. The ranges of possible values for the two estimates of $P(<a_1,b_3,c_1,d_2>)$ overlap completely, so the conclusion appears decidedly dubious.

Suppose exactly the same probability estimates had been produced from a larger sample. The confidence interval widths would be reduced by a factor of approximately ÷10 for every order of magnitude increase in sample size. The effect of such increased precision on the worst case bounds for the estimates of $P(<a_1,b_3,c_1,d_2>)$ is shown in the rest of Table 3. With 140 items, the ranges still overlap, with 1,400 items the ranges are clearly distinct, and only a small increase in their separation is achieved by increasing the sample to 14,000.

For classifying this particular item there appears to be no point in using a sample substantially larger than 1400. This does not contradict the conclusion that a sample of 4,250 was required. In this case, the alternative values for $P(<a_1,b_3,c_1,d_2>)$ are well separated; had they been closer the greater precision produced by a larger sample would have been necessary.

These conclusions suggest that a naïve Bayes classifier need not consider all the items in a very large data set. A small initial random sample could be taken to provide rough estimates of the probability parameters and these estimates could be used to determine how large a sample was necessary. The potential saving is not as large as that for many other machine learning procedures because constructing a naïve Bayes model has a constant space complexity for a given set of attributes and classes. Furthermore, in contrast to the method to be considered next, only one model is constructed.

Table 3: Bounds on probability of a specified attribute tuple given class y *for different sample sizes (see text). Expected values are:* y_1 *0.0053;* y_2 *0.021.*

		Sample			
		14	140	1400	14000
y1	Low	0.000	0.001	0.003	0.005
	High	0.132	0.017	0.008	0.006
y2	Low	0.000	0.004	0.012	0.018
	High	0.508	0.068	0.031	0.023

Decision Tree Induction

Decision tree induction (Breiman, Freidman, Olsen & Stone 1984; Quinlan, 1986) depends fundamentally on deriving probabilities by estimating proportions from a sample. Although there are many variations, all decision tree induction procedures construct the tree in a top down fashion; at each stage of construction a terminal node is expanded by selecting the attribute that best partitions the set of examples associated with that node. Various attribute selection criteria have been used but all depend on probability estimates. We will use information gain but similar conclusions would apply to all the alternatives.

Information gain for attribute a is defined

$$Gain_a \equiv U_0 - \sum_{i=1}^{k} P(a_i)U(a_i) \qquad (3)$$

where U_0 is the entropy for the entire set of examples and $U(a_i)$ is the entropy for the subset of examples having the ith value of attribute a. Entropy is defined

$$U \equiv -\sum_{y \in Y} P(y)\log_2 P(y)$$

where the sum is taken over the set of all possible values of classification y. Thus calculating U_0 requires estimates of the unconditional prior probabilities $P(y)$, while calculating each $U(a_i)$ requires estimates of the conditional probabilities $P(y_j|a_i)$. The derivation of confidence intervals for the probability estimates is similar to that for the naïve Bayes classifier. However, because entropy is not a monotonic function of its constituent probabilities, the bounds of the confidence interval for U

Table 4: Estimated values and bounds of confidence intervals at two sample sizes for entropies U(a_i) *(see text).*

x	$U(x)$				
		Sample Size			
		14		140	
	Estimate	Low	High	Low	High
a_1	0.971	0.000	1.000	0.831	1.000
a_2	0.000	0.000	0.998	0.000	0.374
a_3	0.971	0.000	1.000	0.831	1.000
b_1	1.000	0.000	1.000	0.927	1.000
b_2	0.918	0.000	1.000	0.747	0.994
b_3	0.811	0.000	1.000	0.512	0.962
c_1	0.985	0.207	1.000	0.895	1.000
c_2	0.592	0.000	1.000	0.329	0.770
d_1	0.811	0.000	1.000	0.621	0.930
d_2	1.000	0.334	1.000	0.953	1.000

do not necessarily correspond to the bounds of the probability estimates. Where there are only two possible classifications, dealing with this complication is a matter of determining if the probability interval includes the point 0.5; if it does then the upper bound on U is 1 and the lower bound is the lesser of the two values corresponding to the probability bounds estimate.

We shall explore the impact of sample size on decision tree induction using the same example data set as in our discussion of naïve Bayes classification. We begin by considering the choice of attribute for expanding the root node. The smallest non-zero probability to be estimated is $P(y_2|c_2)$, which is 0.143. Equation (1) implies that a sample of about 2,300 would be needed to estimate this with a relative error of ±10%. Since $P(c_2)$ is 0.5 in this data set, a total sample of 4,600 is required.

As before, we demonstrate the validity of this conclusion and the effect of varying sample size by considering the consequences of different-sized samples for the entropy and information gain estimates. In the example data set of 14 items, the estimated value of U_0 is 0.940. However, because the sample is small, the confidence interval is broad with a lower bound of 0.562 and an upper bound of 1.0. The estimates for the entropies $U(a_i)$ are derived from smaller samples: in each case, the examples having the *i*th value of attribute *a*. Consequently the confidence intervals are very wide (see Table 4); in the majority of cases they cover the entire possible range from 0 to 1. It is clear that 14 is far too small a sample size on which to base a reliable selection of the best attribute. Table 4 also shows the corresponding values for a sample ten times larger that produced the same probability estimates. The confidence interval for U_0 is reduced to the range 0.860..1.00.

What are the implications of this imprecision for the estimates of information gain that determine the choice of best attribute? The estimated values are: Gain_a 0.247, Gain_b 0.029, Gain_c 0.152 and Gain_d 0.048. These values suggest attribute *a* should be chosen. The associated confidence intervals for different sample sizes are shown in Table 5. As the results in Table 4 suggested, a sample size of 14 does not provide a useful estimate of the gains; their ranges overlap substantially and cover

Table 5: Bounds on information gain provided by each attribute for different sample sizes (see text). Expected values are: a *0.247;* b *0.029;* c *0.152;* d *0.048.*

Information Gain		Sample			
		14	140	1400	14000
a	Low	0	0.072	0.193	0.232
	High	0.940	0.409	0.291	0.260
b	Low	0	0	0	0.017
	High	0.940	0.240	0.077	0.043
c	Low	0	0.036	0.106	0.136
	High	0.893	0.376	0.236	0.166
d	Low	0	0	0.019	0.038
	High	0.887	0.205	0.085	0.059

most of the possible range of values. With a sample of 140, there is some evidence that *a* is the best choice, but its range still overlaps those of the others, particularly that of attribute *c*. Even a sample of 1,400 is insufficient to separate *a* and *c*, but when it is increased to 14,000, *a* is very clearly the attribute producing the greatest information gain. This result is consistent with the original calculation that a sample of 4600 was required.

At this point, it might appear that a naïve Bayes classifier and decision tree induction require similar sample size but this is not the case. The naïve Bayes method, which simply constructs one model to be used for classification, has finished whereas decision tree induction has only just started. Four models have been built, one for each attribute, and the best selected. The remaining models are discarded. If the final tree is to reflect any association between the rejected attributes and the class, the process must be repeated for each of the newly created daughter nodes. Thus, the same considerations of sample size must now be applied to each of the sample subsets created by partitioning on the selected attribute. Only a proportion of the original sample is associated with each daughter node. Hence, the original sample size must normally be increased substantially to provide large enough subsamples for the reliable selection of attributes to expand the daughter nodes.

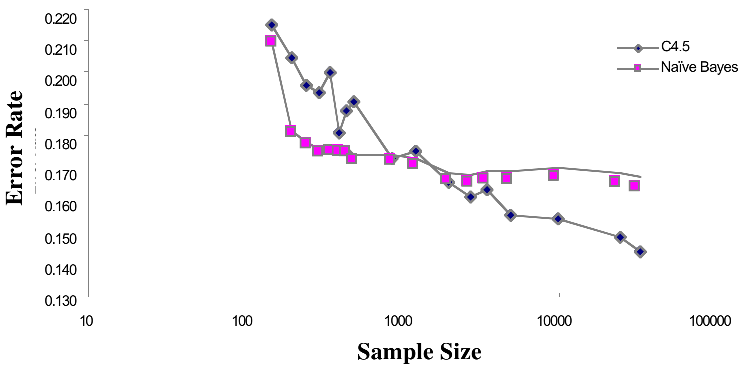
It is thus clear that the repeated fragmentation of the original sample, as the decision tree is constructed, is likely to increase the sample size required very significantly. It is also clear that, in contrast to the naïve Bayes classifier, it is usually impossible to estimate the necessary sample size on the basis of a small initial sample, since the relevant probabilities depend on the particular tree that is developed. One way of addressing this problem is to build new trees, using successively larger samples, until a satisfactory performance is achieved: an early example is the windowing method employed in ID3 (Quinlan, 1979). This approach is known variously as iterative sampling and dynamic sampling (John & Langley, 1996). It will be discussed in more detail later in the section on stratification.

Comparison of Naïve Bayes Classifiers and Decision Tree Induction

These detailed examinations of the impact of sample size on the performance of naïve Bayes classification and decision tree induction demonstrate that there are important differences. This implies that there can be no simple answer to the question “How big a sample do you need for data mining?” since the answer will depend on the particular procedure employed.

The fragmentation of the sample that occurs as the tree is built up means that a decision tree induction procedure will typically, though not necessarily, require a larger sample than a naïve Bayes classifier. The sample economy of the latter is a direct consequence of its basic assumption that the conditional probabilities are independent given the classification. There is a penalty for this economy. The resulting model cannot reflect any attribute interactions: that is, dependencies on

Figure 2: Error rate as a function of sample size for naïve Bayes classification and decision tree induction Each point is derived from the average of 5 runs (see text).



combinations of attribute values other than those implied by the independence assumption. No such restriction applies to decision trees. Thus, decision tree induction procedures typically require larger samples because they must acquire evidence of attribute interactions. It is not surprising that more data is needed to build a model that is both more complex and potentially, though not necessarily, more accurate.

These differences are demonstrated in Figure 2. This shows the results obtained when a naïve Bayes classifier and a decision tree induction procedure are applied to samples of different sizes, taken from the well-known Adult Census data set, available in the UCI repository of machine learning databases (Blake & Merz, 1998). This comprises 48,842 examples. Samples ranging from 147 to 32578 were selected as training sets; in each case the unselected examples were used as the test set. The samples were randomly selected and the experiments run using MLC++ (1998). The naïve Bayes procedure was that distributed with MLC++; the decision tree induction procedure was C4.5 (Release 8) (Quinlan, 1993, 1996).

The naïve Bayes classifier improves rapidly as the sample size increases and is close to its best performance by the time the sample size reaches 300. The further improvement produced using training sets 100 times as big (32,578) was only just statistically significant at the 5% level. The decision tree procedure performed less well than the naïve Bayes classifier for small sample sizes but continued to improve as larger training sets were used and ultimately performed significantly better. Samples of at least 4,800 were needed to bring it close to its best performance. Thus, for this particular data set, decision tree induction requires at least an order of magnitude more training data than naïve Bayes classification.

The disparity in sample requirements of naïve Bayesian classifiers and decision tree induction also explains why the latter is much more vulnerable to overfitting. It is not a major problem for naïve Bayes classifiers since the entire sample is available for all parameter estimates. In contrast, because decision tree induction procedures repeatedly fragment the original sample, a stage is eventually reached in which the probability estimates may be very inaccurate and hence overfitting occurs. The preceding discussion suggests that directly addressing the question of sample size might provide a robust method of pre-pruning.

The results discussed in this section could also provide an explanation for the comparative success of naïve Bayes classifiers. A number of studies have shown that such programs achieve results comparable with those obtained using decision tree induction or neural net methods (see, for example, Michie, Spiegelhalter & Taylor,1994). This is a surprising finding because real data sets seldom conform to the independence assumptions upon which naïve Bayes classification is based. However, as we have seen, a method that is not based on such assumptions may require very substantially more data in order to achieve a significant performance advantage.

STRATIFIED SAMPLING

Considerable reductions in sample size can be realised through a technique known as *stratification* (Kalton, 1983; Kish, 1965). It is particularly useful when there are large differences in the frequencies of class or attribute values. The basic idea of stratification is simple: the population is partitioned into subsets, called *strata*, using the values of some attribute (or group of attributes); a separate random sample is then selected from each stratum. If the strata sample sizes are proportional to the strata population sizes, such a procedure is called *proportionate stratification*; otherwise it is known as *disproportionate stratification*.

Stratification for the Naïve Bayes Classifier

In the preceding section, it was shown that the size of sample required for a naïve Bayes classifier was determined by the need to have enough items to make a sufficiently accurate estimate of the smallest non-zero conditional probabilities. In the example data set the lowest such probabilities are $P(b_3|y_2)$ and $P(c_2|y_2)$ which were each 0.2. These are derived from those items in class y_2, which constituted 36% of the whole sample. Suppose that items of class y_2 had been much rarer and comprised only 3.6% of the population. In order to obtain enough items to estimate $P(b_3|y_2)$ and $P(c_2|y_2)$, it would have been necessary to increase the total sample tenfold from 4,250 to 42,500. Of these, 96.4% (i.e., 40,970) would be items of class y_1. This is greatly in excess of the number needed to make estimates of the conditional probabilities of form $P(x|y_1)$. Consequently confidence intervals for these probability estimates would be about five times narrower than those for the conditional probabilities $P(x|y_2)$. This additional precision would be of no benefit since the ultimate objective is to compare the posterior probabilities of y_1 and y_2. Hence over 90% of the sample, and the effort required to process it, would be wasted. This type of problem, which arises from major disparities in the class or attribute frequencies, is very common with real data sets.

Disproportionate stratification offers a solution to the problem of major disparities in class or attribute frequencies. The earlier discussion of naïve Bayes classifiers assumed that the same single random sample would be used for all probability estimates. An alternative approach would be to stratify the data set by class; each stratum would be used to estimate the corresponding conditional probabilities and hence each stratum sample need only be larger enough to estimate those probabilities. The example data set would require samples of about 1340 from the y_1 stratum and 1530 from the y_2 stratum, giving a total of 2870. These numbers would be the same whatever the relative proportions of y_1 and y_2, so very big savings may be achieved when the disparities are large.

The estimates of the prior unconditional probabilities, $P(y)$, require consideration of the whole population. They cannot be estimated from the strata samples since these are based on assumptions about their values. The example data set would require about 700 items to estimate the priors. If the initial small random sample, taken to estimate the approximate magnitude of the probabilities, was large enough no further sampling would be necessary. Alternatively, a further simple random sample of the whole population would be needed.

It should also be noted that using disproportionate stratification could enable a model to be built when simple random sampling would suggest it not feasible. If one class is very rare, the simple sample size required may be too large to be processed. Such a sample would be almost entirely made up of the commoner classes; these items would contribute little to the accuracy of the model and so are not necessary. Provided the data set contains sufficient examples of the rare class, disproportionate stratification eliminates this potential problem.

Stratification for Decision Tree Induction

Disproportionate stratification could be used in a similar way to select the best attribute for expanding the root node of a decision tree. The procedure would be more complex because a different stratification would be needed for each candidate attribute. These strata samples would be large enough to select the best attribute for the root node, but unfortunately, they would usually be too small for the expansion of its daughter nodes.

This problem could be addressed by resampling every time a node is expanded. The stratum sample inherited when the parent was expanded would be used to provide approximate probability estimates; these would be used to select the appropriate disproportionate strata samples for the candidate attributes. Note that it would only be necessary to select sufficient *additional* items to increase the stratum sample to the required size. There are two advantages in this approach: first, all node expansions would be based on sufficient evidence, thus greatly reducing the chances of overfitting, and, second, none of the samples would be unnecessarily large. The disadvantage is that it could involve a lot of strata sampling, though this would be reduced if it was confined to plausible candidates for the best attribute using the peepholing method (Catlett, 1992).

Static and Dynamic Sampling

This procedure of taking new strata samples every time a node is expanded is an example of what John and Langley (1996) call *dynamic sampling*. They propose a different form of dynamic sampling, applicable in principle to any learning procedure, in which the sample is repeatedly increased, and a complete new model derived, until no significant performance improvement is achieved. This is computationally efficient only if it is easy to derive the new model from its predecessor. As they point out, their approach automatically adjusts the choice of sample size to that appropriate for the learning procedure employed.

It was therefore somewhat perverse of John and Langley to use naïve Bayes classification as the learning method to demonstrate experimentally the value of their approach. As we have seen, because it builds a simple model, the naïve Bayes classifier has no need of dynamic sampling, except in a very trivial sense. It is possible to determine the optimal sample size on the basis of a small initial sample before constructing the naïve Bayes model: in other words, *static sampling* is sufficient for such systems. John and Langley (1996) claim that "static sampling ignores the data-mining tool that will be used". As the detailed discussions above demonstrate, this is not true; it is not only possible but also highly desirable to use information about the tool to be used in determining sample size statically.

COMPLEXITY OF THE TARGET FUNCTION

So far we have only been indirectly concerned with the influence of the complexity of the target classification function. It has been addressed implicitly in that procedures that can model interactions between attributes (such as decision tree induction) can model more complex classification functions than can those that assume independence (such as naïve Bayes classifiers). However, target function complexity forms the main plank of the argument against sampling advanced by Freitas and Lavington (1998).

Their argument hinges on two closely related concepts: small disjuncts and dispersion. Small disjuncts occur whenever small regions of the attribute tuple space belong to a different class of all neighbouring regions. Freitas and Lavington argue that a small sample may not include items from such a region and hence the resulting model will be inaccurate. It is certainly true that too small a sample will indeed lead to such errors. The approach advocated in this chapter is to use the smallest sample that will provide good estimates of all the probabilities contributing to the model. The existence of such a small disjunct would imply the need for a larger sample, and stratification would avoid the need to consider large numbers of items from the rest of the attribute tuple space.

Even if a single small disjunct is not represented in the training sample, the effect on performance accuracy will be slight because the classifier will rarely be required to classify items from this region of tuple space. However, there are target functions that are composed entirely of such small disjuncts. Freitas and Lavington (1998) describe such functions as highly dispersed. A simple artificial example of a maximally dispersed target function is the parity of an N-bit binary vector. Target functions with high dispersion can arise in real data sets (Danyluk & Provost, 1993). As with single small disjuncts, a highly dispersed target function will require a large sample. Indeed, extreme examples, such as parity, usually require samples that cover the whole tuple space. As with single simple disjuncts, using the smallest sample that will provide good estimates of all the probabilities contributing to the model will lead to an appropriately large sample for a highly dispersed target function. The major difference is that the scope for economy through stratification will be reduced.

Sampling can help avoid wasted effort even with maximally dispersed target functions. Consider using naïve Bayes classification when the target function is the parity of a 20-bit binary vector. All probabilities are 0.5, so the sample size suggested by an initial small sample would be about 800, a small fraction of the 2^{20} items in the tuple space. The performance of the resulting classifier would be dismal, but the model parameter estimates would be reasonably accurate. However, further increases in sample size would produce no improvement because a naïve Bayes classifier cannot model this target function. There is nothing to be gained by increasing the sample size beyond that suggested by sampling theory.

Freitas and Lavington (1998) are right that target function complexity has a major effect on the required sample size, but this is not an argument against sampling. Rather it is an argument against a too simplistic approach to sampling. The sample size must be large enough to estimate the parameters of the model, and this will depend on both the target function and the particular model being built. Determining sample size on the basis of an initial small sample will avoid both sampling too few items for a complex target function and sampling an unnecessarily large number of items for a simple model.

Freitas and Lavington (1998) also suggest that the sample size should be a function of the size of the attribute tuple space. Their argument is an appeal to intuition. Although data sets with large tuple spaces may require large samples, this is because they offer greater scope for dispersed target functions and because the probabilities to be estimated will often be smaller. There is no need to consider this factor explicitly in determining the sample size, and to do so would be a mistake since large regions of the tuple space may be empty.

DISCUSSION

This section provides a summary of the major results, a consideration of their relevance for small data sets and some brief remarks on the role of sampling for other data mining procedures.

Conclusions: Implications for Large Data Sets

A number of important conclusions can be drawn from this investigation into the effect of sample size on naïve Bayes classifiers and decision tree induction:

- Basic sampling theory can be applied to determine the sample size needed when a particular learning procedure is applied to a particular data set. (Equation (1)).
- The use of sampling theory can often show that it is unnecessary to use more than a small fraction of a very large data set.
- Both the learning procedure and the data set have a major impact on the size of sample that is required. The learning procedure is important because the sample will be used to estimate the parameters that will determine its behaviour. Only data pertinent to those parameters has relevance to the required sample size. The data set is important because the actual distribution of items pertinent to estimating model parameters will determine how much data is necessary.
- The size of a large data set has no impact on the size of sample necessary.
- The necessary sample size can be determined without explicit consideration of the size of the attribute tuple space or the complexity of the target function.
- The sample size needed for a naïve Bayes classifier can be determined statically on the basis of a small initial sample used to gain information about the approximate values of the relevant probabilities.
- The sample size required to expand the root node of a decision tree can be determined in a similarly static manner, but expansion of subsequent nodes requires some form of dynamic sampling.
- Disproportionate stratification can be used to eliminate the need for very large samples when some classes or groups of attributes are uncommon. Very large economies in sample size are possible using this form of sampling.
- Disproportionate stratification is the natural form of dynamic sampling for decision tree induction and could be used as a basis for principled pre-pruning.
- Disproportionate stratification offers a solution to the dilemma presented by huge data sets because it concentrates resources on items of relevance to the model while potentially considering every item in the data set.

Implications for Small Data Sets

It is probably obvious that it is not necessary to consider 10^8 items if you want to estimate a probability of around 0.2, but the discovery that as many as 1,500 are required may be more of a surprise. Naïve Bayes classification and decision tree induction have been successfully applied to many data sets considerably smaller than this. These successes do not contradict the conclusions presented in this chapter. The calculated sample sizes were derived from the arbitrary, though reasonable, assumption that it was desirable to estimate probabilities with an accuracy of $\pm 10\%$. Had a larger figure been chosen, the required samples would have been smaller. Thus, success with a smaller data set implies that particular classification task can be performed using cruder estimates for the model parameters.

These bounds on sample size have important implications for the results of comparative studies of machine learning procedures. A recent example (Lim, Loh & Shih, 2000) compares 33 procedures using 16 data sets, of which only 3 contained more than 2000 items.. Had larger data sets been used the performance rank ordering may have been very different. As Figure 2 demonstrates, a sample of nearly 5,000 items may be needed before C4.5 demonstrates a clear advantage over the naïve Bayes classifier. If the results of such studies are to be applied in choosing data-mining procedures for very large data sets, they must be based on large data sets.

Other Machine Learning Procedures

The discussion in this chapter has been largely confined to two particular machine learning methods. We conclude with a few remarks on how far they apply to other techniques. In some cases extrapolation is straightforward. Procedures that use a sequential covering strategy to learn sets of propositional rules (e.g. CN2; Clark & Niblett, 1989) closely resemble decision tree induction in that the available sample is progressively reduced as further terms are added to a rule. Thus the required sample will depend on the number of terms included in the rules and it will not, in general, be possible to determine this in advance from a small initial sample. Thus dynamic proportionate sampling will be necessary.

The simplest form of Bayesian belief network (Pearl, 1988), in which the network structure is known and all variables are observable, has much in common with naïve Bayes classifiers. The fundamental difference is that some specified conditional independence assumptions are dropped. As with naïve Bayes classifiers it is known a priori which probability estimates will be required, so static sampling using a small initial sample is possible. How practical this would be depends on the number of conditional independence assumptions that are dropped. As more conditional dependencies are included in the model, so the number of probability estimates needed rises exponentially and the proportion of the sample relating to each of them diminishes. Thus the required sample size will grow rapidly with the number of included dependencies.

Instance based learning procedures (Aha, 1992) are fundamentally incremental in nature and hence lend themselves to an extreme form of dynamic sampling. It is not possible to estimate the required sample from a small initial sample, but it is possible to derive a bound on the likelihood of a future example being misclassified. This can be done by keeping a count of the number of correctly classified training examples that have been presented since the most recent misclassified example. The training set should be increased until this falls to an acceptably low level or no more data items are available.

Other data mining procedures have less resemblance to either naïve Bayes classification or decision tree induction. A large and important group is made up of those methods that data mining has acquired from statistics rather than machine learning. These are outside the scope of this chapter because there is a very substantial literature on these methods which includes discussions of the relationship between sample size and standard error of the model parameters (see, for example, Fox, 1984, on linear models). With these results, it is possible to determine the necessary sample size from an initial run using a small sample of the data.

Neural network methods, such as back-propagation using the extended delta rule (Rumelhart & McClelland, 1986) form another important group. It is notoriously difficult to derive simple models of the behaviour of such networks. Some insight into what might be expected from a back-propagation network can be obtained from the fact that its simpler forebear, the delta rule (Widrow & Hoff, 1960), is essentially a computationally expensive way of performing multiple regression on the training set; the weights will converge on the values of the regression coefficients. Hence the standard results for regression also apply to the delta rule. These provide lower bounds to the sample size that will be needed by back-propagation networks since the nonlinear modelling they perform is likely to require more data than finding the best fitting linear model. While such bounds may provide a useful warning when there is too little data, they can offer no reassurance that sufficient data has been used.

ACKNOWLEDGMENTS

This chapter has its origins in discussions with Alex Freitas that were ably refereed by Simon Lavington; while thanks are due to both for its existence, neither bear any responsibility for its conclusions. The work described was supported by EPSRC grant GR/M44705. Thanks are also due to David Hales for the experimental comparison of a naïve Bayes classifier and C4.5

REFERENCES

Aha, D. (1992). Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies. 36(2)*, 207-216.

Blake, C.L. & Merz, C.J. (1998). *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Breiman, L., Freidman, J. H., Olsen, R. A. & Stone, P. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Catlett, J. (1992). Peepholing: Choosing attributes efficiently for megainduction. In D. Sleeman & P. Edwards (Eds.). *Proc. Ninth Int. Workshop on Machine Learning (ML-92)* (pp 49-54). San Mateo, CA: Morgan Kaufmann.

Clark, P. & Niblett, R. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261-284.

Danyluk, A. P. & Provost, F. J. (1993). Small disjuncts in action: learning to diagnose errors in the local loop of the telephone network. *Proc. Tenth Int. Conf. Machine Learning*, 81-88.

Duda, R. O. & Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: John

Wiley.

Fox, J. (1984). *Linear statistical models and related methods*. New York: John Wiley.

Frawley, W. J., Piatetsky-Shapiro, G. & Matheus, C. J. (1991). Knowledge discovery in databases: an overview. In G. Piatetsky-Shapiro & W. J. Frawley (Eds.) *Knowledge discovery in databases* (pp 1-27). Menlo Park, CA.: AAAI Press/MIT Press.

Freitas, A. A. & Lavington, S. H. (1998). *Mining very large databases with parallel processing*. Boston: Kluwer.

Kalton, G. (1983). *Introduction to survey sampling*. Beverly Hills: Sage Publications.

Kish, L. (1965). *Survey sampling*. New York: John Wiley.

Lim, T.–S., Loh, W.–Y & Shih, Y.–S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning, 40*, 203-229.

Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (Eds.). (1994*). Machine learning, neural and statistical classification*. New York: Ellis Horwood.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill,

MLC++ (1998). *MLC++ Machine Learning Library in C++ , SUN MLC++ utilities v2.1*, [http://www.sgi.com]. Silicon Graphics Inc..

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In D. Michie (Ed.), *Expert systems in the micro electronic age*. Edinburg Univ. Press.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*, 81-106.

Quinlan, J. R. (1993) *Programs for Machine Learning*. Los Altos, CA: Morgan Kaufmann.

Quinlan, J. R. (1996). Improved Use of Continuous Attributes in C4.5, *Journal of Artificial Intelligence Research, 4*, 77-90.

Rumelhart, D. & McClelland, J. L. (1986). *Parallel distributed processing: Exploration in the microstructure of cognition*. Cambridge, MA: MIT Press.

Widrow, B. & Hoff, M. E. (1960). Adaptive switching circuits. *IRE WESCON Convention Record, 4*, 96-104.

Witten, I. H. & Frank, E. (2000). *Data mining: Practical machine learning tools and techniques with Java implementations*. San Francisco: Morgan Kaufmann.

Wonnacott, T. H. & Wonnacott, R. J. (1990). *Introductory statistics. (*5th ed.). New York: John Wiley.

Chapter IX

The Gamma Test

Antonia J. Jones, Dafydd Evans, Steve Margetts and Peter J. Durrant
Cardiff University, UK

The Gamma Test is a non-linear modelling analysis tool that allows us to quantify the extent to which a numerical input/output data set can be expressed as a smooth relationship. In essence, it allows us to efficiently calculate that part of the variance of the output that cannot be accounted for by the existence of any smooth model based on the inputs, even though this model is unknown. A key aspect of this tool is its speed: the Gamma Test has time complexity O(Mlog M), where M is the number of data-points. For data sets consisting of a few thousand points and a reasonable number of attributes, a single run of the Gamma Test typically takes a few seconds.

In this chapter we will show how the Gamma Test can be used in the construction of predictive models and classifiers for numerical data. In doing so, we will demonstrate the use of this technique for feature selection, and for the selection of embedding dimension when dealing with a time-series.

INTRODUCTION

The Gamma test was originally developed as a tool to aid the construction of data-derived models of *smooth* systems, where we seek to construct a model directly from a set of measurements of the system's behaviour, without assuming any *a priori* knowledge of the underlying equations that determine this behaviour. Neural networks may be considered as the generic example of a data-derived modelling technique.

We think of the system as transforming some *input* into a corresponding *output*, so the output is in some way 'determined' by the input. This is a fairly general representation – in the case of a dynamical system ,the current state of the system may be thought of as the input, with the output representing the state of the system after some time interval has elapsed.

One problem in constructing models solely on the basis of observation is that measurements are often corrupted by *noise*. We define noise to be any component of the output that cannot be accounted for by a smooth transformation of the corresponding input.

The Gamma test (Aðalbjörn Stefánsson, Končar & Jones, 1997; Končar, 1997) is a technique for estimating the noise level present in a data set. It computes this estimate *directly from the data* and does not assume anything regarding the parametric form of the equations that govern the system. The only requirement in this direction is that the system is *smooth* (i.e. the transformation from input to output is continuous and has bounded first partial derivatives over the input space).

Noise may occur in a set of measurements for several reasons:

- Inaccuracy of measurement.
- Not all causative factors that influence the output are included in the input.
- The underlying relationship between input and output is not smooth.

The applications of a data-derived estimate of noise for non-linear modelling are clear. In the first instance, it provides a measure of the *quality* of the data – if the noise level is high we may abandon any hope of fitting a smooth model to the data. In cases where the noise level is moderately low, the Gamma test can be used to determine the best time to stop fitting a model to the data set. If we fit a model beyond the point where the mean squared error over the training data falls significantly below the noise level, we will have incorporated some element of the noise into the model itself, and the model will perform poorly on previously unseen inputs despite the fact that its performance on the training data may be almost perfect. Taking our noise estimate as the optimal mean squared error by running the Gamma test for an increasing number of data points and seeing how many points are required for the noise estimate to stabilize, we may also obtain an indication of the number of data points required to build a model which can be expected to perform with this mean squared error .

It is not immediately apparent that a technique for estimating noise levels can be of use in data mining applications. Its usefulness derives from the fact that low noise levels will only be encountered when *all* of the principal causative factors that determine the output have been included in the input. Some input variables may be irrelevant, while others may be subject to high measurement error so that incorporating them into the model will be counter productive (leading to a higher *effective* noise level on the output). Since performing a single Gamma test is a relatively fast procedure, provided the number of possible inputs is not too large, we may compute a noise estimate for each subset of the input variables. The subset for which the associated noise estimate is closest to zero can then be taken as the "best selection" of inputs.

The objectives of this chapter are to provide a clear exposition of what the Gamma test is, to describe how it works, and to demonstrate how it can be used as a data mining tool.

BACKGROUND

Suppose we have a set of input-output observations of the form:

$$\{(\boldsymbol{x}_i, y_i) : 1 \le i \le M\} \qquad (1)$$

where the inputs $\boldsymbol{x} \in \Re^m$ are *vectors* confined to some closed bounded set $C \subseteq \mathbb{R}^m$, and the corresponding outputs $y \in \mathbb{R}$ are *scalars* (in the case where the outputs are vectors, the method may be applied independently to each component with very little extra computational cost).

The relationship between an input $\boldsymbol{x}$ and the corresponding output y is expressed as

$$y = f(\boldsymbol{x}) + r \qquad (2)$$

where

- f is a smooth function representing the system.
- r is a random variable representing noise.

Without loss of generality, since any constant bias can be absorbed into the unknown function f we may assume that the *mean* of the noise variable r is zero. Despite the fact that f is unknown, subject to certain conditions, the Gamma test provides an estimate for Var(r), the *variance* of the noise. This estimate is called the *Gamma statistic*, denoted by Γ, and may be derived in $O(M \log M)$ time[2] where the implied constant depends only on the dimension m of the input space. The Gamma test is a non-parametric technique and the results apply regardless of the particular methods subsequently used to build a model.

Before describing how the Gamma test works, we state the conditions under which the method may be applied.

Assumptions

First we restrict the domain of possible models to the *class of functions having bounded first and second partial derivatives*. We require:

F.1. The function f has bounded first and second partial derivatives. In particular, there exists some finite constant B such that $|\nabla f(\boldsymbol{x})| \le B$ for every $x \in C$.

Next we consider the question of how the input points are generated. In some situations it may be possible for the experimenter to set values for the inputs and then measure the corresponding output. However, in many data analysis situations, the system we seek to model may generate input and output values autonomously.

In general we suppose that inputs $\boldsymbol{x}_i$ are selected according to some sampling distribution having density function φ, whose support[3] is some closed bounded subset $C \subseteq \mathbb{R}^m$. We call this set the *input space*. As we shall see, the Gamma test computes its estimate for the variance of the noise by considering the hypothetical behaviour of arbitrarily close near neighbours in input space. We require that the distances between nearest neighbours become small as the number of points increases, and it is therefore a prerequisite of the method that the input space C is *perfect* so that in addition to being closed and bounded, the input space C *contains no isolated points*.

C.1 The input space C is a perfect set.

In many cases the support C has positive measure in the classical (Lebesgue) sense. Another case of interest is the analysis of *chaotic time series*. Here, following Takens (1981), we seek to model a time series by predicting the next value (the output) based on a number m of previous values (the input vector). For many chaotic time series this relationship is smooth and so the Gamma test might reasonably be applied. In such cases the input space C is a very often a chaotic attractor of *non-integral* dimension and of zero measure but this appears to be largely irrelevant to the estimates of noise variance returned by the Gamma test. Indeed, in the zero measure case the Gamma test may be more efficient (in terms of the number of data points required) because the input data is confined to some *lower dimensional* subset of the full input space.

Finally we must consider the random variable r representing the noise on an output. The probability density function of this random variable is denoted by η. To avoid confusion, an expectation taken with respect to η will be denoted by $\mathbf{E}_\eta$ while an expectation taken with respect to the sampling density function φ is denoted by $\mathbf{E}_\varphi$.

As we have seen, by including any bias within the system into the (unknown) smooth component f, we may suppose without loss of generality that $\mathbf{E}_\eta(r) = 0$ and as we aim to estimate the variance of the noise, we also require that $\mathbf{E}_\eta(r^2)$ is finite[4]:

N.1 $\mathbf{E}_\eta(r) = 0$ and $\mathbf{E}_\eta(r^2) < \infty$

We also need the following:

N.2 The noise is *independent* of the input vector $\boldsymbol{x}$ corresponding to the output y being measured, i.e. we assume that the noise on an output is *homogeneous* over the input space (if the noise is *not* homogeneous, the Gamma test will compute an estimate of the *average* noise variance—this can still provide useful information when selecting relevant inputs).

N.3 Two noise values r_i and r_j measured on two different outputs y_i and y_j are *independent* so that $\mathbf{E}_\eta(r_i r_j) = 0$ whenever $i \neq j$.

Description of the Method

The Gamma test works by exploiting the hypothesised continuity of the unknown function f. If two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are close together in input space then since *f* is continuous, we can also expect that $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ are close together in output space. If this is not the case, this can only be because of the addition of noise to each of these values.

Since $y=f(\boldsymbol{x})+r$ we have that:

$$\frac{1}{2}(y'-y)^2=\frac{1}{2}\left((r'-r)+(f(\boldsymbol{x}')-f(\boldsymbol{x}))\right)^2 \tag{3}$$

The continuity of *f* implies that $|f(\boldsymbol{x}')-f(\boldsymbol{x})|\rightarrow 0$ as $\boldsymbol{x}'\rightarrow\boldsymbol{x}$ so:

$$\frac{1}{2}(y'-y)^2\rightarrow\frac{1}{2}(r'-r)^2 \text{ as } \boldsymbol{x}'\rightarrow\boldsymbol{x} \tag{4}$$

Taking the expectation of both sides we get:

$$\mathbf{E}\left(\frac{1}{2}(y'-y)^2\right)\rightarrow \mathrm{Var}(r) \text{ as } \boldsymbol{x}'\rightarrow\boldsymbol{x} \tag{5}$$

Here, the expectation is defined with respect to the noise and sampling distributions. However, for any finite data set we cannot evaluate this limit directly.

Since *f* is assumed continuous over a closed bounded region $C\subseteq\mathbb{R}^m$, the error involved in estimating Var(r) by the expectation $\mathbf{E}(1/2(y'-y)^2)$ essentially depends on the distance $|\boldsymbol{x}'-\boldsymbol{x}|$. Clearly, this error will be minimised if we consider the differences $1/2(y'-y)^2$ between pairs of output values y and y' where the corresponding inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$ are *nearest neighbours* in input space.

Let $\boldsymbol{x}_{\mathrm{N}[i,k]}$ denote the *k*th nearest neighbour of the point $\boldsymbol{x}_i$ in the input set $\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_M\}$. This is defined to be the point for which there are exactly k-1 other points closer to $\boldsymbol{x}_i$[5]. Nearest neighbour lists can be found in O($M\log M$) time, for example, using the *kd*-tree method developed by Friedman, Bentley and Finkel (1977).

For *k* in the range $1\leq k\leq p$ (where *p* is typically taken in the range 10-50), we compute a sequence of estimates for $\mathbf{E}(1/2(y'-y^2))$ by the sample means[6]:

$$\gamma_M(k)=\frac{1}{2M}\sum_{i=1}^{M}|y_{\mathrm{N}[i,k]}-y_i|^2 \tag{6}$$

and in each case, we obtain an indication of the "error" by computing the mean square distance between *k*th nearest neighbours, defined by:

$$\delta_M(k)=\frac{1}{M}\sum_{i=1}^{M}|\boldsymbol{x}_{\mathrm{N}[i,k]}-\boldsymbol{x}_i|^2 \tag{7}$$

where |.| denotes the Euclidean distance[7]. As in (5) we can show that:

$$\gamma_M(k)\rightarrow\mathrm{Var}(r) \text{ as } \delta_M(k)\rightarrow 0 \tag{8}$$

However, given any finite data set we cannot make the distances between nearest neighbours arbitrarily small, so as in (5), we cannot evaluate this limit directly.

The Gamma test is based on the assertion that the relationship between $\gamma_M(k)$

and $\delta_M(k)$ is *approximately linear* near $\delta_M(k) = 0$, i.e., there exists some constant A such that:

$$\gamma_M(k) = \mathrm{Var}(r)+A\delta_M(k)+o(\delta_M(k)) \quad \text{as} \quad \delta_M(k)\to 0 \tag{9}$$

As a result of our condition that the input space has no isolated points, we can expect that the mean distance between nearest neighbours converges to zero (in probability) as the number of data points increases, that is:

$$\delta_M \to 0 \quad \text{as} \quad M\to\infty \tag{10}$$

Thus (9) becomes:

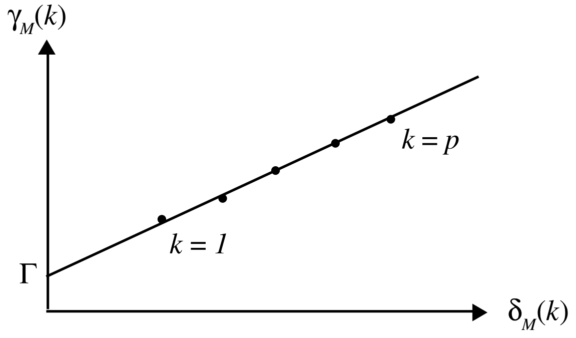
$$\gamma_M(k) = \mathrm{Var}(r)+A\delta_M(k)+o(\delta_M(k)) \quad \text{as} \quad M\to\infty \tag{11}$$

and we see that a linear relationship between $\gamma_M(k)$ and $\delta_M(k)$ can be established subject to the condition that we have sufficiently many data points.

Having established the linearity of $\gamma_M(k)$ and $\delta_M(k)$ we estimate the limit (8) of $\gamma_M(k)$ as $\delta_M(k)\to 0$ by performing linear regression on the pairs $\{\delta_M(k), \gamma_M(k)) : 1 \le k \le p\}$.

As illustrated in Figure 1, our estimate for Var(r) is determined by the intercept Γ of this regression line with the $\delta_M(k)=0$ axis.

Figure 1: The Gamma test regression plot



THE GAMMA TEST ALGORITHM

The method used to calculate the Gamma value is given in Algorithm 1. As well as returning the intercept Γ of the regression line the algorithm also returns its *gradient*, denoted by A. As we shall see, this often contains useful information regarding the system under investigation, represented by the (unknown) smooth function f.

Formal Justification of the Method

A rigorous mathematical justification of the Gamma test is established by the following result (Evans, 2001). The theorem states that with probability approaching one, the relationship between the points $(\delta_M(k), \gamma_M(k))$ becomes approximately linear as $M \to \infty$, with constant term equal to Var(r).

Algorithm 1: The Gamma test

```
Procedure Gamma test (data)
 (* data is a set of points $\{(\boldsymbol{x}_i, y_i): 1 \le i \le M\}$ where $\boldsymbol{x}_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$*)

(* Compute the nearest neighbour lists for each point—this can be done in
O($M\log M$) time *)
For $i = 1$ to $M$
    For $k = 1$ to $p$
        Compute the index $N[i, k]$ of the $k$th nearest neighbour of $\boldsymbol{x}_i$.
    endfor $k$
endfor $i$

(* If multiple outputs do the remainder for each output *)
For $k = 1$ to $p$
    Compute $\gamma_M(k)$ as in (6)
    Compute $\delta_M(k)$ as in (7)
endfor $k$

Perform linear regression on $\{\delta_M(k), \gamma_M(k)): 1 \le k \le p\}$, obtaining (say) $\gamma = Ax + \Gamma$
Return $(\Gamma, A)$
```

Theorem. Subject to conditions **F.1, C.1** and **N.1 - N.3** stated above, for every $\kappa > 0$ we have:

$$\gamma_M(k) = Var(r) + A(M,k)\ \delta_M(k) + o(\delta_M(k)) + O(M^{-1/2+\kappa}) \tag{12}$$

with probability greater than $1 - O(1/M^{2\kappa})$ as $M \to \infty$, where $A(M, k)$ is defined by:

$$A(M,k)\ \mathbf{E}_\varphi\ (|\boldsymbol{x}_{N[i,k]} - \boldsymbol{x}_i|^2) = 1/2\ \mathbf{E}_\varphi(((x_{N[i,k]} - x_i) \bullet \nabla f(\boldsymbol{x}_i))^2) \tag{13}$$

and satisfies:

$$0 \le A(M,k) \le 1/2B^2 < \infty \tag{14}$$

where B is the upper bound on $|\nabla f|$ over the input space C given in condition **F.1**.

The term $O(M^{-1/2+\kappa})$ is due to sampling error. It may be shown that for a smooth positive sampling density φ the expected value of $\delta_M(k)$ is of order $O(M^{-2/m})$ as $M \to \infty$, where m is the dimension of the input space. Hence for $m \le 4$, it is possible that $\delta_M(k)$ is of smaller order than the sampling error and that linearity will not hold. In this case Algorithm 1 is effectively estimating the noise variance directly according to (8). Despite this, in practice we often see that linearity does hold when $m \le 4$ (see Figures 3 and 4 in which $m = 1$).

In general, the gradient *A(M,k)* may depend on the number of points *M* and also on the nearest neighbour index *k*. First we note that any dependence on *M* is of no consequence, since each regression point $(\delta_M(k), \gamma_M(k))$ is computed with *M* fixed. If the input space *C* is of *integral* dimension, it can be shown that $A(M,k)$ is independent of k and by (12), the points ($\delta_M(k)$, $\gamma_M(k)$) all lie on a straight line which intercepts the $\delta_M(k)=0$ axis at Var(r).

Potential problems arise in the case where C is of *nonintegral* dimension, where experimental evidence suggests that $A(M,k)$ does indeed depend on k. However, subject to a fairly weak condition[8] on the relative magnitude of $\delta_M(p)$ in terms of $\delta_M(1)$, by virtue of the fact that the gradient $A(M,k)$ is bounded it can be shown that the intercept of the regression line determined by the points ($\delta_M(k)$, $\gamma_M(k)$) approaches Var(r) as $M\rightarrow\infty$ and hence the method remains valid.

The Gradient as an Indicator of System Complexity

In many cases the gradient *A(M,k)* of the regression line contains useful information regarding the underlying system, represented by the unknown function f. To see this, consider:

$$((\boldsymbol{x}_{N[i,k]}-\boldsymbol{x}_i) \bullet \nabla f(x_i))^2 = |\boldsymbol{x}_{N[i,k]}-\boldsymbol{x}_i|^2\, |\nabla f(\boldsymbol{x}_i)^2|\, cos^2\theta_i \tag{15}$$

where θ_i is the angle between the vectors $\boldsymbol{x}_{N[i,k]}-\boldsymbol{x}_i$ and $\nabla f(\boldsymbol{x}_i)$.

Since the near neighbour vector $\boldsymbol{x}_{N[i,k]}-\boldsymbol{x}_i$ depends only on the sampling density φ, we may suppose that its magnitude $|\boldsymbol{x}_{N[i,k]}-\boldsymbol{x}_i|$ is independent of $|f(\boldsymbol{x}_i)|$. Furthermore, if the sampling density is approximately *isotropic* in small neighbourhoods of the points of C, the near neighbour distances $|\boldsymbol{x}_{N[i,k]}-\boldsymbol{x}_i|$ can be assumed to be independent of the *direction* of the near neighbour vectors[9].

Under these circumstances,

$$\mathbf{E}_\varphi(((\boldsymbol{x}_{N[i,k]}-\boldsymbol{x}_i) \bullet \nabla f(x_i))^2) = \mathbf{E}_\varphi(|x_{N[i,k]}-x_i|^2)\, \mathbf{E}_\varphi(|\nabla f(x_i)|^2\, cos^2\theta_i) \tag{16}$$

so we have

$$A(M,k) = 1/2\ \mathbf{E}_\varphi(|\nabla f(\boldsymbol{x}_i)|^2\, cos^2\theta_i) \tag{17}$$

In many cases it is also reasonable to suppose that $|\nabla f(\boldsymbol{x}_i)|$ is independent of θ_i so that

$$\mathbf{E}_\varphi(|\nabla f(\boldsymbol{x}_i)|^2\, cos^2\theta_i) = \mathbf{E}_\varphi(|\nabla f(\boldsymbol{x}_i)|^2)\, \mathbf{E}_\varphi(cos^2\theta_i) \tag{18}$$

Furthermore, if the angles θ_i are uniformly distributed over $[0,2\pi]$ then $\mathbf{E}_\varphi(cos^2\theta_i)=1/2$ and hence

$$A(M,k) = 1/4\ \mathbf{E}_\varphi(|\nabla f(\boldsymbol{x}_i)^2|) \tag{19}$$

Thus, particularly if the input/output variables are normalised to a standard range, we may consider the gradient of the regression line computed by the Gamma test as a crude estimate of the complexity of the system under investigation, proportional to the mean squared value of the gradient over the input space.

DATA ANALYSIS USING THE GAMMA TEST

We will now demonstrate how the Gamma test can assist with non-linear data analysis as a precursor to modelling. The techniques shown here are used to illustrate that non-linear data sets can be assessed for quality, and to demonstrate how the Gamma test can be used to find important features.

A Simple Example

We will illustrate the utility of the Gamma test using the following smooth function:

$$f(x) = sin(4\pi x) + cos(2\pi x) \tag{20}$$

Uniformly distributed noise with variance 0.03 was added to the function and sampled M=1000 times over the interval [0,1]. Figure 2 shows the underlying smooth function and the sampled "noisy" points.

We may visualise the Gamma test by considering the pairs

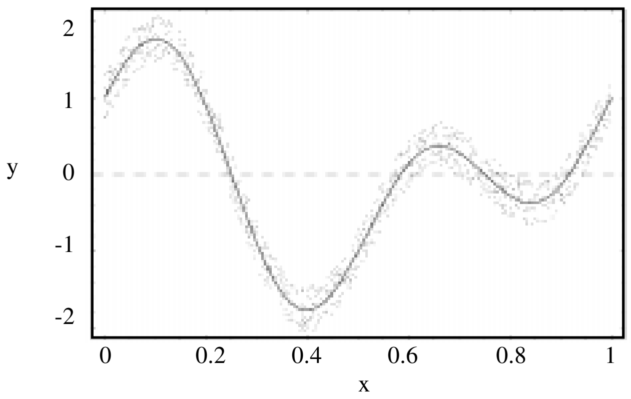
$$\begin{aligned}\delta &= |\boldsymbol{x}_i-\boldsymbol{x}_j|^2\\ \gamma &= 1/2\,|\boldsymbol{y}_i-\boldsymbol{y}_j|^2\end{aligned} \tag{21}$$

for each $1 \leq i \neq j \leq M$. Plotting each γ against the corresponding δ we obtain a scatter plot which visually indicates the noise level. Superimposed on this we plot $\gamma_M(k)$ and $\delta_M(k)$ defined in (6) and (7), and perform a linear regression through these points. The intercept with the axis at $\delta = 0$ gives the estimate for the variance of the noise.

Figure 3 shows a Gamma scatter plot for the smooth function (20) with no added noise. As expected, for a noise-free function we have $\gamma \to 0$ as $\delta \to 0$ and the estimate for the variance of the noise is 7.53 x10^{-7}, i.e., very close to zero.

When the Gamma test is run on the "noisy" data set, the form of the Gamma scatter plot changes. The effect of the noise is apparent in Figure 4 because as δ tends

Figure 2: The smooth function with sampled "noisy" points.



to 0, γ does not. Here, the estimate for the noise level produced by the Gamma test is equal to 0.031, close to the actual noise variance of 0.03.

This simple example illustrates that the Gamma test does indeed estimate the variance of the noise correctly, even though it has no knowledge of the underlying smooth function *f*.

Figure 5 further illustrates that the Gamma statistic Γ converges to the true noise variance as the number of points M increases. Indeed, for $M > 600$ we can be reasonably confident that the estimate lies within 10% of the true noise variance.

Feature Selection

Although the Gamma test is, in the first instance, an algorithm designed to estimate noise, it can be used very effectively to select relevant features for a non-linear model in both noisy and low or zero noise situations. The following examples will first illustrate why the Gamma test can be used to select relevant features in a zero noise case.

The technique of feature selection is used to extract useful information (or features) from a data set. Redundant or irrelevant variables in the data should be excluded. With the Gamma test we can define useful information as being those inputs that contribute to lowering the noise estimate of our input-output data set. In theory, the combination of inputs with the lowest noise estimate will provide the best model. In a mathematical context the features correspond to the *independent variables* and the output corresponds to the *dependent variable*.

Feature selection algorithms have two main components: a *criterion function* and a *search strategy* (Scherf & Brauer, 1997). The criterion function determines how good a particular feature set is and the search strategy decides which set to try next.

The search through feature space has to be performed to ensure that all (or as many) combinations of inputs are tested within reasonable computational time. For a small number, 20 say, of inputs, all possible combinations can be tested. However, this "exhaustive" search strategy quickly becomes impractical as the number of inputs increases. In general, for m inputs, there are 2^m-1 combinations of those inputs[10]. For larger data sets or for rapid feature selection, a heuristic search

Figure 3: Gamma scatter plot for the smooth function with no added noise.

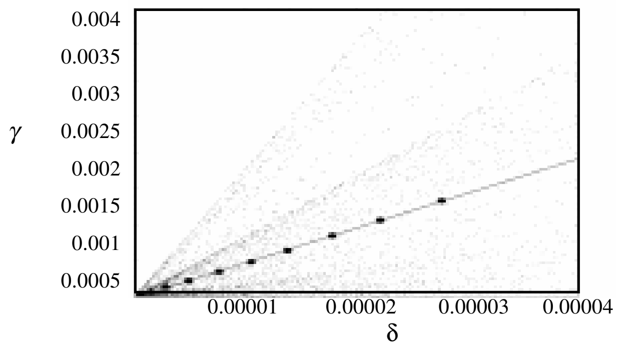


Figure 4: Gamma scatter plot for the smooth function with noise of variance 0.03 added.

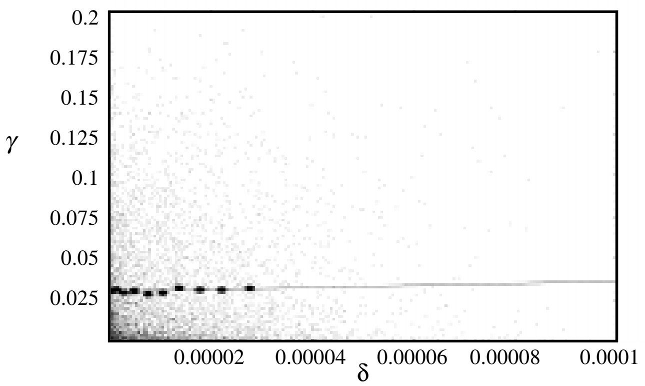
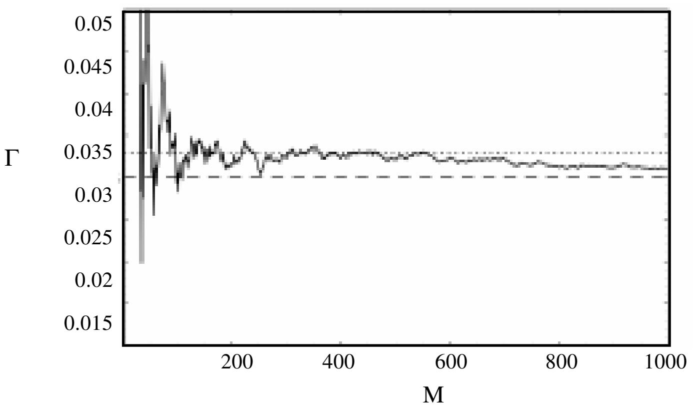


Figure 5: The Gamma statistic computed for increasing M. The dashed line indicates the true noise variance and the dotted line indicates a 10% error.



technique must be applied. The primary technique that we propose uses a genetic algorithm (Holland, 1975), but we have also implemented hillclimbing and other heuristics. We must recognise that these heuristic methods are not guaranteed to find the best possible feature set.

Whatever search strategy we choose, we clearly need an efficient criterion function. The two main types are *filters* and *wrappers* (Pfleger, John & Kohavi 1994). A wrapper uses a model to evaluate the feature set: the performance of a model constructed using the chosen features determines the significance of the feature set. The filter method does not rely on model building for the evaluation of a set of features. Instead, it uses the data directly to evaluate a given feature set. Our intention here is to show that the Gamma test performs this task and has other benefits, such as determining the variance of the noise. For reference, some other examples of filter methods are described in Cherkauer & Shavlik (1995).

Masks: Describing a Combination of Inputs

We describe feature sets in an efficient way using a *mask*. For any given data set, the inputs $(x_1,x_2,x_3, \ldots ,x_m)$ can be masked off to describe a subset of inputs. Using a $m = 4$ input data set as an example, we can describe the selection of input x_4 using the mask 0001, and the selection of all the inputs using the mask 1111. We use this representation within the context of feature selection to describe which inputs are used (1) and which are not (0). A complete feature space search requires all possible combinations of inputs to be analysed.

As an example of why the Gamma-test can be used for feature selection, we will consider a section through a three-dimensional cone. Five hundred data points were sampled uniformly in input space across the surface to produce a 3-dimensional data structure of two inputs (x,y) and one output z. Part of the cone is shown in Figure 6. The height of the cone z is dependent on the (x,y) coordinates. We should discover that using x or y alone will not determine z, but using x and y together will.

The darkly shaded projection onto the x-z plane in Figure 6 corresponds to the component part of the signal that is expected to act like noise when data is sampled from across the cone but where only the x input is used to model z. This *effective* noise is not uniform across the x-input space and the variation of noise variance as a function of x is shown in Figure 7. If we average this noise variance across the x input space we obtain the value 14.0126. Thus we might expect the associated Gamma statistic to be approximately this value. Similarly, if we project the cone onto the y-z plane (shown as the lighter shaded region in Figure 6) we see an even larger effective noise variance when sampling across the cone but using only input y to model z. These projections allow us to see geometrically that z is far more sensitive to variation in x than in y.

Figure 6: A conical function. The darkly shaded projection on the x-z plane shows the effective noise from sampling in the x-dimension only. The lighter shaded projection on the y-z plane shows the effective noise from sampling in the y-dimension only.

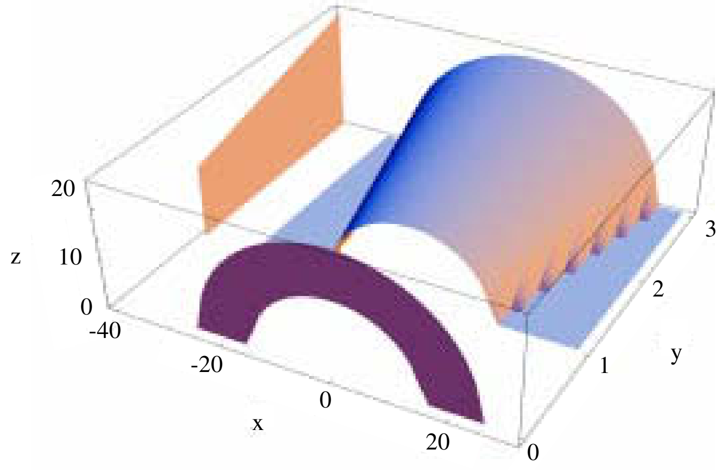


Table 1 lists the feature space search results. As expected, the effective noise variance was lowest when inputs *x* and *y* were used together. For the results where either *x* or *y* were exclusively used, the noise variance corresponds to the variance of *z* sampled in the interval over which *x* or *y* were also sampled. When input *x* is used to determine *z*, the estimated noise variance of Γ=14.76 closely corresponds to the average noise variance shown in Figure 7.

A 16-Dimensional Input Space (With No Noise)

The previous example was intended to give an intuitive understanding of why, even using noise-free data, the Gamma test results for different selections of input variables can be used to discriminate significant inputs for a non-linear model. In this section we illustrate that this procedure remains effective where functional dependences are subtler and many more input variables are present.

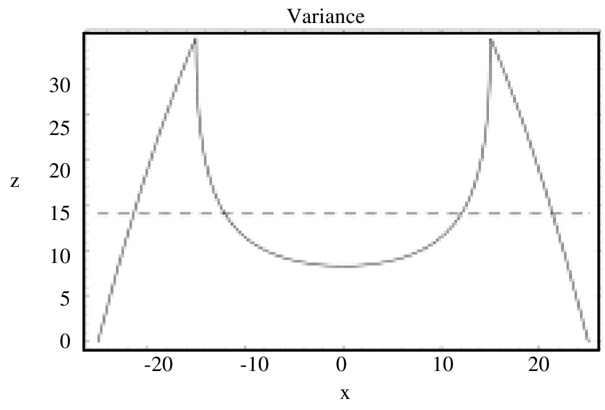
We consider *m*=16 inputs and 1 output. The first 10 inputs, $x_1, x_2, \ldots, x_{10}$, are all random numbers in the range [0, π]. The final 6 inputs are defined to be

$$\begin{aligned}
x_{11} &= \sin(2x_1)\\
x_{12} &= \cos(4x_2)\\
x_{13} &= \sin(x_3^2)+\cos(x_4^2) \qquad (22)\\
x_{14} &= e^{x_5}\\
x_{15} &= -x_6^2\\
x_{16} &= x_7^3
\end{aligned}$$

Table 1: Feature space search results for the cone section (M = 500)

Γ	*A*	*xy*
0.44217	11.257	11
14.76000	6.642	10
52.56900	4896	01

Figure 7: The effective noise variance of output z determined by input x. The dashed line indicates the average noise variance 14.0126 in the sampling interval.



The target output y is defined by the equivalent expressions

$$
\begin{aligned}
&y= sin(2x_1)\text{-}cos(4x_2)\\
&y= x_{11}\text{-}cos(4x_2)\\
&y= sin(2x_1)\text{-}x_{12} \qquad\qquad (23)\\
&y= x_{11}\text{-}x_{12}
\end{aligned}
$$

A set of $M = 5000$ points were generated and no noise was added to the output.

The output y is a relatively complicated function of inputs x_1 and x_2. There is a much simpler relationship between the output and x_{11} and x_{12}. There are also the intermediate relationships involving x_1 and x_{12}, and x_2 and x_{11}. In order to demonstrate the effectiveness of the Gamma test, the feature space search should discover these relationships. It should also highlight the simplest relationship as being the best.

The best results from the complete feature space search are shown in Table 2 where Γ represents the estimate for the variance of the noise and A is the gradient of the regression line fit.

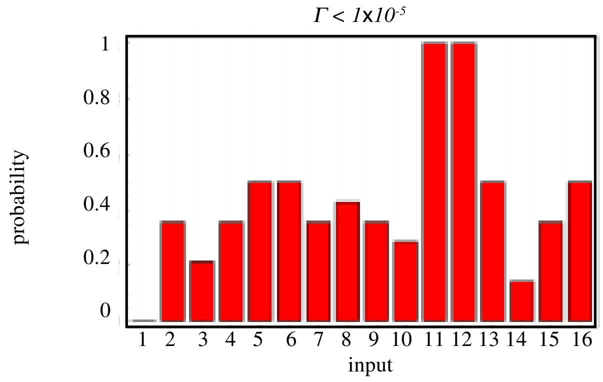
Inputs, x_1, x_2, x_{11} and x_{12} are underlined in the mask to highlight their expected significance as defined by (23). The results do show the importance of inputs x_{11} and x_{12} in determining the output y as they are each included[11] in *all* of the best results, defined to be those for which $\Gamma < 1\text{x}10^{-5}$.

The results in Table 2 therefore show that the Gamma test is able to select the

Table 2: Best results from a complete feature space search ($\Gamma < 10^{-5}$), $M = 5000$

Γ	A	Mask $\underline{x_1}\,\underline{x_2}\ldots\underline{x_{11}}\,\underline{x_{12}}\ldots x_{16}$
3.01x10^{-7}	0.142124	$\underline{00}00110100\underline{11}0011$
6.35x10^{-7}	0.12549	$\underline{00}01101010\underline{11}1100$
2.00x10^{-6}	0.0881483	$\underline{01}01111110\underline{11}1001$
2.49x10^{-6}	0.33071	$\underline{00}00010000\underline{11}0000$
4.08x10^{-6}	0.293724	$\underline{01}00000001\underline{11}0000$
4.15x10^{-6}	0.0955764	$\underline{01}00111101\underline{11}1001$
4.79x10^{-6}	0.506928	$\underline{00}00000000\underline{11}0000$
5.71x10^{-6}	0.149792	$\underline{00}01000010\underline{11}1001$
5.80x10^{-6}	0.17813	$\underline{00}00000101\underline{11}0010$
6.31x10^{-6}	0.0997976	$\underline{01}10101010\underline{11}1010$
6.36x10^{-6}	0.224083	$\underline{00}00000000\underline{11}1010$
6.86x10^{-6}	0.143837	$\underline{00}10110100\underline{11}0010$
8.70x10^{-6}	0.0910738	$\underline{01}11011100\underline{11}1100$
9.79x10^{-6}	0.107996	$\underline{00}01110011\underline{11}0001$

Figure 8: Frequency of the features for the best results ($\Gamma < 1\times10^{-5}$)



best combination of inputs. If we were to take any of the individual results we would include more inputs than are required to model the function. However, the necessary information would be incorporated as both x_{11} and x_{12} appear in all of these results.

The power of this technique is increased by the analysis of a *set* of results. If we examine the frequency of occurrence of each input in the set of best results given in Table 2, we are able to establish that only a small subset of inputs are actually relevant in determining the output. The frequency histogram of features shown in Figure 8 illustrates this.

In the following section we discuss this in more detail by analysing the Γ value corresponding to each of the feature sets encountered during the feature space search.

The Gamma Histogram

We construct a *Gamma histogram* to show the distribution of the noise variance estimates Γ over the space of feature sets. Using the complete feature space search of the previous section, we obtain the histogram shown in Figure 9.

Figure 9: Gamma histogram for a complete feature-space search over 16 inputs. The output of the function contained no noise so the histogram starts at $\Gamma \approx 0$.

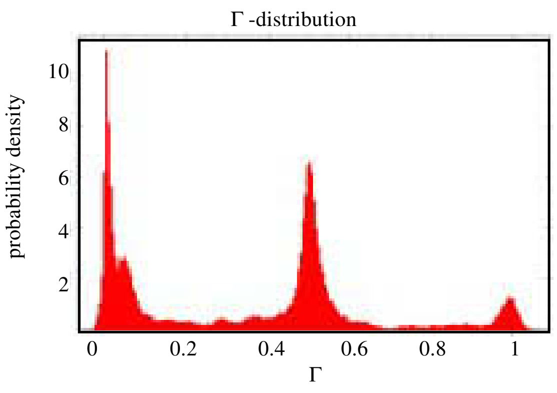
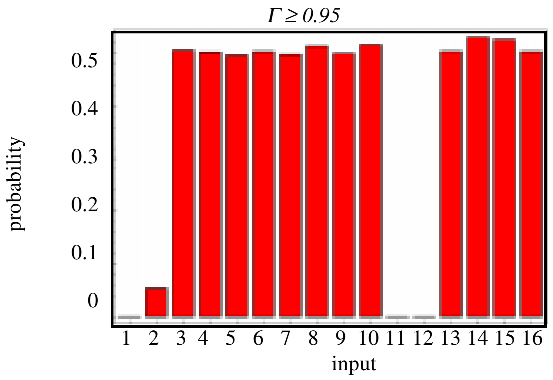


Figure 10: Frequency of the features for the worst results ($\Gamma \geq 0.95$).



There are six significant parts to this distribution:

1. The first peak, $\Gamma < 0.03$.
2. The second peak, $0.03 \leq \Gamma < 0.1$.
3. The space between the second and third peaks, $0.1 \leq \Gamma < 0.4$
4. The third peak, $0.4 \leq \Gamma < 0.6$.
5. The space between the third and fourth peaks, $0.6 \leq \Gamma < 0.95$.
6. The fourth peak, $\Gamma > 0.95$.

The first peak in the Gamma histogram consists of those features that produced results having $\Gamma < 0.03$. From the frequency graph shown in Figure 8, we see that x_{11} and x_{12} are the most important features in the low Γ region, corresponding to the two features that best predict the output. A similar frequency graph (not shown) for the second peak $0.03 \leq \Gamma < 0.1$ indicates that x_1 and x_{12} are the most important features in this region.

For higher Γ values, we may expect that less favourable but nevertheless useful features become apparent. A frequency graph for $0.1 \leq \Gamma < 0.4$ (not shown) indicates the significance of x_1 and x_2 in this region. From (23) we see that x_1 and x_2 are involved in more complex relationships with the output than the relationships between inputs x_{11} and x_{12} and the output.

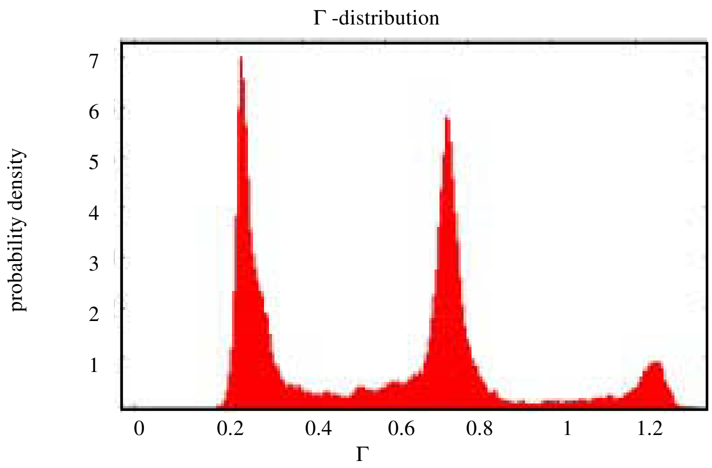
For high values of Γ we may expect favourable features to be noticeable by their absence. Indeed, the frequency graph for $\Gamma > 0.95$ shown in Figure 10 illustrates this, where we observe that x_1, x_2, x_{11} and x_{12} all occur with low frequency.

The overall conclusions are very striking: favourable features have a *higher* frequency than average in regions of low Γ and *lower* frequency than average in regions of high Γ.

A 16-Dimensional Input Space (With Added Noise)

Next we examine how the feature selection algorithm performs in the presence of noise. We adapt the previous example by adding noise of variance $\text{Var}(r) = 0.25$ to the output y. We remark that this is a relatively high noise variance considering

Figure 11: Gamma histogram of a complete feature-space search over 16 inputs with added noise having variance 0.25. Note that the histogram begins at a $\Gamma \approx 0.25$.



the size of the outputs (which are restricted to the range [-2,2]).

Comparing Figure 11 with Figure 9, we see how the addition of noise affects the Gamma histogram. The histogram is offset from the origin by an amount approximately equal to the noise variance. However, again we have three major peaks which are in roughly the same relative positions as in the noise-free case. We now examine these peaks and show that the selection of relevant features as determined by their relative frequency in the high and low peaks of the Gamma histogram remains essentially unchanged in the presence of noise.

The first peak contains the feature sets that produced results having $\Gamma < 0.3$. A frequency graph of these features is shown in Figure 12. As in the noise-free case of Figure 8, this again shows that inputs x_{11} and x_{12} are the most significant features since they appear in most of the results. The remaining inputs appear with approximately equal frequency (with the exception of x_1 which appears slightly

Figure 12: Frequency of the features for the best results ($\Gamma < 0.3$).

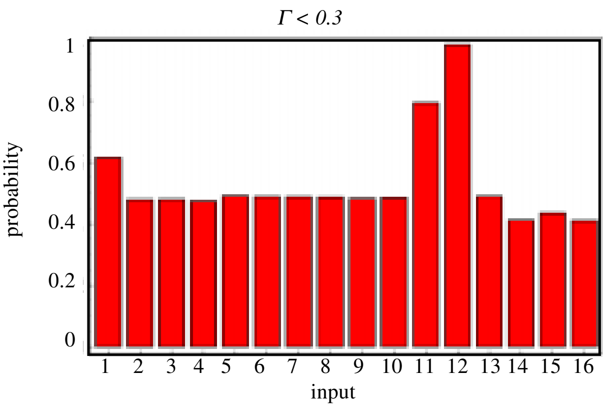
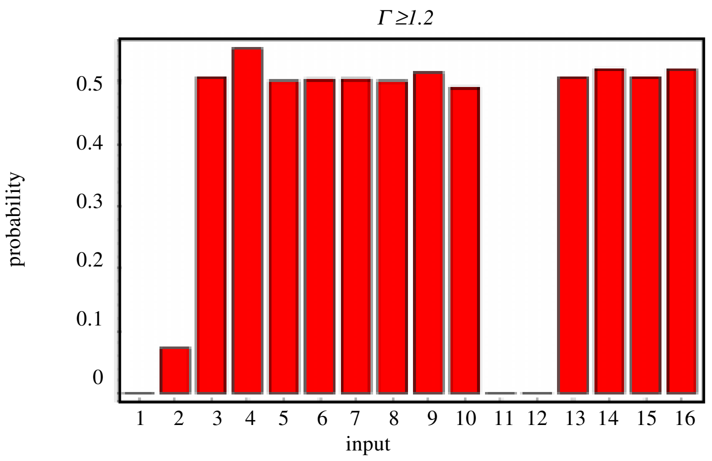


Figure 13: Frequency of the features for the worst results ($\Gamma \geq 1.2$).



more often) and on this evidence we may again conclude that x_{11} and x_{12} provide the information necessary to model the output y.

As before, a frequency graph (not shown) for the intermediate region $0.3 \leq \Gamma < 0.7$ indicates that the feature combination of x_1 and x_2 is also significant. Again, as in the zero noise case of Figure 10, the frequency graph for $\Gamma > 1.2$ shown in Figure 13 provides additional evidence that inputs x_1, x_2, x_{11} and x_{12} are significant, demonstrated by the absence of these inputs in the worst results.

These experiments demonstrate that feature selection via a Gamma test analysis remains effective in the presence of noise. For fixed M, as the noise increases, the Gamma histogram gradually degrades as a feature selection tool, but it does so gracefully. For high levels of noise we can still apply this technique, but it may be necessary to increase the number of points M accordingly.

In conclusion, a peak at the lower end of the Gamma histogram should contain results that use all of the available relevant input variables. A peak at the higher end of the Gamma histogram should show results generated from input variables that have little or no relevance in determining the output (Durrant, 2001).

TIME-SERIES MODELLING

One important application area of these techniques is that of modelling time series data. This can arise in many diverse situations from predicting sunspot activity (Tsui, 1999; Tsui, Jones & Oliveira, 2000) to the analysis of economic or financial time series data.

If the underlying process is a *dynamical system*, where the behaviour is determined by a system of differential equations, then we may construct a smooth non-linear input/output model that generates new states of the system based on a finite window of previous states (see Takens,1981), or the later extensions by Sauer, Yorke and Casdagli, 1991). Of course the extent to which economic or financial time series data reflects underlying smooth dynamics is, in many instances, somewhat conjectural but it is customary to use the same approach to modelling such data and the Gamma test can at least tell us the extent to which such an assumption is justified in practice.

We will now illustrate how the Gamma test can be used for feature selection in a zero noise chaotic time series. In this case, feature selection corresponds to the determination of appropriate lags in an embedding model, and the sampling distribution over the input space corresponds to sampling an attractor with fractional dimension. In this context the Gamma test can be applied first to estimate an appropriate embedding dimension, i.e., the number of past values of the time series to be used as inputs in predicting the next value (the output). Then, once the embedding dimension has been fixed, it can be used to determine an appropriate irregular embedding, i.e., an appropriate subset of these past values

We first compare the initial estimate of the embedding dimension as calculated using the Gamma test using an "increasing embedding" algorithm, with that found by the more conventional *false nearest neighbour* algorithm (Kennel, Brown & Abarbanel, 1992). We shall see that both approaches give similar results.

False Nearest Neighbours

The *false nearest neighbour* (FNN) algorithm (Kennel, Brown & Abarbanel, 1992) is a technique to determine the embedding dimension for phase-space reconstruction. A chaotic attractor is typically a compact object in phase-space, such that points of an orbit on the attractor acquire neighbours. If the embedding dimension of an attractor is sufficient there will be a one-to-one mapping from the delay-space (the time series) to the original phase-space of the attractor such that the topological properties of the attractor will be maintained.

The assumed smoothness of the function means that neighbourhoods of points in delay-space will map to neighbourhoods of points in phase-space. An embedding dimension that is too small will not preserve the topological structure of the attractor, so that points that are neighbours in one embedding dimension, m, will not necessarily be neighbours in the next higher embedding dimension, $m+1$, because the attractor has not been completely unfolded. It is these points that are classified as *false nearest neighbours,* and the number present for a particular embedding dimension determine whether that embedding dimension, m, sufficiently describes the attractor. The FNN algorithm identifies these points for a range of embedding dimensions, and (in theory) the optimal embedding dimension has the minimum number of false nearest neighbours.

Increasing Embedding

An *increasing embedding* technique based on the Gamma test can perform the same task of identifying the optimal embedding dimension.

Suppose we have a time series consisting of M points $x_1, ..., x_M$. For each embedding dimension m ranging from 1 to some predefined maximum, we construct the set of m-1 dimensional delay vectors defined by:

$$z_i = (x_i, ..., x_{i+m-1}) \tag{24}$$

We use the Gamma test to measure how smoothly the vectors z_i determine the "next" point x_{i+m} of the time series. The value of m for which the Gamma statistic is closest to zero is taken to be the optimal embedding dimension of the time series.

Example: The Henon Map

The Henon map is generated iteratively by the equation

$$x_t = 1 - ax^2_{t-1} + bx_{t-2} \tag{25}$$

where $x_0 = 0$, $x_1 = 0$, $a = 1.4$ and $b = 0.3$ (these values are known to generate a chaotic time series).

Figure 14 shows the first 50 points of this time series. The points of the map are sampled from the *attractor* of the map which can be extracted from the time series data and visualised by simply plotting the inputs to the function against the output as shown in Figure 15. At the bottom of the diagram (in the 3-dimensional representation), the relationship between the output x_t and the input variables x_{t-1} and x_{t-2} is shown (this is the hypothetical surface that we may seek to construct in a modelling exercise). At the top of the diagram a projection shows the distribution of the input variables (x_{t-1}, x_{t-2}).

Figure 16 shows the results from the FNN algorithm, and Figure 17 shows the increasing embedding on the time-series data generated from the Henon map. The optimal embedding dimension selected by both methods is equal to 2.

Hence the increasing embedding algorithm can be used to recover the embed-

Figure 14: The first 50 points of the Henon time series x_t against t

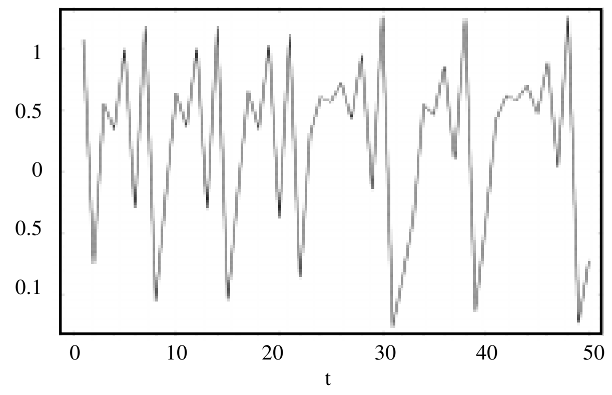


Figure 15: The Henon attractor with no added noise. The input space sampling is shown above the attractor.

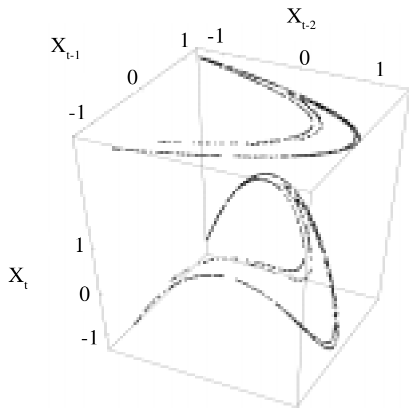
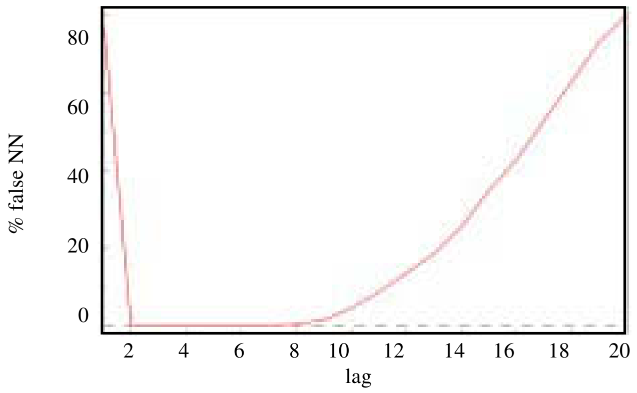


Figure 16: False nearest neighbour embedding for the Henon map. The graph shows that an embedding dimension m=2 would be suitable.



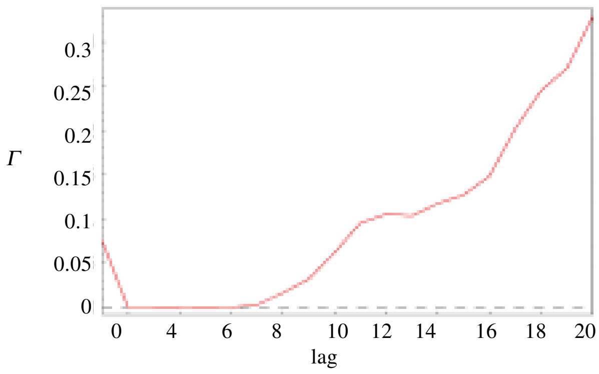
ding dimension of a time series in much the same way as the FNN algorithm.

Irregular Embeddings

For a more complex time series, once the embedding dimension has been fixed the Gamma test can be used to select an appropriate irregular embedding using a full embedding search—in much the same way that we selected features in the 16-dimensional example given earlier.

Although it might appear that choosing an optimal subset of the input variables should not be critical, since in principle all the information required is present if the embedding dimension has been correctly estimated, nevertheless in practice correct choice of an irregular embedding, with a consequent reduction in the number of input variables, can make a significant improvement to both model quality and training time.

Figure 17: Increasing embedding for the Henon map. The graph shows that an embedding dimension d between 2 and 7 may be suitable. Empirical evidence suggests that selecting the lowest embedding dimension from a range of possible solutions is best, in this case m = 2.



winGamma™

The results above were generated with the help of *winGamma*™. This is an easy-to-use non-linear data-analysis and modelling package produced by Cardiff University as a Windows application. Its primary use is as a research and teaching tool for academics and students, but it is also available as a commercial product for data modellers requiring state-of-the-art analysis and modelling techniques[12].

FUTURE TRENDS

Data Mining With the Gamma Test

winGamma™ was constructed as a non-linear analyst's workbench, and as with any such tool there is a learning curve which must be ascended to acquire the necessary skills to apply the tool effectively. However, as we have gained more experience in the use of *winGamma*, and began to develop an analysis protocol, it has become apparent that the analysis process could be *automated* with little loss of effectiveness.

Because the Gamma test runs extremely quickly one can therefore envisage a more sophisticated program (*GammaMiner*) which automatically scans large databases looking for relationships between numerical fields which can be used for smooth modelling and prediction. The user could define which attributes were of particular interest (the *targets* or *outputs* required to be predicted) and which other attributes the targets might reasonably depend on (these would form the set of potential inputs to the model). Designing such a program is not without pitfalls. For example, attribute values may not be time-stamped and one could easily find the program "predicting" values that predate the attribute values used as inputs. There are some problems regarding database semantics that need to be addressed. Because not all data falls into the category of numerical fields which might be modelled by a smooth function and because other types of tools (e.g., decision trees) may be more appropriate for constructing predictive models on discrete inputs or categorical outputs, one could also envisage engineering a subset of *GammaMiner* as a re-usable *component* designed to be integrated into existing or future data mining tools.

Nevertheless, it is possible to imagine such a program running continually in the background and notifying its owner only when it found something interesting; e.g., "By the way I have a predictive model for X for one month ahead which gives an accuracy of 0.5% are you interested?" While such program behaviour is arguably not intelligent in any real sense, there is no doubt that such a tool would be useful.

Many users of *winGamma* are explicitly interested in time series prediction of economic data. Here the most promising approach seems to be to bring to bear user domain knowledge to determine which other available time series data can act as *leading indicators* for the target time series. We propose in the first instance to provide a set of time-series editing tools that facilitate the alignment in time of attribute values from different time series and the selection of subsets of lagged data to be explored by *GammaMiner* in seeking to evaluate predictive capability.

The *GammaMiner* project seeks to prototype an automated model extraction capability with special reference to time series. Whilst it is indeed possible that genuine new scientific knowledge might result from the use of such a program, it is worthwhile to reflect briefly on the scientific value of such opportunistic model building.

The Status of Data-Derived Predictions

When physicists try to make predictions they are following one of the basic principles of science:

- Postulate the basic laws—they are supposed to hold for all time and in all places.
- Calculate the consequences.
- Perform experiments or observations to verify the predictions. Successful verification does not constitute a "proof" of the law but failure to verify might constitute a disproof (if all the loopholes in the logic have been plugged).

The philosophical study of our sources of knowledge is known as *epistemology*. Since the laws of physics are supposed to be invariant over all time and space we could say loosely that physics espouses a Platonian view of knowledge in which the "laws" are there and fixed and it is up to us to discover them, usually in some very pure mathematical form.

- The *advantage* of having such laws available is that because they are supposedly invariant over time and space one may be able to make predictions for circumstances that have never been observed before—we call this extrapolation.

- The *disadvantage* is that sometimes the calculations directly from the laws (or "first principles" as it is often called) may be so complicated or take so long as to be impractical.

The barrier that often presents itself is one of computational complexity. As a simple example consider the *protein-folding problem.* A big protein has thousands of constituent atoms and we might know its atomic structure exactly. The biological action of the protein is what we would like to predict. Now if you were to hold the protein by both ends and let go it would collapse into something that on the right scale would look like a tangled ball of wool. The biological action of the protein is largely determined by what is left on the outside of the ball of wool. So the problem is simple: we know the effects of atomic bonds, we know the structure so let's just plug all this into a computer program and compute the folded structure. That sounds good, but except for fairly small molecules it can't be done—the program takes too long to run. But things are even worse than this!

Indeed even *without* the Heisenberg uncertainty principle (which says you cannot measure both the position and momentum of a particle with an arbitrary degree of precision) the universe *a la* Newton really contained the seeds of its own destruction. Even if you know all the initial conditions of some quite simple chaotic process exactly, then the amount of computation required to predict a long way into the future with the fastest computer one could imagine would still *require a time greater than the estimated life expectancy of the universe*.

This is the first lesson of chaos. An example is the weather—where we know all the laws and can measure to our heart's content but we cannot even predict reliably several days into the future, let alone several months. But there are other, more pragmatic, approaches. When we talk about "predicting the future" in this context we have a rather cavalier approach in mind—a kind of *opportunistic epistemology* which runs more along the following lines

- A model is "good" just as long as it is predicting well. When it stops predicting well, we just try to build another model.

This is because we come at the question of prediction from an artificial intelligence perspective. What we need are predictive models which *work* and which c*an be computed rapidly*. Of course, the extent to which economic or sociological models discovered by application of tools such as the Gamma test are truly scientific depends on the context.

CONCLUSIONS

The Gamma test is an interesting new tool for model discovery and these ideas have already been used in diverse applications. Examples are:

- Chuzhanova, Jones and Margetts (1998) use the Gamma test to produce a fast algorithm for feature selection and hence classifications of the large subunits of ribosomal RNA according to RDP (Ribosomal Database Project) phylogenetic classes. In plain language: to quickly recognise the species described by a DNA sequence.

- Connellan and James (1998) and James and Connellan (2000), use the Gamma test to model short term property prices based on the time series of valuations and 15-year gilts (British Government Securities).
- Oliveira (1999) shows how these techniques can be used to extract digital signals masked by a chaotic carrier.
- We are developing the application of these ideas to a Modular Automated Prediction and Flood Warning System (*MAPFLOWS*) for the non-linear modelling of river systems.
- We expect to see them applied to many other problems of prediction and control.

One very interesting and potentially extremely beneficial application of these ideas, and other techniques for dealing with discrete input and output variables, is to large medical databases. A not inconsiderable problem in this respect is that even anonymised data is protected to an extent that hampers access for such blanket research approaches.

REFERENCES

Aoalbjörn Stefánsson[13], Koncar, N. & Jones, A. J. (1997). A note on the Gamma test. *Neural Computing & Applications*, 5, 131-133.

Cherkauer, K. J., & Shavlik, J. W. (1995). Rapidly estimating the quality of input representations for neural networks. In IJCAI-95 Workshop on Data Engineering for Inductive Learning.

Chuzhanova, N. A., Jones, A. J. & Margetts, S. (1997). Feature selection for genetic sequence classification. *Bioinformatics, 14(2)*, 139-143.

Connellan, O. P. & James, H. (1998). Forcasting Commercial Property Values in the Short Term. RICS Cutting Edge Conference, Leicester, 1998. RICS London. Available electronically from RICS.org.uk

Durrant, P. (2001). winGamma™: A non-linear data analysis and modelling tool with applications to flood prediction. Ph.D. thesis, Department of Computer Science, University of Wales, Cardiff, Wales U.K. Available in PDF from http://www.cs.cf.ac.uk/user/Antonia.J.Jones/.

Evans, D. (2001). The Gamma test: data derived estimates of noise for unknown smooth models using near neighbour asymptotics. Ph.D. thesis, Department of Computer Science, University of Wales, Cardiff, Wales, U.K.

Friedman, J. H., Bentley, J. L. & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software, 3(3)*, 200-226.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* MIT Press.

James, H. & Connellan, O. P. (2000). Forecasts of a Small feature in a Property Index. Proc. RICS Cutting Edge Conference, London, 2000. RICS London. Available electronically from RICS.org.uk

Kennel, M. B., Brown, R., & Abarbanel, H. D. I. (1992). Determining embedding dimension fŏr phase-space reconstruction using a geometrical construction. *Physical Review A, 45(6)*, 3403-3411.

Koncar N. (1997). Optimisation methodologies for direct inverse neurocontrol. Ph.D. thesis, Department of Computing, Imperial College, University of London, U.K.

Oliveira, A. G. (1999). Synchronisation of chaos and applications to secure communications. Ph.D. thesis, Department of Computing, Imperial College, University of London, U.K., 1999.

Pfleger, K., John, G., & Kohavi, R. (1994). Irrelevant features and the subset selection problem. In W. W. Cohne and H. Hirsh (Ed.), *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 121-129).

Sauer, T., Yorke, J. A., & Casdagli, M. (1991). Embedology. *Journal of Statistical Physics, 65(3)*, 579-616.

Scherf, M. & Brauer, W. (1997). Feature selection by means of a feature weighting approach. Technical Report FKI-221-97, Forschungsberichte Kunstliche Intelligenz, Institut für Informatik, Technische Universitat, Munchen.

Takens F. (1981). Detecting strange attractors in turbulence. In D. A. Rand and L. S. Young (Ed.), *Dynamical Systems and Turbulence* (pp. 366-381). Lecture Notes in Mathematics, Vol. 898, Springer-Verlag.

Tsui, A. P. M. (1999). Smooth data modelling and stimulus-response via stabilisation of neural chaos. Ph.D. thesis, Department of Computing, Imperial College, University of London, U.K. Available in PDF from http://www.cs.cf.ac.uk/user/Antonia.J.Jones/.

Tsui, P. M., Jones, A. J. & Oliveira, A. G. (2000). The construction of smooth models using irregular embeddings determined by a Gamma test analysis. Paper accepted by *Neural Computing & Applications*, 2001.

ENDNOTES

1 This process is known as the *M*-test

2. In practice on a 300 MHz PC with $M \approx 1000$ and $m \approx 50$ a single Gamma test computation is complete within a few seconds.

3 The support is defined to be the set of points where the probability density function is non-zero.

4 A formal proof of the method also requires that both the third and fourth moments of the noise distribution are finite.

5 In practice we consider the possibility that several points may be equidistant from a given point and maintain the index $N[i,k]$ as a list where each of these points are considered to the *k*th nearest neighbour of the given point. This is done in order to detect unusual distributions of input data.

6. Note that $y_{N[i,k]}$ is not necessarily the *k*th nearest neighbour of y_i.

7. Of course, other metrics are possible and which is most effective may be problem dependent, but since any two metrics in a finite dimensional space are equivalent to within a positive constant, the asymptotics of interest are

essentially invariant with respect to a change of metric.

8 The condition requires that for some fixed p, $0 < c < 1$ such that $\delta_M(1)/\delta_M(p) \leq c < 1$ as $M \to \infty$. This enforces a reasonable "spread" of the $\delta_M(k)$ values.

9 This is certainly the case for a smooth positive sampling density φ over C.

10 To perform a full search of 10 inputs requires 1,023 Gamma test experiments, whereas 1,048,575 experiments are required for 20 inputs.

11 A 1 in the mask indicates the inclusion of the input in the calculation and a 0 indicates exclusion.

12 See http://www.cs.cf.ac.uk/wingamma for more information about *winGamma*™.

13 This name is of Icelandic extraction and so the first name must be given in full.

SECTION FOUR

NEURAL NETWORKS AND DATA MINING

Chapter X

Neural Networks— Their Use and Abuse for Small Data Sets

Denny Meyer
Massey University at Albany, New Zealand

Andrew Balemi and Chris Wearing
Colmar Brunton Research, New Zealand

Neural networks are commonly used for prediction and classification when data sets are large. They have a big advantage over conventional statistical tools in that it is not necessary to assume any mathematical form for the functional relationship between the variables. However, they also have a few associated problems, chief of which are probably the risk of over-parametrization in the absence of P-values, the lack of appropriate diagnostic tools and the difficulties associated with model interpretation. These problems are particularly pertinent in the case of small data sets. This chapter investigates these problems from a statistical perspective in the context of typical market research data.

INTRODUCTION

McCulloch and Pitts (1943) are credited with generating the first interest in neural networks (in the nervous system) but it was not until the 1980s that technology allowed the rapid development of neural networks for solving application problems in numerous fields. The reader is referred to Haykin (1999) for a comprehensive coverage of this evolution.

With something akin to horror, statisticians have been watching this recent growth in neural network popularity. Mackinnon and Glick (1999) are particularly concerned by the "black box" or "computational algorithm-oriented" nature of neural network rules. It is difficult to trust a model that is not transparent (i.e., cannot be interpreted). Some of the concern is fuelled by the common (mis)conception that neural networks are about automating data analysis and data modelling (Elder & Pregibon, 1996). Model selection is regarded as a vital part of the statistician's job and to automate this function may seem threatening to statisticians. However, Chatfield (1995) has warned that statisticians have yet to confront the issues surrounding model selection. In particular he points out that the errors caused by model misspecification are likely to be far worse than those arising from other sources. He recommends that statisticians should allow for model uncertainty by averaging over several plausible models or *by choosing a flexible procedure* (such as neural networks) *which does not force a particular form of model on the data.*

Statisticians are being encouraged to test neural networks with typical (small) data sets (Cheng & Titterington, 1994; Warner & Misra, 1996). Although the jury is still out, it seems that, although neural networks need to be applied carefully and creatively (Brierley & Batty, 1999), they have much to offer statisticians as "one of a class of flexible nonlinear regression methods" (Ripley, 1994, p.409). Many studies have focused on a comparison of conventional methods with neural networks (e.g. Cooper, 1999; Haykin, 1999; Ripley, 1996) and several authors have concluded that neural networks should be used in conjunction with conventional statistical tools.

For example, Faraway and Chatfield(1998) suggest that for time series models the Bayesian Information Criterion (Schwarz, 1978) should be used for comparing different models. Borggaard (1995) comments on the advantages of using major principal component scores instead of numerous raw input variables for developing neural network models. Having only a few variables results in smaller networks, which are quicker to train and easier to optimise in a global sense. Markham and Ragsdale (1995) have found that neural networks do not always outperform classical discriminant analysis as a classification tool and advise that a combination of classical and neural network predictions is more accurate. Haykin (1999) recommends the use of cross-validation (Stone, 1974, 1978), to prevent over-parameterisation of neural network models.

From the above it appears that standard statistical tools can be used to improve neural network models. In this chapter we explore this idea further while trying to fit three neural network models using small data sets and typical commercial software. In these models we confront the problems of over-parameterisation, diagnostics and interpretation for small data sets, suggesting how the range and power of commercial neural network software can be enhanced. But this chapter makes no attempt to review the many exciting theoretical developments in this field. Readers are referred to the excellent books of Haykin (1999) and Bishop (1995) for this purpose.

NEURAL NETWORK METHODOLOGY

The neural networks used in this chapter are artificial in that they are algorithms rather than real neural networks, such as those found in the brain. Maindonald (1998) describes such a neural network as "a mathematical model for a learning process". The networks considered here are commonly referred to as multilayer perceptrons, in that they are organised hierarchically into layers of neurons or nodes. The first layer corresponds to the input variables and the last layer corresponds to the output variables, one node for each variable. In the case of small sample sizes only one intermediate hidden layer of nodes should be used. Adding additional hidden layers increases the risk of over-parameterization because each additional hidden node increases the number of parameters associated with the neural network model. However, there must be at least two nodes in the hidden layer in order for a non-linear model to be fitted.

Figure 1 shows the form of a very simple neural network. In Figure 1 the nodes i_1 and i_2 denote two input or explanatory variables while o indicates a single output or response variable. The second column of nodes (circles) represent the hidden nodes (h_1 and h_2). Associated with each pair of nodes in succeeding layers is a weight (w). These weights are used to define a linear function of input variables for each node in the hidden layer. The weights associated with the "-1" nodes are the constant terms.

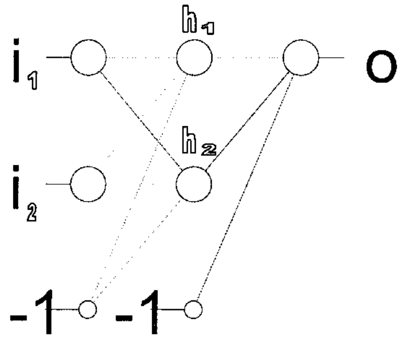
Figure 1: What is a neural network?

Equations for neural networks when there is a layer of input nodes (i_1 and i_2), a single layer of hidden nodes (h_1 and h_2) and a single output node (o).

$$h_k = f\left(\sum_j w_{jk} i_j\right)$$

$$\hat{o} = \sum_k w_k h_k$$

Optimise the weights (w) by minimising $\sum(o - \hat{o})^2$



The linear functions of input variables are transformed using an activation function (f), which acts as a filter in that it usually outputs values close to zero or one, rather than something in between. Sigmoid and hyperbolic tangent functions are commonly used as activation functions. When a neural network is trained the weights are changed using an iterative procedure which seeks to minimise the difference between the observed and predicted values for the final layer of variables. The initial set of weights is chosen randomly so it is advisable to check that the final solution is a global optimum rather than a local optimum. This can be done by using several sets of initial weights and comparing the final models in each case.

There are numerous procedures for attaining the optimum weights, but the backward propagation algorithm of McClelland, Rumelhart and PDP Research Group (1986), employed in this chapter, is commonly used in conjunction with a steepest ascent or conjugate gradient search algorithm (Bishop, 1995). Once the weights have been optimised, the required output values can be predicted.

Overfitting, which is a particular problem for small data sets, is addressed by using a validation data set. The training data set is used to determine what weight adjustments are required. However, these adjustments are made only if they produce an improvement in predictive accuracy for the validation data set. Haykin (1999) and Bishop (1995) describe how over-parameterisation can be avoided by introducing a complexity penalty, which works to reduce or eliminate unnecessary coefficients in the neural network model, and Hall and Holmes (2000) compare various methods for ranking the importance of the input variables. Alternatively linear stepwise variable selection methods can be used to select only the most important input variables. Finally, visual checks of the resulting neural network models are needed in order to confirm that the models are sensible.

When sample sizes are very small, optimising the predictive accuracy of the neural network model in terms of the number of hidden nodes is often not an option, and the maximum number of hidden nodes is two for a prediction problem and k for a classification problem with k classes. An interpretation of the hidden nodes is essential because the salient features of the data are developed within the hidden nodes. For each hidden node the activation function acts as a filter for different types of information. The input weights associated with each hidden node determine what type of information it filters. The identification of this filtered information for each hidden node is known as rule extraction. Further information on this topic can be found in Brierley & Batty (1999), Maire (1999), and Andrews, Diederich and Tickle (1995).

Two data sets are used in the analyses that appear below. The first data set concerns the market shares, prices, advertising and promotion in a cold drink market during a 61-week period. We shall use these data to model the market share for three cold drink brands. Such models are useful for predicting the effects of price changes, advertising and promotion decisions on market share. In the second data set, demographic and attitudinal data for 2,010 people are used to develop classification rules for six market segments. These market segments are needed for the targeting of advertising and for product repositioning exercises. Independent weekly samples are used to track the performance of brands and advertisements for each of these segments. In both of the above analyses, the accuracy of conventional linear models is compared with the accuracy of non-linear neural network models, with stepwise linear selection procedures being used to identify the most important input variables. The data were standardised with a mean of zero and a standard deviation of one in order to permit a meaningful comparison of weights.

DATA ANALYSIS

Our analysis is performed from the perspective of a business that produces three cold drink brands. Two of these brands (True Treat and Northern Delight) compete at the low end of the youth cold drink market, while the third brand (Burgen Broth) is a health drink competing at the top end of what constitutes a different market. The youth and health cold drink markets are distinctly different in that the youth cold drink market is extremely price sensitive while the health cold drink market is not sensitive to price. The effect of a change in price is immediate for the youth cold drink market so it is not necessary for us to consider any time lags in this market share model. The health cold drink market is sensitive to advertising expenditure, especially TV advertising, but the effect of this advertising is not necessarily immediate, suggesting that we need a lagged model when we predict the market share for health cold drinks. We start by considering models for predicting market share and then consider a customer classification model for this market.

Market Share Model for True Treat and Northern Delight

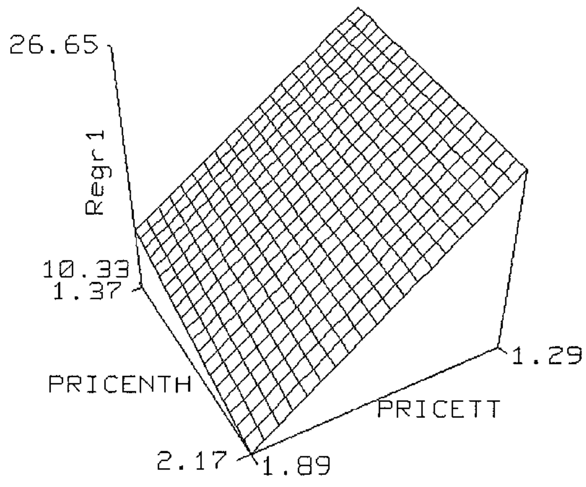
In the case of the youth cold drink market there were 11 variables which were thought to affect the relative market shares. The market share for our business, supplying True Treat (TT) and Northern Delight (NTH), was modelled in terms of these 11 variables using a regression model and a neural network model. As expected the linear regression model indicated an over-parameterised model with the majority of the coefficients (weights) insignificant. In addition it suggested an outlier in a week during which prices for one brand had been cut below cost. Removing the insignificant variables and the outlier produced what Nelder (1999) would call "an inferentially uninteresting linear model."

$$SHARE = 57.2 - 19.3(PREICETT) - 4.67(PRICENTH)$$

as illustrated in Figure 2. The price of the TT brand obviously has a strong influence on market share while the price of the NTH brand has a slight effect, with an increase in either of these prices reducing market share.

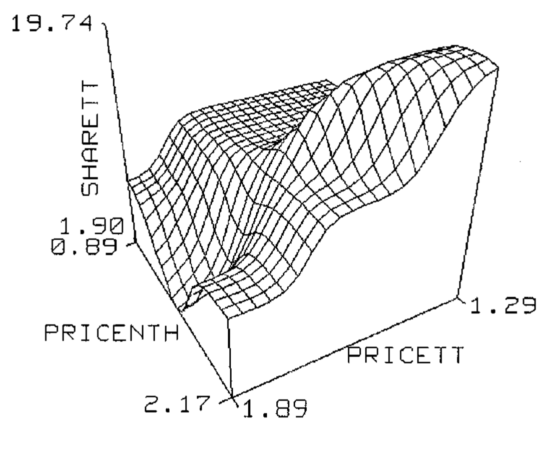
Applying a neural network to these same data in order to predict the market share for True Treat (TT) and Northern Delight (NTH) separately, our software

Figure 2: Regression model: Effect of prices on market share (TT + NTH)



package used the number of inputs and the relationships between these inputs to choose four hidden nodes. No P-values were obtained in the output, and no information regarding outliers was automatically supplied. The model for TT market share, as illustrated in Figure 3, was anything but uninteresting. The idea of market saturation levels (when falling prices failed to affect market shares) was particularly interesting, but the pleat in the surface at high TT prices suggested over-parameterisation. In this case we had 53 weights (44 coefficients for the inputs, 4 coefficients for the hidden layer nodes and 5 constant coefficients) to be estimated from 61 observations. Hair, Anderson, Tatham and Black (1998) advise that when fitting linear regression models there should be at least 5 but preferably 15 observations for every linear regression coefficient (weight) estimated, in order to avoid over-parameterization. A similar rule is probably appropriate for neural

Figure 3: Over-parameterized neural network model: Effect of prices on share



networks.

Radical pruning of the model was required. After removing the outlier identified previously, the number of hidden nodes was reduced to the minimum required for a nonlinear model (i.e., two). Instead of trying to predict the market share for both products, the output variable was defined as the market share for True Treat and Northern Delight combined. The number of predictor variables was reduced to three, PRICETT and PRICENTH and an indicator variable called PROBLEM. The PROBLEM variable defined the weeks in which a major problem on the manufacturer's premises had affected sales. The following equations were obtained for the hidden nodes using a training data set of 54 randomly chosen weeks and a validation data set consisting of the remaining 6 weeks. In these equations x_1=PRICETT, x_2=PRICENTH, x_3=PROBLEM, with means and standard deviations for these variables shown as $\bar{x}$ and s.

$$h_1 = \tanh\{0.04 - 0.35\left(\frac{x_1 - \bar{x}_1}{s_1}\right) + 0.12\left(\frac{x_2 - \bar{x}_2}{s_2}\right) - 0.30\left(\frac{x_3 - \bar{x}_3}{s_3}\right)\}$$

$$h_2 = \tanh\{-0.08 + 1.52\left(\frac{x_1 - \bar{x}_1}{s_1}\right) + 0.43\left(\frac{x_2 - \bar{x}_2}{s_2}\right) + 0.44\left(\frac{x_3 - \bar{x}_3}{s_3}\right)\}$$

The coefficients (weights) in these equations suggest that the first hidden node measures the effect of PRICENTH as opposed to PRICETT and PROBLEM, while the second hidden node measures the effect of all three of these variables with most weight given to PRICETT. The final market share for both the True Treat and Northern Delight brands, (y = SHARE) was predicted using the following equation in which $\bar{y}$ and s_y denote the mean share and the share standard deviation.

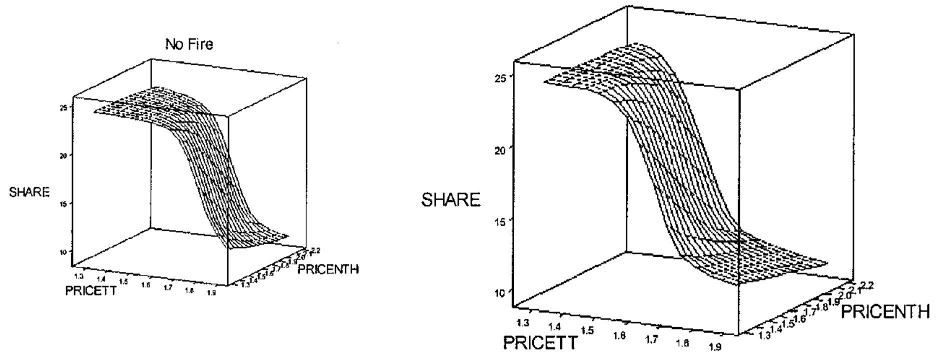
$$\hat{y} = \bar{y} + s_y(0.12 + 0.51h_1 - 0.17h_2)$$

The negative coefficient for h_2 is indicative of the negative impact of both the prices and the problem on market share, while the positive coefficient for h_1 suggests that h_1 is acting as a filter for the non-problem period, when Northern Delight's price was relatively high in comparison with True Treat's price.

This model can be interpreted using two plots. In Figure 4a the PROBLEM variable is set at zero, indicating no problem, and in Figure 4b the PROBLEM variable is set at one, indicating a week in which the problem was affecting sales. The surface with no problem tends to be higher than the problem surface, indicating that the problem adversely affected market share as expected. This justifies the inclusion of the PROBLEM variable in the model. The other interesting feature of these graphs is the range of PRICETT values in which SHARE is very sensitive. Without the problem SHARE is sensitive to PRICETT for prices of more than about \$1.60. This means that there is no point in reducing prices below \$1.60 because it will have very little effect on market share. However, with the problem present the sensitive range reduces to \$1.50-\$1.70, again indicating the adverse effect of the problem, because demand falls faster in response to price increases in the presence of the problem.

Figure 4: Visualisation: Effect of prices on share for the youth cold drink market when PROBLEM was absent and present

PROBLEM absent **PROBLEM present**



(a) (b)

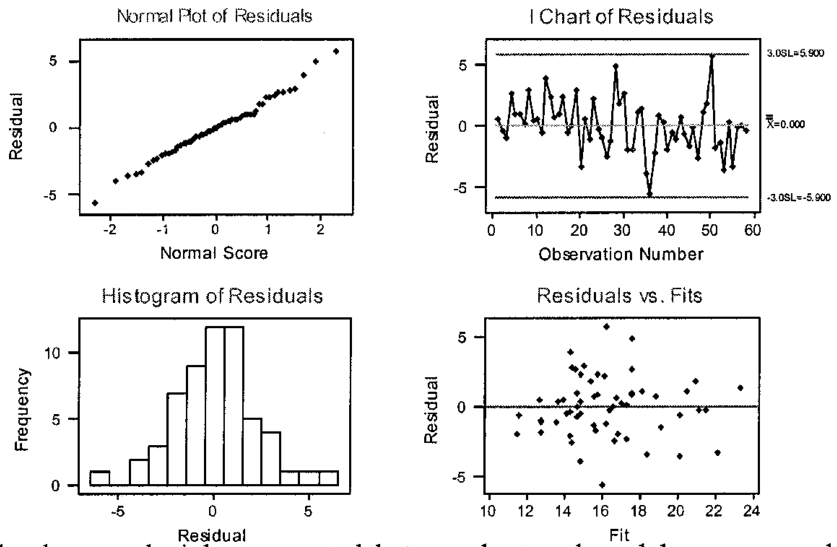
The above graphical analysis has served two purposes. Firstly it has validated the model in that it has confirmed that our model is sensible and, secondly, it has allowed us to interpret the model in a more useful manner than was possible using only the coefficients (weights). Increased prices are associated with reduced share, but only above a specific level defined by the presence or absence of the PROBLEM. In addition, the graphs confirm that the PROBLEM effect impacts negatively on share. As is to be expected the relationships between these variables are definitely nonlinear and the regions of price sensitivity are of great financial importance.

We must now turn back to Figure 2 and ask whether the neural network model is superior to the regression model. Figure 4 seems to suggest that the neural network model is more realistic but what of the behaviour of the residuals? Table 1 indicates that the residuals for the neural network model do not have the appealing characteristic of a zero mean, and the residual standard deviation is larger for the neural network model suggesting an inferior fit. For the sake of consistency the same standard formula for sample standard deviation has been used despite the differing number of coefficients for the two models. However, the plots in Figure 5 suggest that the residual behaviour for both the neural network and regression model is reasonably good (i.e., random and approximately normal in distribution). Indeed there seems to be some similarity in these plots.

Table 1: Descriptive statistics for residuals

Model	Residual Mean	Residual Standard Deviation
Regression	0.000	2.171
Neural Network	-0.230	2.472

Figure 5(a): Residual plots for regression model



The above analysis has suggested that neural network models are more realistic than linear regression models, provided that they are not over-parameterised. Residual and predicted value plots (Figures 5 and 4) have been used successfully for diagnostic checking and interpretation of the neural network models; however, these are rather simplistic tools and with more input variables the predicted value plots would have been less useful.

Market Share Model for Burgen Broth

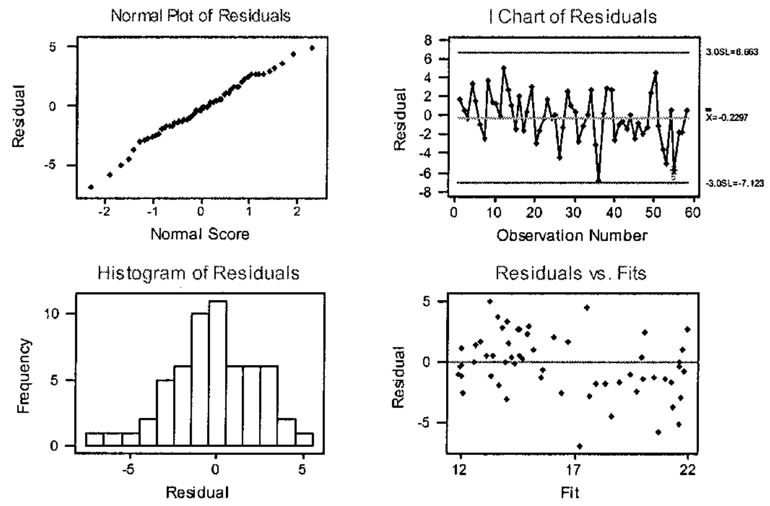
We now consider the third (health) cold drink, that is Burgen Broth (BGN). As noted previously, the health cold drink market is not price sensitive and advertising is expected to have a lagged effect on market share. Market knowledge and stepwise regression (Hair et al., 1998) were used to select only four of the available input variables and, on account of the small sample size, only two hidden nodes were allowed. The market share for next week (y) was predicted using this week's share (x_1), the effect of Christmas (x_2), the PROBLEM effect (x_3) and advertising spend (x_4) in the following neural network model.

$$h_1 = \tanh\{-3.16 + 1.45\left(\frac{x_1 - \bar{x}_1}{s_1}\right) + 0.10\left(\frac{x_2 - \bar{x}_2}{s_2}\right) + 0.17\left(\frac{x_3 - \bar{x}_3}{s_3}\right) + 0.65\left(\frac{x_4 - \bar{x}_4}{s_4}\right)\}$$

$$h_2 = \tanh\{-0.68 + 0.83\left(\frac{x_1 - \bar{x}_1}{s_1}\right) + 1.15\left(\frac{x_2 - \bar{x}_2}{s_2}\right) + 1.74\left(\frac{x_3 - \bar{x}_3}{s_3}\right) - 2.45\left(\frac{x_4 - \bar{x}_4}{s_4}\right)\}$$

The first hidden node filters data for which this week's share (x_1) and advertising spend (x_4) have the biggest impact on next week's share. The second hidden node filters the Christmas and/or PROBLEM periods when advertising spend has less effect on market share. These equations suggest that there are short-term trends in share with share rising in response to advertising expenditure, except

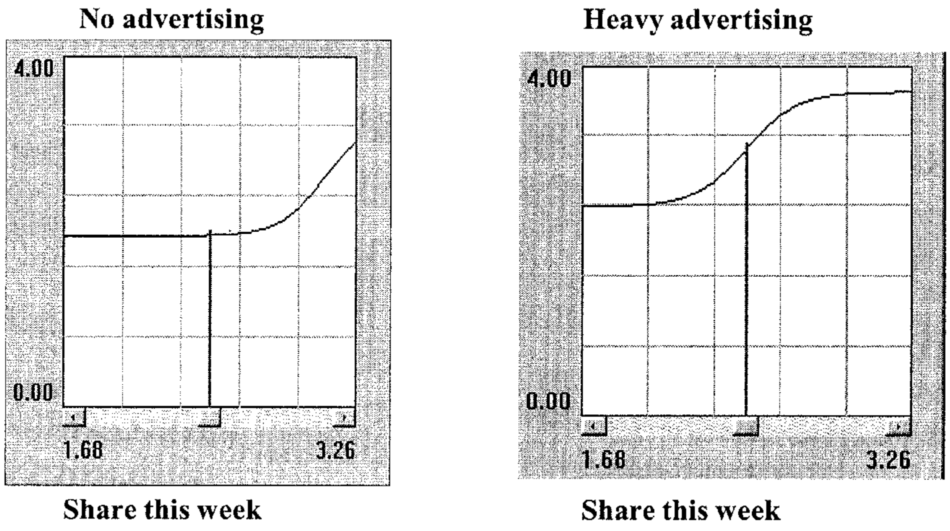
Figure 5(b): Residual plots for neural network model



at Christmas and during the PROBLEM period.

Figure 6 shows the effect of last week's share on next week's share when there is no advertising and when there is heavy advertising ($1,640 per week). These graphs suggest that increasing advertising from 0 to $1,640 will increase market share by 0.5% when this week's share is low (e.g., 1.6%) and by about 1% when this week's share is high (e.g., 3.26%). In addition it is clear from the right-hand graph that advertising is particularly beneficial when this week's share is about 2-3% of the total market.

Figure 6: The affect of advertising on BGN share in consecutive weeks



Finally Figure 7 shows the influence of first the PROBLEM and then CHRISTMAS on the effect of ADVERTISING on next week's share (SHARE). It confirms that advertising had less influence on next week's share at the time of the problem (PROBLEM = 1) and over Christmas (CHRISTMAS = 1).

Again it seems that the neural network model has produced a sensible result when care has been taken to avoid over-parameterisation. In the equivalent linear regression model, PROBLEM and CHRISTMAS are insignificant and the model is again "inferentially uninteresting,"

$$\text{SHARE}(t+1) = 0.81\ \text{SHARE}\ (t) + 0.46$$

Comparing the residuals for the neural network and the above time series model, we obtain the results shown in Table 2 and Figure 8.

The neural network model has produced some outliers and its residual plots are definitely not random, showing an upward trend over time. This indicates that the neural network model has failed to track the changes in share over time and is therefore inappropriate. The I Chart suggests that we need to identify and incorporate in our model, the reason(s) for the rise in market share in the last few weeks. Having obtained rather mixed prediction performance for neural networks in the case of our very small sample, let us now consider a classification example.

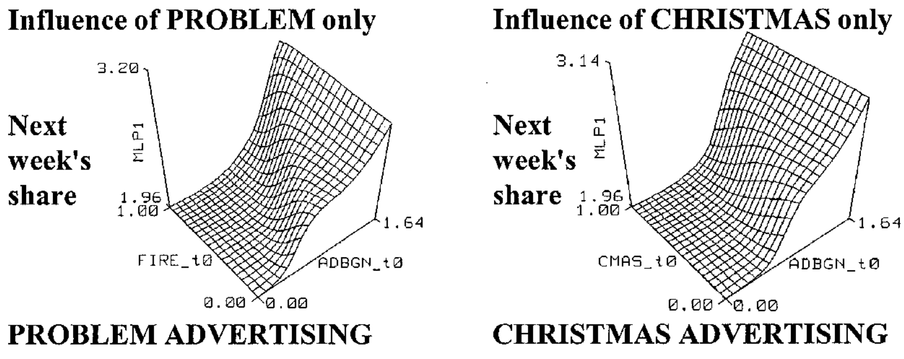
Classification of Customers According to Their Market Segments

We now consider the application of neural networks in terms of a typical market research classification example. This analysis is more data rich in that we

Table 2: Residual analysis

Model	Residual Mean	Residual Standard Deviation
Time Series	0.0044	0.2386
Neural Network	0.0829	0.3173

Figure 7: Effect of ADVERTISING, PROBLEM and CHRISTMAS on SHARE



have 2,010 observations to analyse, with each observation corresponding to a respondent in a survey performed for the youth cold drink market. However, in data mining terms this is still a small data set. The data consisted of more than 800 variables including a segment variable with six categories—namely, Stable Types, Party Types, Sporty Types, Trendy Types, Independent Types and Discerning Types. The variables tended to be binary (Yes/No), delving into areas such as "Ideal Drink Characteristics," "Ideal Person Characteristics" and "Ideal Activities and Pastimes." A parsimonious rule for tracking this segmentation was required. This meant that the best variables had to be selected and used to derive a discriminant rule. Two of the non-binary variables (cold drink consumption per week and age) were log-transformed prior to analysis on account of the skewed right-hand tails of these

Figure 8(a): Residual analysis for lagged regression model

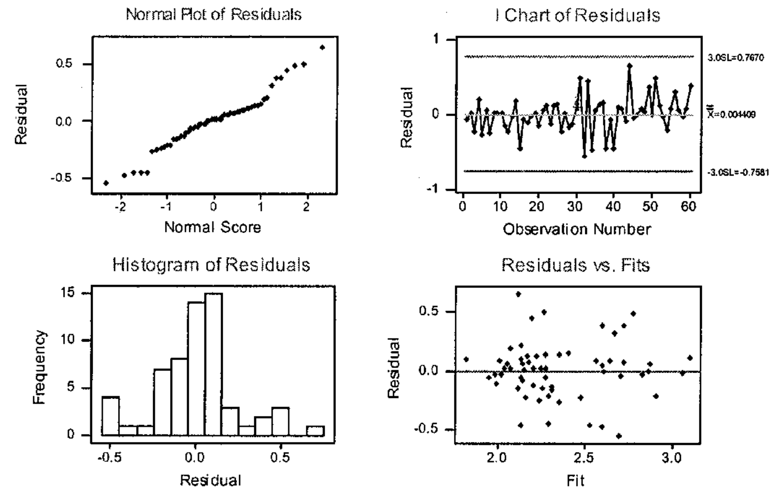
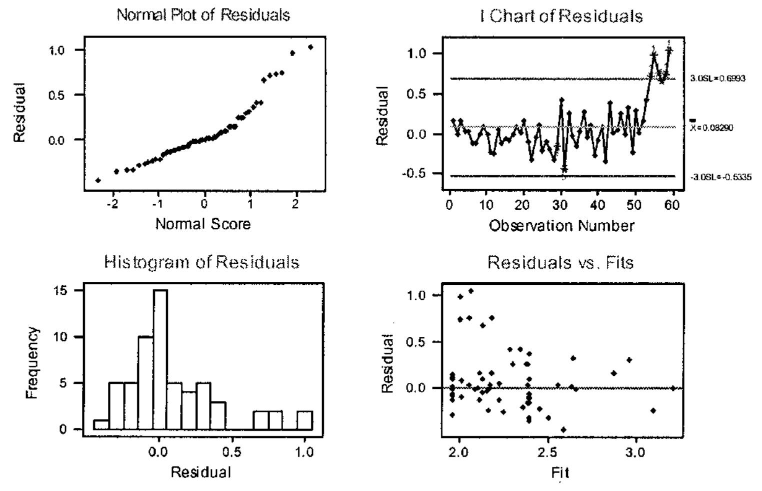


Figure 8(b): Residual analysis for neural network model



distributions. Conventional stepwise discriminant analysis (Hair et al., 1998) was used to select the 43 most important discriminatory variables and the data were analysed using a conventional linear discriminant analysis. The discriminant rule was developed using 90% of the data and was tested on the remaining 10% of the data, producing a 27.6% predictive error rate. The binary nature of much of the data suggests that this linear method should perform particularly well.

The neural network required to classify these data had 43 input nodes (corresponding to the above 43 most important discriminatory variables) and 6 outputs, one output for each of the segments. With only 1,608 observations in the training data set, this meant that we needed at most six hidden nodes in order to keep the number of observations per weight above five. In the following analysis we varied the number of hidden nodes with two different sets of initial weights in order to determine whether the number of hidden nodes could be reduced below six. A 80%:10%:10% training: validation:test split was applied to the data, allowing a proper validation and test of the neural network models. Figure 9 reports the predicted error rate obtained from the test data, suggesting that four hidden nodes is sufficient. This relatively low number of hidden nodes in relation to the number of segments (i.e., six) is indicative of a fairly linear system which is to be expected on account of the binary nature of most of the variables, as mentioned previously. The misclassification rate was clearly bigger than the 27.6% achieved with conventional discriminant analysis, and this rate was obviously affected by the initial weights because the predictive error rate varied even when the same number of hidden nodes was used.

The interpretation of the neural discriminant rule is difficult when there are so many variables involved because a plot of predicted values for all input variables becomes impossible. We therefore used the relative size and sign of the weights in order to interpret the model, as suggested in Table 3.

The Solid segment is identified largely in terms of the second hidden node (h_2). The Social and Sporty segments are identified by h_1, h_3 and h_4. The Trendy segment is identified mostly by h_2, while the Independent segment is identified mostly by h_3

Figure 9: Optimising the number of hidden nodes for a neural network

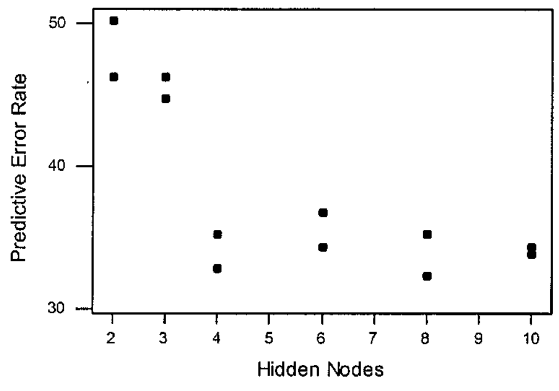


Table 3: Interpretation of neural network discriminant rules

Hidden Nodes(sign)	h1	h2	h3	h4
Absent Interests(-)	Rugby	Parties		
Absent Characteristics(-)	Team player	Let's Go	Energetic	Stylish
Absent Characteristics(-)			Stylish	
Present Interests(+)	Surf/Skate		Clubbing	
Present Characteristics(+)	Different	Stylish	Boys	Older Girls
Present Characteristics(+)	Discerning	Older Boys		

Segment	Hidden Node Weights(w)			
Solid Types(O_1)	-0.08	**0.26**	0.15	-0.12
Social Types (O_2)	**-0.14**	-0.02	**-0.12**	**0.13**
Sporty Types (O_3)	**-0.21**	-0.11	**-0.24**	**-0.21**
Trendy Types (O_4)	0.14	**0.18**	-0.13	0.11
Independent Types (O_5)	-0.06	-0.15	**0.21**	0.17
Discerning Types (O_6)	**0.35**	-0.17	0.13	-0.09

and the Discerning segment is identified mostly by the first hidden node. As expected Solid Types tend to be older and more reserved (+ h_2). The negative coefficients for h_1 and h_3 for Social Types means that Social Types are team players, stylish and energetic. So are Sporty Types, but more so. Trendy Types like to be stylish (+ h_2) and Independent Types like clubbing (+ h_3). Discerning Types are just that (+ h_1).

In this example stepwise discriminant analysis has been used to prune the number of input variables before attempting to fit a neural network model. Because of the large number of input variables the model has been interpreted using weights rather than a graphical analysis. It has been found that the neural network model has a slightly lower predictive accuracy than the classical linear discriminant analysis model. Nevertheless it seems that neural networks are a viable classification tool for this small data set.

CONCLUSIONS

In this chapter we have considered the use of neural networks for prediction and classification purposes when sample sizes are relatively small. It has been found that typical commercial neural network software can be used to fit neural network models in this situation, provided that the number of hidden nodes is minimised and only important input variables are selected. In our case, linear stepwise selection procedures worked reasonably well, but for more complex systems, with more non-linearity and interaction, more advanced selection methods are required. For this reason the recent theoretical developments in regard to network growing and pruning (Haykin, 1999; Bishop, 1995; LeCun, Denker & Solla, 1990; Hertz, Krogh

& Palmer, 1991) are most important for small data sets.

Neural networks tend to include more input variables than conventional statistical models and, with small data sets, it is especially important to understand and trust the logic of these more complicated models. Graphical procedures for illustrating neural network models that allow for easy (push-button) setting of levels for input variables are needed if neural networks are to be effective query and simulation tools. In addition, procedures that perform automatic rule extraction for each hidden node are needed. Finally, small data sets are less robust to data error than large data sets. For this reason it is vital that the diagnostic theory for neural networks be developed and implemented as an integral part of commercial neural network packages. These diagnostics need to be presented in a way that highlights model deficiencies.

It seems that with large data sets the above requirements are not essential because neural networks can still produce useful information without them. This is not to say that the application of neural networks with huge data sets will not benefit from these developments. Indeed the reverse will be true because all neural networks will gain more credence, more efficiency and more reliability if the irrelevant input variables can be detected and omitted, the hidden node outputs can be better explained and diagnostic checking can be used to detect data errors and suggest model improvements.

In our analyses the neural network models failed to match the predictive accuracy of conventional linear regression and linear discriminant analysis, but this may be due to high levels of linearity in the systems studied. It is recommended that neural networks and conventional statistical tools should be compared for many more small and large data sets. It is only in this way that statisticians will learn when and where to trust neural networks.

REFERENCES

Andrews, R., Diederich, J. & Tickle, A. (1995). A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6), 373-389.

Bishop, C.M. (1995). *Neural networks for pattern recognition*. Oxford: Clarendon Press.

Brierley, P. & Batty, B. (1999). Data mining with neural networks – an applied example in understanding electricity consumption patterns. In M.A. Bramer (Ed.) *Knowledge Discovery and Data Mining*. (pp. 240-303). London: The Institution of Electrical Engineers.

Chatfield, C.(1995). Model uncertainty, data mining and statistical inference. *J.R. Statist. Soc. A*, *158*(3), 419-466.

Cheng, B. & Titterington, D.M. (1994). Neural networks: A review from a statistical perspective. *Statistical Science, 9*(1), 2-54.

Ciampi, A. & Lechevallier Y. (1997). Statistical Models as building blocks of neural networks. *Commun. Statist. Theory and Methods, 26*(4), 991-1009.

Cooper, J.C.B. (1999). Artificial neural networks versus multivariate statistics: an application from economics. *Journal of Applied Statistics, 26*(8), 909-921.

Duh, M., Walker, A.M., Pagano, M. & Kronlund, K. (1998). Prediction and cross-validation of neural networks versus logistic regression: Using hepatic disorders as an example. *American Journal of Epidemiology*. 147(4), 407-413.

Elder, J.F. & Pregibon, D. (1996). A statistical perspective on knowledge discovery in data bases. In Fayyad, U.M., Piatesky-Shapiro, G., Smyth, P. & Uthurusamy, R.(Ed.), *Advances in Knowledge Discovery and Data Mining*. 83-113, Cambridge, MA: AAAI Press/MIT Press.

Faraway, J. & Chatfield, C. (1998). Time series forecasting with neural networks: a comparative study using the airline data. *Applied Statistics*, 47(2), 231-250.

Hair, J.F., Anderson, R.E., Tatham, R.L. & Black W.C. (1998). *Multivariate Data Analysis*. Upper Saddle River, New Jersey : Prentice-Hall International.

Hall, M.A. & Holmes, G. (2000). *Benchmarking attribute selection techniques for data mining*. Working paper 00/10, Department of Computer Science, University of Waikato, Hamilton, New Zealand.

Haykin, S. (1999). *Neural Networks, A Comprehensive Foundation*, (2nd Ed.). New Jersey: Prentice Hall.

Hertz, J., Krogh, A. & Palmer, R.G. (1991). *Introduction to the theory of neural computation*. Sante Fe Institute.

LeCun, Y., Denker, J.J. & Solla, S.A. (1990) Optimal Brain Damage. In D. Touretsky (Ed.) *Advances in Neural Information Processing Systems*. Morgan Kaufman.

Mackinnon, M.J. & Glick, N. (1999). Data mining and knowledge discovery in databases - an overview. *Australian and New Zealand Journal of Statistics, 41*(3), 255-276.

Maindonald, J.H.(1998). New approaches to using scientific data statistics, data mining and related technologies in research and research training. RetrievedOctober 26, 1999 from http://www.anu.edu.au/academia/graduate/papers/gs98_2.html.

Maire, F. (1999). Rule extraction by backpropagation of polyhedra. *Neural Networks*, 12(4-5), 373-389.

McClelland, J.L., Rumelhart, D.E. & the PDP Research Group (1986). Parallel distributed processing: Exploration in the microstructure of cognition, Volume 2: *Psychological and Biological Models*, Cambridge, MA:MIT Press.

McCulloch, W.S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115-133.

Nelder, J.A.(1999). From statistics to statistical science. *The Statistician, 48*(2), 257-269.

Ripley, B.D. (1994). Neural Networks and Related Methods for Classification. *Journal of the Royal Statistical Association, 56*(3), 409-456.

Ripley, B.D. (1996). *Pattern Recognition and Neural Networks*. New York: Cambridge University Press.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6, 461-464.

Stone, M. (1978). Cross-validation: A review. *Mathematische Operationsforschung Statischen, Serie Statistics, 9*. 127-139.

Stone, M. (1974). Cross-validatory choice and assessment of statistical prediction. *Journal of the Royal Statistical Society, B36*, 111-133.

Warner , B. & Misra, M. (1996). Understanding neural networks as statistical tools. *The American Statistician, 50*(4), 284-293.

Chapter XI

How to Train Multilayer Perceptrons Efficiently With Large Data Sets

Hyeyoung Park
Brain Science Institute, RIKEN, Japan

Feed forward neural networks or multilayer perceptrons have been successfully applied to a number of difficult and diverse applications by using the gradient descent learning method known as the error backpropagation algorithm. However, it is known that the backpropagation method is extremely slow in many cases mainly due to plateaus. In data mining, the data set is usually large and the slow learning speed of neural networks is a critical defect. In this chapter, we present an efficient on-line learning method called adaptive natural gradient learning. It can solve the plateau problems, and can be successfully applied to the learning associated with large data sets. We compare the presented method with various popular learning algorithms with the aim of improving the learning speed and discuss briefly the merits and defects of each method so that one can get some guidance as to the choice of the proper method for a given application. In addition, we also give a number of technical tips, which can be easily implemented with low computational cost and can sometimes make a remarkable improvement in the learning speed.

INTRODUCTION

Neural networks are one of the most important methodologies in the field of data mining and knowledge discovery, in which one needs to extract abstract knowledge such as rules or consistent properties from large data sets. With a large data set, one sometimes wants to find a functional mapping from input to output in order to give a prediction for a new input. On the other hand, one sometimes wants to cluster enormous amounts of data into several groups in order to get some information about the distributions of the data sets. Neural networks can manage such tasks well through the "learning process." Even though there are various neural network models, the multilayer perceptron (MLP) is one of the most popular models and so has been widely used for a variety of applications. In addition, it is also theoretically proven that MLP with one hidden layer can approximate any continuous function to any desired accuracy (Cybenko, 1989). In practical situations, however, those who are trying to use MLP as an application tool are often disappointed by the low learning performance such as poor generalization and slow convergence. In most cases, the low learning performance is caused by the use of a network with a bad structure or by the use of an inappropriate learning algorithm.

Following the development of the backpropagation learning algorithm (Rumelhart & McClelland, 1986), the MLP has been successfully applied to various fields of application such as pattern recognition, system control, time series prediction as well as data mining (Haykin, 1999; Michalski, Bratko, & Kubat, 1998). However, its slow learning speed has been a serious obstacle when used for real-world applications. Many studies have tried to solve this problem. Some of the studies are based on simple heuristics, while others are theoretical in nature. Even though there is little theoretical justification in the heuristic solutions, they are simple to apply and perform well in many cases. On the other hand, the theoretically inspired methods can give rigorous proof about their theoretical learning efficiency. However, they are sometimes too complex to apply to practical applications. Thus, it is important to select a method that is appropriate for an application in order to succeed.

In this chapter, we focus on the problem of slow convergence and propose a solution especially for learning using large data sets. We present an efficient learning method called adaptive natural gradient learning, which has been recently developed by Park, Amari, and Fukumizu (2000). We also give a number of technical tips for training MLP, which can be easily combined with general learning algorithms and the adaptive natural gradient learning method. In addition, we compare the adaptive natural gradient method with various popular learning algorithms for improving the learning speed and give short comments on the merits and defects of each method so that one can get some guidance towards choosing a proper method for a given problem.

In the Background section, we start our discussion by describing the structure of MLP and its learning process. In the section on How to Train Networks, we present simple tips for when designing MLP and explain various methods for accelerating the learning speed including the adaptive natural gradient learning method. In the Future Trends section, we mention current trends and some interesting issues relating to the learning of neural networks. Some brief discussion and conclusions are stated in the Conclusions section.

BACKGROUND

Network Model

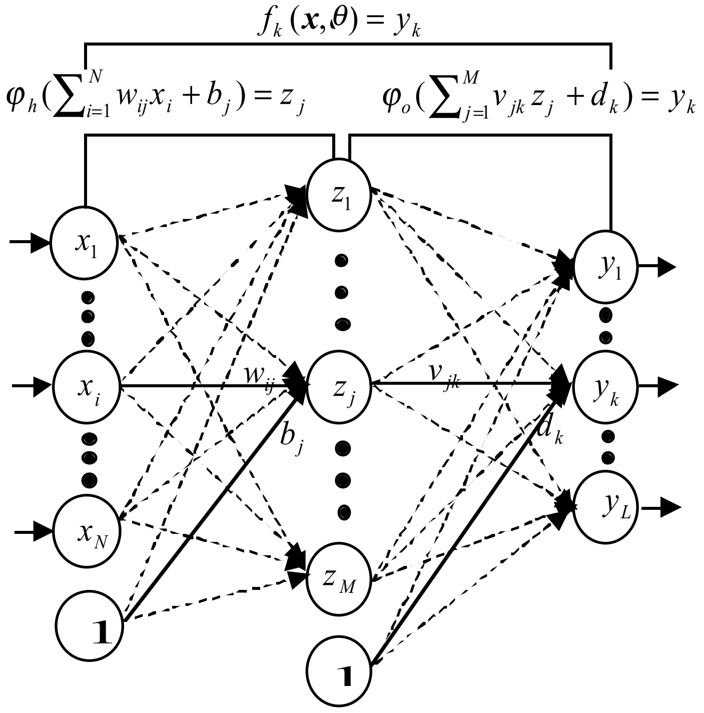
The overall architecture of MLP is shown in Figure 1, which can be defined by

$$y_k = f(\boldsymbol{x}, \theta) = \varphi_o\left(\sum_{j=1}^{M} v_{jk} z_j + d_k\right), \qquad k = 1,2,\ldots,L, \tag{1}$$

$$z_j = \varphi_h\left(\sum_{i=1}^{N} w_{ij} x_i + b_j\right), \qquad j = 1,2,\ldots,M, \tag{2}$$

where y_k is the output value of k-th output node, x_i is the input value of i-th input node, w_{ij} is the weight parameter from the i-th input node to the j-th hidden node, v_{jk} is the

Figure 1: A multilayer perceptron


$f_k(\boldsymbol{x}, \theta) = y_k$
$\varphi_h(\sum_{i=1}^{N} w_{ij} x_i + b_j) = z_j$
$\varphi_o(\sum_{j=1}^{M} v_{jk} z_j + d_k) = y_k$
x_1
x_i
x_N
1
z_1
z_j
z_M
1
y_1
y_k
y_L
w_{ij}
b_j
v_{jk}
d_k

weight parameter from the j-th hidden node to the k-th output node, b_j is the bias to the j-th hidden node, and d_k is the bias to the k-th output node. All of these parameters are represented by the vector θ, for simplicity. The functions φ_o and φ_h are the activation functions for hidden nodes and output nodes, respectively. The network has N input nodes, M hidden nodes in one hidden layer, and L output nodes. We are focusing on the model with only one hidden layer for simplicity, even though most methods presented in this chapter can be used for more general models. Theoretically, MLP with one hidden layer has been proven to have universal approximation ability (Cybenko, 1989).

Now, let us talk about the learning of the network. The multilayer perceptrons are trained by a supervised learning scheme, which means that the desired output $\boldsymbol{y}^*$ for a given input $\boldsymbol{x}$ is presented as learning data of the form $(\boldsymbol{x},\boldsymbol{y}^*)$. More formally, given a learning data set $\boldsymbol{D} = \{\boldsymbol{x}^p,(\boldsymbol{y}^*)^p\}_{p=1,\ldots,P}$ with P observations, the goal of learning is to find an optimal function mapping $f(\boldsymbol{x},\theta^*)$, which predicts desired outputs for given inputs. To this end, we first need to define an error function $E(\theta)$ that measures the goodness of the outputs of the network function $f(\boldsymbol{x},\theta)$ specified by the current parameter θ. Then the learning is a process of searching for an optimal parameter that minimizes the value of the error function $E(\theta)$, starting from an initial parameter θ_1 in the parameter space.

The search process proceeds step by step, according to a learning algorithm. At each step, we need to decide the direction and the amount of movement for parameter θ. The equation of one-step movement of the parameter at time t can be written by

$$\theta_{t+1}=\theta_t+\Delta\theta_t \tag{3}$$

$$\Delta\theta_t=\eta_t\boldsymbol{d}_t \tag{4}$$

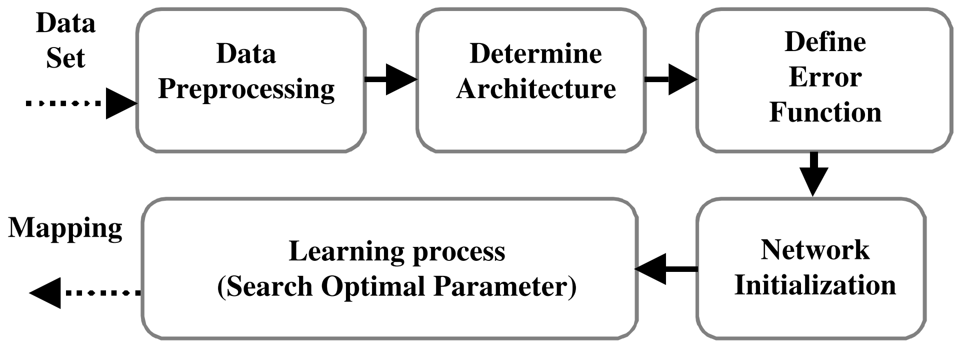
where $\boldsymbol{d}_t$ is the search direction, and the learning rate η_t is the amount of movement at that step. In gradient descent learning, the search direction $\boldsymbol{d}_t$ is determined according to the negative derivative of the error function —$E(\theta)$. Thus, the standard rule for updating parameters in the gradient descent method is expressed by the form,

$$\theta_{t+1}=\theta_t-\eta_t\nabla E(\theta_t). \tag{5}$$

From Equation 5, we can sense the problem of the gradient descent learning method. Since the search direction is mainly dependent on the gradient of the error function, the size of update is very small in the region where the error surface is flat. Such a flat region in the error surface is called a plateau, and so the problem of slow learning convergence in the gradient descent method is called the plateau problem.

The overall process for designing a neural network system is illustrated in Figure 2. First, we are given a large data set, in which each piece of data consists of an input vector and an output vector. Then, we need to transform the data set into a suitable form for learning by a neural network, the process of which is usually called the preprocessing stage. With the preprocessed data set, the next stage is to

Figure 2: Overall process of designing a neural network system



determine the detailed structure of the network to be trained. For example, we need to determine the number of hidden layers and hidden nodes, the form of the activation function of the nodes, and so on. We then define the goal of learning, namely, the error function, under the determined architecture. Finally, after taking a starting point of the learning through an initialization process of parameters, the network starts the search process, using a learning algorithm. After this whole process is finished successfully, we expect the network to produce an appropriate function mapping.

HOW TO TRAIN NETWORKS

In this chapter, we are concentrating on the learning algorithm based on the gradient descent method. There are many variations of the gradient descent method depending on how the derivative of the error function is used and how the learning rate is determined, and thus we can expect to improve the learning speed by selecting a method in a suitable way. In this section, we will briefly describe some of the popular learning methods, discuss their characteristics, and finally give an efficient learning method for the problem of large data sets. In addition, we should note that there are some other important factors to be considered when improving the learning performance. In some cases, handling of these factors in an appropriate manner can give remarkable improvement with small computational cost. Thus, before discussing learning algorithms, we present a number of useful technical tips for preparing for the actual learning process.

Preparation for Learning

Data Preprocessing

We can expect remarkable improvement of the learning speed just by transforming given raw data into appropriate values for learning. If the range of input values of the raw data set is too wide, then the search space for the weight parameter becomes large and it will take a significant amount of time to find an optimal parameter. On the other hand, if the range is too narrow, it can make the adjustment of parameter values difficult. Moreover, if the range of values corresponding to each input node is much different from the others, then the influence of each input node on each hidden node differs, which also can make the learning process difficult. Therefore, it is necessary to transform the values of input nodes so that they exist in the same range. The most basic and popular method is to standardize the input data. The following algorithm can achieve this.

1. For a given data set $\boldsymbol{D} = \{\boldsymbol{x}^p, (\boldsymbol{y}^*)^p\}_{p=1,\ldots,P}$ with P observations, calculate the sample mean μ_i and the standard deviation σ_i for each element x_i (i=1,...,N) of the input vector by using

$$\mu_i = \frac{1}{P}\sum_{p=1}^{P} x_i^p \tag{6}$$

$$\sigma_i = \frac{1}{P}\sum_{p=1}^{P} (x_i^p - \mu_i)^2 \tag{7}$$

2. For each element of each data input ($x^p(p=1,\ldots,P)$, transform it to generate a new value by

$$\tilde{x}_i^p = \frac{(x_i^p - \mu_i)}{\sigma_i} \tag{8}$$

Then we use the transformed data $\tilde{x}^p$ as the input data for learning instead of the raw data $\boldsymbol{x}^p$. However, we should note here that we apply the same transformation process to the new data as used for testing the network trained by the transformed data. For this purpose, the mean and the standard deviation need to be stored for each variable. The effect of this preprocessing is shown in the section on Computational Experiments.

For the output nodes, on the other hand, there are two cases to be considered separately. In the case of continuous output values, no special transformation is necessary except when the desired output values are too large or their variance is too small. In these cases, the standardized transformation technique used for the input values above may give a better representation for the output values. However, in the application of data mining and pattern recognition, the final outputs are often nominal, indicating the class that the input data belongs to. In this case, 1-of-L code scheme is used for making a proper output representation. If the number of classes is L, then the number of output nodes should be L. The value of the k-th output node is one only when the input data belongs to the k-th class and is zero otherwise. Besides the above two cases, we should note that the value of the output data is sometimes closely related to the activation function of the output nodes, as we shall explain in the next subsection.

Definition of Network Architecture

Though there are many factors to be considered when designing a network model, we consider only two factors in this section: the form of the activation function and the number of hidden nodes. For the activation function, the most general one is a sigmoidal function, such as the logistic sigmoid and the hyperbolic tangent function, but there are other special forms of activation function. The most common activation functions and their characteristics are listed in Table 1.

Even though every sigmoidal function can be used for hidden nodes, LeCun, Bottou, Orr, and Müller (1998) claim that the hyperbolic tangent is better than the logistic sigmoid because it has a similar effect of standardizing input values to the output nodes. With more specialized activation functions such as the circular function (Ridella, Rovetta, & Zunino, 1997) or the second order function (Hassoun, 1995), we can expect better performance when the decision boundaries among classes are complex. For the activation function of the output nodes, we need to consider the type of output value and the type of error function, as we will discuss later. In the case of continuous output values, the linear or identity function is desirable. In the case of nominal outputs with 1-of-L coding, the sigmoidal function with bounded range is recommended. In addition, if one uses the cross entropy error function, it is required to use the softmax activation function for output nodes so as to force the sum of values for all output nodes to be one. The formal definition of the softmax function is shown in Table 1.

Another important factor in defining the network architecture is the number of hidden nodes. Unfortunately, there is no clear guide for determining this because it is strongly dependent on the application and the learning data set. However, there has been a significant amount of research carried out to determine the optimal

Table 1: Common activation functions

Name	Definition $\varphi(u)$	Derivative $\varphi'(u)$	Properties of $\varphi(u)$
Identity (Linear)	u	1	Appropriate for output nodes of regression problems with continuous outputs
Logistic Sigmoid	$\frac{1}{1 - e^{-u}}$	$\varphi(u)(1-\varphi(u))$	Bounded to (0,1)
Hyperbolic Tangent	$\frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$	$(1+\varphi(u))(1-\varphi(u))$	Bounded to (-1,1)
Softmax	$\frac{e^{u}}{\sum_{k-1}^{L} e^{uk}}$	$\varphi(u)(1-\varphi(u))$	Appropriate for output nodes of classification problem with the cross entropy error function

number of hidden nodes. Methods such as pruning and growing can be well applied when we need an optimal and compact architecture (Haykin, 1999), but they are computationally costly and the implementation is not simple. In this section, we give a theoretical result as a hint for determining the number of hidden nodes. Hassoun (1995) showed that the lower bound for the necessary number of hidden nodes in a one-hidden-layer MLP with the linear threshold activation function for classifying two patterns of arbitrary shapes is $P/(N\log_2(Pe/N))$, where P is the number of observations, e is the natural base of logarithms, and N is the number of input nodes. From this result, we can get some hints as to the necessary number of hidden nodes. If we have a large number of learning data sets and a small dimension of the input, we may need many hidden nodes. On the contrary, with the same number of data sets and a large input dimension, we may need a much smaller number of hidden nodes, relatively speaking.

Definition of Error Function

It is also very important to choose an appropriate error function for a given application. On the basis of the output values, most applications can be divided into two types of problems: the regression problem and the classification problem. While the regression problem such as function approximation, time series prediction or non-linear system identification generally has continuous output values, the classification problem has nominal output values in general. From the statistical point of view, these two types of problems should be described by strictly different stochastic models, and thus we get different forms of error function from the stochastic approach (Bishop, 1995). Since the details about the stochastic models are beyond the purpose of this chapter, we only give the definition of error functions appropriate for each problem so that they can be used for the practical implementation of learning.

For the regression problem with continuous output values, the sum of the squared error (SE) function can be used successfully. The sum of the SE function is defined by

$$E_{SE}(\theta, D) = \frac{1}{P}\frac{1}{L}\sum_{p=1}^{P}\sum_{k=1}^{L}((y_k^*)^p - y_k)^2 \tag{9}$$

If we use the error function of Equation 9, then the whole data set needs to be shown to the network for calculating the corresponding error value and the one-step updating of parameters. This type of learning is called batch learning. However, we can take on-line learning by updating the learning parameters whenever each point of data is shown. For this case, the on-line version of the SE function is defined by

$$E_{SE}(\theta, (\boldsymbol{x}^p, (\boldsymbol{y}^*)^p)) = \frac{1}{L}\sum_{k=1}^{L}((y_k^*)^p - y_k)^2 \tag{10}$$

On the other hand, for the classification problem with binary output (zero or one), we need to consider a different type of stochastic model from that for the regression problem, and the cross entropy error function is obtained by using an appropriate stochastic model (See Bishop, 1995, for details). In the section on Computational Experiments, we show that the cross entropy error function gives a better performance than the SE function. The cross entropy function for L classes is defined by

$$E_{CE}(\theta, D) = \frac{1}{P}\sum_{p=1}^{P}\sum_{k=1}^{L}(y_k{}^*)^p \ln y_k \tag{11}$$

Its on-line version has the form of

$$E_{CE}(\theta, (\boldsymbol{x}^p, (\boldsymbol{y}^*)^p)) = \sum_{k=1}^{L}(y_k{}^*)^p \ln y_k \tag{12}$$

Note that when one uses the cross entropy function, the softmax activation function should be used for output nodes in order to make the sum of values of all output nodes to be one (Bishop, 1995).

Network Initialization

The starting point of learning also has some influence on the learning speed. Unfortunately, however, it is difficult to find a good starting point for general problems. The most common and safe but not so efficient method is just to initialize all parameters as small random values extracted either from a normal distribution with zero mean and small standard deviation, say, 0.1, or from the uniform distribution with small interval, say, [-0.2, 0.2]. The reason why we prefer small initial values is to avoid the node-saturation phenomena, which means that the input to the activation function is so large and its derivative is almost zero. When a node is saturated, the weights related to the nodes can hardly learn.

Learning Algorithms

In this section, we first give a number of widely used learning methods and discuss their theoretical and practical characteristics. Especially for the learning with large data sets, we present explicit algorithms of the adaptive natural gradient learning method. Before going into the discussion on the explicit learning algorithms, let us briefly mention two different learning modes: the batch learning and the on-line learning. As shown in the definition of error functions of Equations 9 and 11 for batch mode and Equations 10 and 12 for on-line mode, the batch mode uses the whole data set for one update, whereas the on-line mode uses just one piece of data for each update. Since the goal of learning is to minimize the error for the whole data set, the batch learning is more natural and stable. However, the batch mode approach is often trapped in plateaus due to the stability. By taking the on-line mode, we can bring stochastic randomness into the learning process to escape from plateaus or some local minima (Saad & Solla, 1995; Fukumizu & Amari, 2000). In addition, for the learning with large data sets, the batch mode takes a great deal of time for even a one-step update, and the batch mode could make the whole learning very slow. Thus, practically, the on-line learning is more efficient in most cases, even though there are some learning algorithms that can be used only for the batch mode.

Heuristic Methods

Many heuristic methods have been developed from experimental or empirical points of view to hasten the learning. Here, we explain three popular methods: the momentum method, the adaptive learning rate method, and the local adaptive learning rate method.

Let us start with the momentum method which is one of the simpler methods to speed up the gradient descent learning. The learning rule is the same as the standard gradient descent learning rule of Equations 3, 4, and 5, except that there is an extra term of momentum that provides the information about the latest update of parameters. The updated rule can be written as

$$\theta_{t+1} = \theta_t - \eta_t \frac{\partial E(\theta_t)}{\partial \theta_t} + \alpha_t(\theta_t - \theta_{t-1}) \qquad (13)$$

where $(\theta - \theta_{t-1})$ is called a momentum and α_t is a momentum rate usually chosen as a small constant between 0 and 1. By adding the momentum term, we can expect to accelerate the learning speed in the average downhill direction instead of fluctuating with every change of sign of the associated partial derivative of the error function.

Another simple heuristic method is to use the adaptive learning rate. Even though there are many different approaches to the adaptive learning rate, the basic rule originates from the same concept as the momentum method. Let us describe a very straightforward procedure where a learning rate is adapted according to the change of error. The rule to adjust the learning rate can be explained using three sub-rules.

1. If the error increases by more than a threshold value ξ, after a weight update, then the weight update is discarded and the learning rate is decreased.
2. If the error decreases after a weight update, the update is accepted and the learning rate is increased.
3. If the error increases by less than ξ, then the weight update is accepted but the learning rate is unchanged.

Even though the use of the adaptive learning rate method and the momentum method can, to some extent, improve the learning speed, they cannot avoid plateaus because they basically use the same direction as that of the standard gradient descent learning algorithm (Orr, 1995).

There is a slightly more sophisticated heuristic method called the local learning rate adaptation, which uses a separate learning rate η_i for each parameter component θ_i. Although there exist a lot of algorithms that use the local learning rate, such as the delta-bar-delta rule (Jacobs, 1988), RPROP (Riedmiller & Braun, 1992), and the Quick propagation (Fahlman, 1990), the basic rule to update each learning rate is the same as that of the adaptive learning rate method mentioned above. Unlike the adaptive learning rate method, the search directions given by this method can differ significantly from that of the standard gradient descent learning method. However, this method is still heuristic and there is no theoretical justification for using it.

Second Order Methods

The second order methods based on optimization theory use the Hessian matrix $H(\theta)$, which is obtained from the second derivative of an error function (Equations 9 and 11) with respect to parameter θ, in order to use the curvature information of the error surface during the learning. The basic and theoretic method is the Newton method. It starts from the local quadratic approximation to obtain an expression for the location of the minimum of the error function. When the error surface around an optimal point is quadratic, the optimal point θ^* can be obtained by just one update of the form,

$$\theta^* = \theta - (H(\theta^*))^{-1} \nabla E(\theta), \tag{14}$$

where $H(\theta^*)$ is the Hessian at the optimal point, and needs to be approximated through real implementation. In most practical problems, however, the error surface is hardly quadratic, so the Newton method is efficient only around the optimal point. Moreover, since the Hessian matrix is not always positive definite, it cannot guarantee the convergence of the algorithm. From a practical point of view, the calculation of the Hessian matrix and its inverse is very time-consuming, as well.

As a solution for the problem of computational cost, the Quasi-Newton method iteratively computes an estimate of the inverse of the Hessian directly by using only the first-order derivative of the error function. The Broyden-Fletcher-Goldfarb-Shannon (BFGS) method is the most successful and commonly used formula for updating the estimate at each learning step t (see Bishop, 1995; LeCun et al., 1998 for details of the method). Even though the Quasi-Newton method can reduce the computational cost, it is still based on the quadratic approximation of the error surface. Thus the Quasi-Newton method can also only achieve good convergence around the optimal points where the error surface can be well approximated by the quadratic form. Moreover, it can only be applied to batch learning because it is based on the Hessian that gives the information of the surface of the error function for the whole learning data. Therefore, the Newton method and Quasi-Newton method are unsuitable for large data sets.

The more practically useful methods are the Gauss-Newton method and the Levenberg-Marquardt (L-M) algorithm designed for minimizing the sum of the SE function (Bishop, 1995). Even though there are theoretically more robust derivations of these methods, it is interesting to note the relationships with the basic Newton method. The Gauss-Newton method uses an approximated form of the pure Hessian of the sum of the SE function. The approximated Hessian is defined by

$$\tilde{\boldsymbol{H}}(\theta) = \sum_{p=1}^{P} \frac{\partial f(\boldsymbol{x}^p, \theta)}{\partial \theta}^T \frac{\partial f(\boldsymbol{x}^p, \theta)}{\partial \theta} \tag{15}$$

where the function specifies the mapping function of a neural network (refer Equations 1 and 2). Therefore, the Gauss-Newton method can be written as

$$\theta_{t+1} = \theta_t - \eta_t (\tilde{\boldsymbol{H}}(\theta_t))^{-1} \nabla E(\theta_t) \tag{16}$$

The L-M method has almost the same formula except that it adds a small term λI to the approximated Hessian in order to avoid too steep changes of the parameter values. Note that these methods are designed for the sum of the SE function and are appropriate for regression problems, and so they are not appropriate for the classification problems. Like other second order methods using the Hessian, they can only be applied for batch mode, which is not appropriate for the learning of large data sets.

On the other hand, the conjugate gradient method (Press, Teukolsky, Vetterling & Flannery, 1992) takes a slightly different approach but still uses second order information. The basic concept is that the search direction at each step should be given so as not to destroy the minimization at the previous step. To achieve this, we create two successive search directions $\boldsymbol{d}_t$ and $\boldsymbol{d}_{t+1}$ to satisfy the condition defined by

$$\boldsymbol{d}_{t+1}^T \boldsymbol{H}_{t+1} \boldsymbol{d}_t = 0 \tag{17}$$

If the two direction vectors satisfy the condition of Equation 17, then the two directions are said to be $\boldsymbol{H}$-conjugate to each other. Since the evaluation of $\boldsymbol{H}_{t+1}$ at each step is computationally costly, several algorithms for calculating $\boldsymbol{H}$-conjugate directions without explicit knowledge of Hessian have been developed. The Polark Riebiere method (Bishop, 1995), which is generally known to show good performances, is given by

$$d_{t+1} = -\nabla E(\theta_{t+1}) + \beta_t d_t, \tag{18}$$

where

$$\beta_t = \frac{\nabla \boldsymbol{E}(\theta_{t+1})^T (\nabla \boldsymbol{E}(\theta_{t+1}) - \nabla \boldsymbol{E}(\theta_t))}{\nabla \boldsymbol{E}(\theta_t)^T \nabla \boldsymbol{E}(\theta_t)} \tag{19}$$

Even though the conjugate gradient method has been applied to the training of MLP with great success, it can only be applied to batch learning and needs the line search method for the learning rate, requiring high computational cost. Consequently, the second order methods are only appropriate for problems with small networks and small data sets requiring high accuracy, in general. On the contrary, it is not suitable for the learning of large networks or data sets.

Natural Gradient Method

The natural gradient learning method, which is a kind of stochastic gradient descent learning method, originated from information geometry (Amari & Nagaoka, 2000). By considering the geometric structure of the space of learning systems such as neural networks, the information geometrical approach gives a more appropriate metric, the Fisher information metric, than the Euclidean metric. Theoretically, we can calculate a gradient of any differentiable error function defined on the space using this metric, and the gradient gives the steepest descent direction of the error surface at any point on the space. This gradient is called the “natural gradient,” and the natural gradient learning method can be obtained directly by using the natural gradient instead of the ordinary gradient obtained by using Euclidean metric.

Several researchers have shown the ideal performances of the natural gradient learning method. Amari (1998) showed that the natural gradient learning algorithm gives the Fisher-efficient estimator in the sense of asymptotic statistics. Park and Amari (1998) suggested that the natural gradient algorithm has the possibility of avoiding or reducing the effect of plateaus. This possibility has been theoretically confirmed by statistical-mechanical analysis (Rattray & Saad, 1999). Comparing with the second order methods such as the Gauss-Newton method and the L-M method, the natural gradient method can give a more general learning scheme in the sense that it can be exploited for various error functions. Moreover, it has a theoretically rigorous justification (Amari, Park & Fukumizu, 2000). Rattray and Saad (1998) also analyzed the dynamics of a number of second order on-line learning algorithms and the natural gradient learning algorithm, and showed the superiority of the natural gradient learning method in the transient dynamics of learning.

Let us start our explanation of the natural gradient learning algorithm through its basic form. The update rule for parameters is written as

$$\theta_{t+1} = \theta_t - \eta_t (G(\theta_t))^{-1} \nabla E(\theta_t). \tag{20}$$

One can see that the rule is very similar to that of the Gauss-Newton method except for the matrix part. The matrix $G(\theta_t)$ is the well-known Fisher information matrix, and its explicit form is determined by the stochastic model of the learning network. (Since theoretical definition of the Fisher information matrix and the theoretical derivation of the natural gradient learning method is beyond the scope of this chapter, please see Amari, 1998; Amari et al., 2000 and Park et al., 2000 for details.)

The problem of the pure natural gradient method is that it is almost impossible to obtain the explicit form of the Fisher information matrix and its computational cost is very high. Accordingly, it is required to find an estimate of $G(\theta_t)$, and the method of estimation could be different depending on the stochastic model assumed for learning system. Recently, Amari et al. (2000) developed an adaptive method of realizing the natural gradient learning for MLP, which is called the adaptive natural gradient descent method. Park et al. (2000) extended it to a general stochastic neural network model. In this chapter, based on the results of Park et al. (2000), we give the explicit algorithms of the adaptive natural gradient learning method for two error functions, the squared error function and the cross entropy error function.

The algorithm for the squared error function can be implemented by the following steps:

1. Choose an initial parameter vector θ_1 of small values randomly, and set the initial estimate of the Fisher information matrix $\hat{G}_0$ as the identity matrix.
2. For each learning data $(\mathbf{x}_t, \mathbf{y}_t^*)$ and the current learning parameter, calculate the output of the network, $y_k = f_k(\boldsymbol{x}_t, \theta_t)$, $(k=1,\ldots,L)$.
3. Calculate the current error $E(\theta_t)$ using the on-line definition of the squared error function written in Equation 10. If the error satisfies the terminating condition, then stop the learning process; otherwise proceed to the next step.
4. Update the estimation of the inverse of the Fisher information matrix using Equations 21 and 22,

$$\hat{\boldsymbol{G}}_t^{-1} = \frac{1}{1-\varepsilon_t}\hat{\boldsymbol{G}}_{t-1}^{-1} - \frac{\varepsilon_t}{1-\varepsilon_t}\hat{\boldsymbol{G}}_{t-1}^{-1}\nabla\boldsymbol{F}_t^T((1-\varepsilon_t)\boldsymbol{I} + \varepsilon_t\nabla\boldsymbol{F}_t^T\hat{\boldsymbol{G}}_{t-1}^{-1}\nabla\boldsymbol{F}_t)^{-1}\nabla\boldsymbol{F}_t^T\hat{\boldsymbol{G}}_{t-1}^{-1}, \quad (21)$$

$$\nabla\boldsymbol{F}_t = \left(\frac{\partial f_1(\boldsymbol{x}_t,\theta_t)}{\partial\theta_t}, \frac{\partial f_2(\boldsymbol{x}_t,\theta_t)}{\partial\theta_t}, \ldots, \frac{\partial f_L(\boldsymbol{x}_t,\theta_t)}{\partial\theta_t}\right) \quad (22)$$

where ε_t could be a small constant c or c/t.

5. Update the current parameter using the updating rule of Equation 23,

$$\theta_{t+1} = \theta_t - \eta_t\hat{\boldsymbol{G}}_t^{-1}\nabla E(\theta_t) \quad (23)$$

where the learning rate η_t(<<1) can be chosen as a small value.

6. Go to step 2.

For the cross entropy error function, the overall process is the same as that for the squared error function except for just two steps. In Step 3, the cross entropy error function of Equation 12 should be used instead of the squared error function. In Step 4, the estimation of the inverse of the Fisher information matrix should be calculated using Equations 24 and 25.

$$\hat{\boldsymbol{G}}_t^{-1} = \frac{1}{1-\varepsilon_t}\hat{\boldsymbol{G}}_{t-1}^{-1} - \frac{\varepsilon_t}{1-\varepsilon_t}\hat{\boldsymbol{G}}_{t-1}^{-1}\nabla\boldsymbol{F}_t^T((1-\varepsilon_t)\boldsymbol{I} + \varepsilon_t\nabla\boldsymbol{F}_t^T\hat{\boldsymbol{G}}_{t-1}^{-1}\nabla\boldsymbol{F}_t)^{-1}\nabla\boldsymbol{F}_t^T\hat{\boldsymbol{G}}_{t-1}^{-1}, \quad (24)$$

$$\nabla\boldsymbol{F}_t = \left(\frac{1}{\sqrt{f_1(\boldsymbol{x}_t,\theta_t)}}\frac{\partial f_1(\boldsymbol{x}_t,\theta_t)}{\partial\theta_t}, \ldots, \frac{1}{\sqrt{f_L(\boldsymbol{x}_t,\theta_t)}}\frac{\partial f_L(\boldsymbol{x}_t,\theta_t)}{\partial\theta_t}\right) \quad (25)$$

Note that one needs to use the softmax activation function for output nodes when learning with the cross entropy error function.

Compared to the standard gradient descent method and heuristic methods, the computational cost of the adaptive natural gradient descent method is high. However, since it can be applied to the on-line learning mode, it could be a good solution for the learning of large data sets and small size of network.

COMPUTATIONAL EXPERIMENTS

We conducted computational experiments for showing the efficiency of the adaptive natural gradient learning algorithm and some techniques mentioned above.

Since we are focusing on the problem including large data sets, we used only the on-line learning that is suitable for large data sets. Even though we presented various tips and learning algorithms in the previous section, we are focusing on showing the performance of the adaptive natural gradient learning method which has been recently developed, and showing how simple tips improve the learning speed, which is often ignored. The problem used is the thyroid disease problem, which is one of the well-known benchmark problems and can be obtained from the UCI machine learning repository (http://www.ics.uci.edu/~mlearn/MLRepository.html). The task is to determine whether or not a patient referred to the clinic is hypothyroid. Three classes are built: normal(not hypothyroid), hyperfunctioning, and subnormal functioning. Each observation has 21 attributes. Fifteen of these attributes have binary values, and the others have continuous values. The number of data points for the training set is 3,772, and the number of data points for the test set is 3,428.

The details of the network used are as follows: 21 input nodes, 5 hidden nodes in one hidden layer, 3 output nodes, and 1-of-3 coding scheme for the output. Considering the generalization performance we tried to use as small a number of hidden nodes as possible so that we can obtain the desirable training error. We exploited four approaches to the problem to compare the performance according to the various techniques as stated above. We conducted ten independent trainings for each approach with different initial values to get average results. The approaches that we chose are as follows:

Approach A: Raw input data + Squared error function + Basic updating rule
Approach B: Preprocessing for input data + Squared error function +Basic updating rule
Approach C: Preprocessing for input data + Cross entropy error function + Basic updating rule
Approach D: Preprocessing for input data + Cross entropy error function + Adaptive natural gradient algorithm with periodical update of the matrix $\hat{\boldsymbol{G}}_t^{-1}$

The basic learning rule is the standard gradient descent learning algorithm defined by Equation 5 with a constant learning rate. The learning rate for each algorithm is optimized to get the fast convergence and the high success rate empirically. We terminated the learning process when the sum of SE was smaller than 10^{-3} or when the number of the learning cycle exceeded 50,000. We regarded the learning task as a failure if the error did not fall below 10^{-3} before 50,000 cycles. The results of each approach are described in Tables 2, 3, 4, and 5, respectively. The relative processing times in the tables mean that the processing time for each trial of each approach divided by the processing time for a trial of approach A. The results showed that some techniques for learning can give remarkable improvements in the learning speed and that the adaptive natural gradient descent learning algorithm has the practical advantage, especially.

Table 2: Results for Approach A

Approach A	training error	learning cycles	relative processing time	classification rate for training data	classification rate for test data
Trial 1	0.0039	50000	(fail) 1.0	99.44	98.25
Trial 2	0.0039	50000	(fail) 1.0	99.28	97.96
Trial 3	0.0034	50000	(fail) 1.0	99.52	98.63
Trial 4	0.0037	50000	(fail) 1.0	99.31	97.99
Trial 5	0.0038	50000	(fail) 1.0	99.34	98.07
Trial 6	0.0042	50000	(fail) 1.0	99.28	98.13
Trial 7	0.0035	50000	(fail) 1.0	99.44	97.99
Trial 8	0.0035	50000	(fail) 1.0	99.36	98.13
Trial 9	0.0038	50000	(fail) 1.0	99.47	98.16
Trial 10	0.0037	50000	(fail) 1.0	99.34	97.93
Average over success	No success	No success	No success	No success	No success
Best result	0.0034	50000	1.0	99.52	98.63
Rate of success	0/10				

Table 3: Results for Approach B

Approach B	training error	learning cycles	relative processing time	classification rate for training data	classification rate for test data
Trial 1	0.0010	8123	0.16	99.84	98.67
Trial 2	0.0010	41844	0.84	99.79	98.51
Trial 3	0.0010	9814	0.20	99.84	98.63
Trial 4	0.0010	9088	0.18	99.84	98.80
Trial 5	0.0010	7025	0.14	99.84	98.60
Trial 6	0.0010	13667	0.27	99.84	98.63
Trial 7	0.0013	50000	(fail) 1.00	99.76	98.37
Trial 8	0.0012	50000	(fail) 1.00	99.79	98.57
Trial 9	0.0010	15941	0.32	99.81	98.71
Trial 10	0.0010	7840	0.16	99.84	98.89
Average over success	0.0010	14168	0.28	99.83	98.69
Best result	0.0010	7025	0.14	99.84	98.89
Rate of success	8/10				

Table 4: Results for Approach C

Approach C	training error	learning cycles	relative processing time	classification rate for training data	classification rate for test data
Trial 1	0.0010	962	0.09	99.76	98.51
Trial 2	0.0010	1875	0.04	99.87	98.75
Trial 3	0.0010	1285	0.59	99.84	98.45
Trial 4	0.0010	3294	0.07	99.87	98.60
Trial 5	0.0010	7440	0.15	99.84	98.16
Trial 6	0.0010	4348	0.09	99.87	98.37
Trial 7	0.0010	1531	0.08	99.84	98.60
Trial 8	0.0010	26638	0.53	99.84	98.37
Trial 9	0.0010	3546	0.07	99.84	98.40
Trial 10	0.0011	50000	(fail) 1.00	99.81	98.42
Average over success	0.0010	5658	0.19	99.84	98.47
Best result	0.0010	962	0.04	99.87	98.75
Rate of success	9/10				

Table 5: Results for Approach D

Approach D	training error	learning cycles	relative processing time	classification rate for training data	classification rate for test data
Trial 1	0.0010	81	0.04	99.89	98.63
Trial 2	0.0010	150	0.07	99.87	98.16
Trial 3	0.0010	84	0.04	99.92	97.93
Trial 4	0.0010	92	0.04	99.87	98.72
Trial 5	0.0010	135	0.06	99.84	98.25
Trial 6	0.0010	215	0.09	99.84	98.20
Trial 7	0.0010	57	0.02	99.87	98.10
Trial 8	0.0010	53	0.02	99.87	98.60
Trial 9	0.0010	42	0.02	99.87	98.63
Trial 10	0.0010	65	0.03	99.89	98.72
Average over success	0.0010	97	0.04	99.87	98.39
Best result	0.0010	42	0.02	99.92	98.72
Rate of success	10/10				

FUTURE TRENDS

The neural network models and their learning algorithms can be explained and investigated from various points of view. For example, by considering a neural network as a stochastic model, we can discuss its learning based on the statistical estimation theory. Bishop (1995) showed how the statistical theory can be applied to neural networks and their applications. The dynamics of learning can be also analyzed by statistical mechanical methodology. Saad and Solla (1995) gave a set of first-order differential equations that describe the dynamical evolution of the overlaps among hidden nodes. By simulating the dynamical equations, they claim that all hidden nodes try to learn the same weight values in the early stage of learning, and this phenomenon is a main reason for the plateaus. In addition, a geometrical approach for analyzing the characteristics of the neural manifold has been studied. Fukumizu and Amari (2000) showed that plateaus and local minima exist in the sub-manifold of a neural manifold, in which more than two hidden nodes have the same weight values. In the future, the combination of research from various fields may be essential in order to make a breakthrough in the field of neural networks.

On the other hand, even though we concentrate on the methods for accelerating the learning speed, there is another important point to be considered. We should note that the ultimate goal of learning is not to minimize the training error, but to accurately predict an output for a newly given input. Namely, we need to minimize the generalization error, not the training error, through the learning process. This kind of problem is called the generalization problem or model selection.

Considering the generalization problem, we first need a criterion of generalization performance of neural networks. Murata, Yoshizaka, and Amari (1994) proposed the network information criterion (NIC) for determining the number of hidden units of a neural network that maximizes the generalization performance. Mackey (1992) proposed a somewhat different criterion for stochastic models, which is called the Bayesian evidence. A more practical and popular method is to use some validation data that has not been used in the learning to estimate the generalization error of learning networks (Haykin, 1999; Larsen, Svarer, Andersen & Hansen, 1998).

We also need to consider how to achieve a good neural network model based on a criterion mentioned above. One simple method is the early stopping, which tries to stop learning before a learning network is overtrained by noisy training data (Haykin, 1999; Prechelt, 1998). Another method for achieving the good generalization performance through learning is the regularization method (Bishop, 1995; Sigurdsson, Larsen, & Hansen, 2000). By adding some extra terms to the standard error function we can expect to get a smooth mapping function. On the other hand, we can also change the number of hidden nodes during or after the learning. The growing method is to grow the network structure until we get a desirable error. The cascade correlation method (Fahlman & Lebiere, 1990) is one of the well-known growing methods. On the contrary, the pruning method is to delete unnecessary parameters from a trained network. Since the Optimal Brain Damage (LeCun, Denker & Solla, 1990) and the Optimal Brain Surgeon (Hassibi, Stork, & Wollf, 1992) were developed, there has been much study to extend or to improve them (Haykin,1999; Laar & Heskes, 1999). In future research, it is necessary and important to consider these issues together with learning efficiency in order to get a total solution for learning machines. Finally, the hardware implementation of neural networks is also an important issue for future studies.

CONCLUSIONS

In this chapter, we consider the training of neural networks with large data sets, which often occurs in the field of data mining, and give explicit algorithms of the adaptive natural gradient learning as a solution. In addition, we also explain various heuristics and methods for accelerating the learning speed of MLP. It should be noted that there is no best method from the overall viewpoint because all of them have their own merits and defects. We finally summarize the tips for selecting a suitable method according to the characteristics of the problems to be solved.

1. For a given data set, we need some preprocessing. Use 1-of-L coding scheme for output data in the case of a classification problem. Transform the input data by using Equations 6, 7 and 8. Make sure that the same preprocessing is applied to the test data.
2. Choose an error function suitable for the given application. In the case of the regression problem, such as the prediction of stock prices, use the sum of the SE function. In the case of the classification problem, such as ranking consumers' levels, the cross entropy error function is better. The learning scheme needs to be decided at this stage, but generally the on-line learning is recommended.
3. Choose the number of hidden nodes by considering the number of input nodes, the number of output nodes, and the number of observations. Generally, the necessary number of hidden nodes is proportioned to the number of training data and output nodes, whereas it is proportioned reciprocally to the number of input nodes.
4. Choose an activation function. For the hidden nodes, any of the sigmoidal activation functions can be used. For the output nodes, however, if the output values of learning data are continuous, one should use the linear activation function. If the output values are binary, the sigmoidal activation function is better. Note that the softmax activation function should be used when the cross entropy error function is exploited.
5. Initialize network parameters with small random values.
6. Choose a learning algorithm. If the network is large and the learning data set is small, the standard gradient descent method with the momentum term or the adaptive learning rate may give a reasonable convergence speed. However, when the size of a network is small, more sophisticated methods such as the second order methods and the adaptive natural gradient methods are better.

Especially when the data set is large, the adaptive natural gradient method is a better choice.

REFERENCES

Amari, S. (1998). Natural gradient works efficiently in learning, *Neural Computation, 10*, 251-276.

Amari, S. & Nagaoka, H. (2000). *Methods of Information Geometry*, AMS and Oxford University Press.

Amari, S., Park, H., & Fukumizu, F. (2000). Adaptive method of realizing natural gradient learning for multilayer perceptrons, *Neural Computation, 12*, 1399-1409.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*, New York: Oxford University Press.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function, *Mathematical Control Signals Systems, 2*, 303-314.

Fahlman, S. H. & Lebiere, C. (1990). The Cascade-Correlation Learning Architecture, *Advances in Neural Information Processing Systems, 2*, 524-532.

Fukumizu, F. & Amari, S. (2000). Local Minima and Plateaus in Hierarchical Structures of Multilayer Perceptrons, *Neural Networks, 13*, 317-328.

Hassibi, B., Stork, D.G., & Wollf, G. J. (1992). Optimal Brain Surgeon and General Network Pruning, *IEEE International Conference on Neural Networks, 1*, 293-299.

Hassoun, M. H. (1995). *Fundamentals of Artificial Neural Networks,* The MIT Press.

Haykin, S. (1999). *Neural Networks-A Comprehensive Foundation*, 2ed., New Jersey: Princeton Hall International, Inc.

Jacobs, R. A. (1988). Increased Rates of Convergence Through Learning Rate Adaptation, *Neural Networks, 1*, 295-307.

Laar, P. & Heskes, T. (1999). Pruning Using Parameter and Neuroal Metrices, *Neural Computation, 11*, 977-993.

Larsen, J., Svarer, C., Andersen, L. N., Hansen, L. K. (1998). Adaptive Regularization in Neural Network Modeling, In G. B. Orr and K. R. Müller, *Neural networks: tricks of the trade, Springer Lecture Notes in Computer Sciences, 1524 (113-132)*, Heidelberg: Springer.

LeCun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal Brain Damage, *Advances in Neural Information Processing Systems, 2*, 598-605.

LeCun, Y., Bottou, L., Orr G. B., & Müller, K.-R. (1998). Efficient BackProp, In G. B. Orr and K. R. Müller, *Neural networks: tricks of the trade, Springer Lecture Notes in Computer Sciences, 1524*, (9-53) Heidelberg: Springer.

Mackey, D. J. (1992). Bayesian Interpolation, *Neural Computation, 4*, 415-447.

Michalski, R. S., Bratko, I., & Kubat, M. (1998). *Machine Learning and Data Mining – Methods and Applications*, New York: John Wiley & Sons Ltd.

Murata, M., Yoshizaka, S. and Amari, S. (1994). Network Information Criterion-Determining the Number of Hidden Units for an Artificial Neural Network Model, *IEEE Transactions on Neural Networks, 5(6)*, 865-872.

Orr, G. B. (1995). *Dynamics and Algorithms for Stochastic Search*, Ph.D. Dissertation, Oregon Graduate Institute of Science and Technology, U.S.A.

Park, H. & Amari. S. (1998). Escaping from Plateaus of Multilayer Perceptron Learning by Natural Gradient, *The 2nd RIEC International Symposium on Design and Architecture of*

Information Processing Systems Based on the Brain Information Principles, 189-192.

Park, H., Amari, S., & Fukumizu, K. (2000), Adaptive Natural Gradient Learning Algorithms for Various Stochastic Models, *Neural Networks, 13*, 755-764.

Prechelt, L. (1998). Early Stopping—But When? In G. B. Orr and K. R. Müller, *Neural networks: tricks of the trade, Springer Lecture Notes in Computer Sciences, 1524*, (54-69) Heidelberg: Springer.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*, Cambridge University Press.

Rattray, M. & Saad, D. (1998). Incorporating curvature information into on-line learning, In D. Saad (Ed.), *On-Line Learning in Neural Networks* (pp.183-207), U.K.: Cambridge University Press.

Rattray, M. & Saad, D. (1999). Analysis of natural gradient descent for multilayer neural networks, *Physical Review E 59(4)*, 4523-4532.

Ridella, S., Rovetta, S., & Zunino, R. (1997). Circular Backpropagation Networks for Classification, *IEEE Transactions on Neural Networks, 8*, 84-99.

Riedmiller, M. & Braun, H. (1992). *RPROP – A Fast Adaptive Learning Algorithm*, Technical Report, Universität Karlsruhe.

Rumelhart, D. E. & McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 1*, Cambridge, MA: MIT Press.

Saad, D. & Solla, S. A. (1995). On-line Learning in Soft Committee Machines, *Physical Review E, 52*, 4225-4243.

Sigurdsson, S. Larsen, J. & Hansen, L. K. (2000). On Comparison of Adaptive Regularization Methods, *IEEE Workshop on Neural Networks for Signal Processing*, 221-230.

SECTION FIVE

APPLICATIONS

Chapter XII

Cluster Analysis of Marketing Data Examining On-line Shopping Orientation: A Comparison of *k*-means and Rough Clustering Approaches

Kevin E. Voges, Nigel K. Ll. Pope and Mark R. Brown
Griffith University, Australia

Cluster analysis is a common market segmentation technique, usually using k-means clustering. Techniques based on developments in computational intelligence are increasingly being used. One such technique is the theory of rough sets. However, previous applications have used rough sets techniques in classification problems, where prior group membership is known. This chapter introduces rough clustering, a technique based on a simple extension of rough sets theory to cluster analysis, and applicable where group membership is unknown. Rough clustering solutions allow multiple cluster membership of objects. The technique is demonstrated through the analysis of a data set containing scores on psychographic variables, obtained from a survey of shopping orientation and Web purchase intentions. The analysis compares k-means and rough clustering approaches. It is suggested that rough clustering can be

considered to be extracting concepts from the data. These concepts can be valuable to marketers attempting to identify different segments of consumers.

INTRODUCTION

Cluster analysis has been a fundamental technique in marketing research for many decades, both as a general data reduction technique and as the basis for market segmentation (Arabie & Hubert, 1994; Punj & Stewart, 1983). While there are numerous ways to undertake market segmentation, the grouping of similar objects through cluster analysis remains one of the fundamental starting points. In marketing research, objects are usually the measured demographic or psychographic characteristics of consumers. Forming groups that are homogenous with respect to these measured characteristics segments the market. One psychographic measure often used in segmentation studies is shopping orientation.

Consumers go shopping for a variety of reasons, not just for the procurement of goods (Tauber, 1972). Reasons may include social interaction, sensory stimulation, role enactment, and physical exercise, to name a few (Tauber, 1972). The psychographic characteristic of shopping orientation refers to the general predisposition of consumers toward the act of shopping and has been used to partially explain retail shopping behaviour (Dholakia, Pedersen & Hikmet, 1995). Six core shopping orientations have been identified in the published marketing literature: economic, recreational, apathetic, convenience-oriented, ethical, and personalising (Brown, 1999).

Economic shoppers are essentially concerned with buying products at the lowest price or getting the best value for the money they spend (Bellenger & Korgaonkar, 1980; Shim & Mahoney, 1992). *Recreational* shoppers enjoy the act of shopping regardless of whether a purchase is made or not (Bellenger & Korgaonkar, 1980). *Apathetic* or inactive shoppers are mostly interested in minimizing shopping effort (Darden & Reynolds, 1971). *Convenience-oriented* shoppers are those under time constraints and possibly also under space and effort constraints (Gehrt, Yale & Lawson, 1996). *Ethical* shoppers can be distinguished by their loyalty, with studies investigating store loyalty, brand loyalty, or both (Darden & Reynolds, 1971). *Personalizing* shoppers demonstrate a propensity to value relationships with suppliers (Darden & Reynolds, 1971; Peppers & Rogers, 1997).

The starting point for many of these market segmentation studies has been cluster analysis. Many clustering methods have been identified, including partitioning, hierarchical, nonhierarchical, overlapping, and mixture models (Arabie & Hubert, 1994; Hair, Anderson, Tatham & Black, 1998). One of the most commonly used nonhierarchical methods is the *k*-means approach. This approach will be considered in more detail in the following section.

In the last few decades many new techniques based on developments in computational intelligence have started to be more widely used as clustering algorithms. For example, the theory of fuzzy sets developed by Zadeh (1965) introduced the concept of partial set membership as a way of handling imprecision in mathematical modeling. Hruschka (1986) has applied the theory to market segmentation. Neural networks, another technique from computational intelligence, have also been applied to a range of marketing problems (Mazanec, 1992; Venugopal & Baets, 1994). A third technique from computational intelligence currently receiving considerable attention is the theory of rough sets (Pawlak, 1981, 1982).

Rough sets theory is also being applied to an increasing number of marketing problems (Kowalczyk & Slisser, 1997; Van den Poel & Piasta, 1998). In these papers, the technique has been used in classification problems, where prior group membership is known, and results are usually expressed in terms of rules for group membership (Pawlak, 1984). This chapter introduces rough clustering, a new technique based on a simple extension of rough sets theory, and applicable where prior group membership is not known. The technique is demonstrated through the analysis of a data set containing 437 responses to an Internet-based survey investigating purchase intentions and behaviour over the Web (Brown, 1999). The example uses five psychographic variables relating to measures of shopping orientation. The results of this analysis are compared with the results of a standard nonhierarchical clustering method using *k*-means.

The following Background section presents an overview of issues in cluster analysis in marketing research, a brief overview of the *k*-means approach, a more extended introduction to rough sets theory, and an introduction to rough clustering. The Results section of the chapter presents the results of the *k*-means and rough clustering approaches. The Conclusion compares the results of both clustering approaches and outlines some possible extensions.

BACKGROUND

Cluster Analysis in Marketing Research

Cluster analysis is an important area of application in both marketing research and data mining. Hartigan (1975, p. 1) offered an early definition of clustering as "the grouping of similar objects." A more comprehensive definition is provided by Hair et al. (1998, p. 470): "Cluster analysis groups individuals or objects into clusters so that objects in the same cluster are more similar to one another than they are to objects in other clusters." There is an extensive literature on cluster analysis in marketing. [For comprehensive review see, for example, Arabie and Hubert (1994) and Punj and Stewart (1983).]

A common application of cluster analysis is the segmentation of a market by identifying homogeneous groups of buyers (Beane & Ennis, 1987; Punj & Stewart, 1983). Market segmentation has been a central idea in marketing theory and practice since the introduction of the concept by Smith (1956). Smith recognized the existence of heterogeneity in the market demand for goods and services and viewed this heterogeneous market as consisting of a number of smaller homogeneous submarkets. These submarkets consist of groups of consumers who respond differently to the marketing mix (Dickson & Ginter, 1987; Mahajan & Jain, 1978; Wind, 1978). Wind (1978) identified many of the problems of market segmentation and concluded that clustering was the principal method for identifying homogeneous subgroups of consumers within a market. Marketing applications of cluster analysis include studies of price sensitivity (Blozan & Prabhaker, 1984), patterns of information search (Furse, Punj & Stewart, 1984), brand loyalty and brand switching (Grover & Srinivasan, 1992), and consumer choice models (Currim & Schneider, 1991).

Non-Hierarchical Cluster Analysis

Traditionally, non-hierarchical methods of cluster analysis in marketing have been based on the *k*-means approach (MacQueen, 1967). Data points are randomly selected as initial seeds or centroids, and the remaining data points are assigned to the closest centroid on the basis of the distance between them. The aim is to obtain maximal homogeneity within subgroups or clusters and maximal heterogeneity between clusters.

Consider a data matrix with M cases ($1 \le i \le M$) and N variables ($1 \le j \le N$), with the ith case of the jth variable having value $R(i, j)$. The partition of this data matrix obtained by the cluster analysis, $P(M, K)$, contains K clusters. Each of the M cases lies in only one of the K clusters. Let $N(k)$ denote the number of cases in cluster k, and $C(k, j)$ denote the mean of the jth variable over the cases in the kth cluster. The (Euclidean) distance, D, between the ith case and the kth cluster is defined as:

$$D(i,k) = \left(\sum_{j=1}^{N} [R(i,j) - C(k,j)]^2 \right)^{1/2}$$

The error of the partition is:

$$e[P(M,k)] = \sum_{i=1}^{M} \left(D[i,k(i)]\right)^2$$

where $k(i)$ is the cluster containing the ith case. The usual approach is to search for a partition with small e by moving cases from one partition to another (Hartigan, 1975).

The search through the problem space to find the lowest value of e is considered computationally expensive and traditionally local optimization has been used. The number of clusters in each partition is decided prior to the analysis, and centroids are selected as the initial estimates of cluster centres. Objects in the data set are allocated to the closest cluster (usually using the Euclidean distance), the cluster centroid is often updated after each object is added, and the process continues until the local minimum for $e[P(M, K)]$ is found. The partition obtained is not necessarily the global minimum of e. *K*-means cluster analysis has been shown to be both robust and less affected by data idiosyncrasies than hierarchical clustering techniques (Punj & Stewart, 1983). Despite this, the approach suffers from many of the problems associated with all traditional multivariate statistical analysis methods. These methods were developed for use with variables which are normally distributed and which have an equal variance-covariance matrix in all groups. In most realistic marketing data sets, neither of these conditions necessarily holds.

In an attempt to overcome the limitations of these conditions, other approaches to data analysis have been used. These approaches are often used in data mining and are of increasing importance in marketing (Berry & Linoff, 1997; Voges, 1997; Voges & Pope, 2000). Most of the techniques have been derived from advances in computational intelligence. The major advantage of such techniques is that most of them make few assumptions about the statistical characteristics of the data being analysed. This paper explores the application of one of these new techniques, rough sets, to cluster analysis.

Rough Sets

The concept of rough or approximation sets was introduced by Pawlak (1981, 1982, 1984, 1991) and is based on the assumption that with every record in the data matrix (in rough set terminology, every object of the information system) there is associated a certain amount of information. This information is expressed by means of some variables (in rough set terminology, attributes) used as descriptions of the objects. The data is treated from the perspective of set theory and none of the traditional assumptions of multivariate analysis are relevant. For a comprehensive introduction to rough set theory, see Pawlak (1991) or Munakata (1998).

Rough sets techniques differ from other techniques in their attitude towards the data. The initial detailed data is used as the basis for the development of subsets of the data that are "coarser" or "rougher" than the original set. As with any data analysis technique, detail is lost, but the removal of detail is controlled to uncover the underlying characteristics of the data. The technique works by "lowering the degree of precision in data, based on a rigorous mathematical theory. By selecting the right roughness or precision of data, we will find the underlying characteristics" (Munakata, 1998, p. 141).

In rough sets theory, the data matrix is represented as a table, the information system. The complete information system expresses all the knowledge available about the objects being studied. More formally, the information system is a pair $S = (U, A)$, where U is a non-empty finite set of objects called the universe and $A = \{ a_1, ..., a_j \}$ is a non-empty finite set of attributes on U. With every attribute $a \in A$ we associate a set Va such that $a : U \rightarrow Va$. The set Va is called the domain or value set of a.

A core concept of rough sets is that of indiscernibility. Two objects in the information system about which we have the same knowledge are indiscernible. This indiscernibility leads to redundancy within the information system, which can make it unnecessarily large. This can happen in two ways. Objects may be represented several times (i.e., two objects in the information set may have the same values for their attributes), or some of the attributes may be superfluous (i.e., two of the variables may have a correlation of 1.0). Consequently one attribute could be removed without losing any of the information in the information system).

Let $S = (U, A)$ be an information system, then with any $B \subseteq A$ there is associated an equivalence relation, $IND_A(B)$, called the B-indiscernibility relation. It is defined as:

$$IND_A(B) = \{ (x, x') \in U^2 \mid \forall a \in B \; a(x) = a(x') \}$$

If $(x, x') \in IND_A(B)$, then the objects x and x' are indiscernible from each other when considering the subset B of attributes.

Equivalence relations lead to the universe being divided into partitions, which can then be used to build new subsets of the universe. Let $S = (U, A)$ be an information system, and let $B \subseteq A$ and $X \subseteq U$. We can describe the set X using only the information contained in the attribute values from B by constructing the B-lower and B-upper approximations of X, denoted $B_*(X)$ and $B^*(X)$ respectively, where:

$$B_*(X) = \{ x \mid [x]_B \subseteq X \}, \text{ and}$$
$$B^*(X) = \{ x \mid [x]_B \cap X \neq \emptyset \}$$

The set $BN_B(X)$ is referred to as the boundary region of X, and is defined as:

$$BN_B(X) = B^*(X) - B_*(X)$$

If the boundary region of X is the empty set ($BN_B(X) = \emptyset$), then X is a crisp (exact) set with respect to B. If the boundary region is not empty ($BN_B(X) \neq \emptyset$), X is referred to as a rough (inexact) set with respect to B.

Most of the published applications literature in rough sets has concentrated on a specific type of information system, the decision system. In a decision system at least one of the attributes is a decision attribute. This decision attribute partitions the information system into groups (in rough set terminology, concepts). In this form, rough sets analysis performs the same type of classification function as discriminant analysis, where there is a known subgrouping identified by the decision attribute. The problem is expressed in rough set theory as finding mappings from the partitions induced by the equivalence relations in the condition attributes to the partitions induced by the equivalence relations in the decision attribute(s). These mappings are usually expressed in terms of decision rules. More formally, with an information system $S = (U, A)$, we can associate a formal language $L(S)$. Expressions in this language are logical formulas built up from attributes and attribute-value pairs and standard logical connectives (Pawlak, 1999). A decision rule in L is an expression $\phi \rightarrow \psi$ (read if ϕ then ψ), where ϕ and ψ are, respectively, the conditions and decisions of the rule.

For any concept in the information system, rules induced from its lower approximation are called certain, as by definition they are valid rules. Rules induced from the boundary region of the concept are called uncertain, as they can lead to different values for a decision attribute. A confidence factor can be defined as the number of objects in the condition attribute subset that also satisfy the decision subset (concept), divided by the total number of objects in the condition attribute subset.

Let ϕ_i be a partition of the condition attributes and ψ_i be a partition of the decision attribute (concept). The confidence factor, α, for a rule $\phi \rightarrow \psi$ is:

$$\alpha = \frac{|\phi_i \cap \psi_i|}{|\phi_i|}$$

where | A | is the cardinality of set A.

The canonical version of rough sets theory as briefly presented above has been extended in a number of ways (Yao, Wong & Lin, 1997). One common extension is that of probabilistic rough sets (Pawlak, Wong & Ziarko, 1988). Other extensions include modified rough sets using genetic algorithms (Hashemi, Pearce, Arani, Hinson & Paule, 1997), rough genetic algorithms (Lingras & Davies, 1999), and the relationship between rough sets and concept lattices (Hu, Lu, Zhou & Shi, 1999; Oosthuizen, 1994). In this paper, the focus is on the information table, not the decision table. Applications of this nature are still rare in the literature, although Pawlak (1991) devotes a chapter to it. However, he used the information table to analyse dissimilarity, rather than similarity. We apply rough set theory to discovering clusters in a data set (cluster analysis) based on a similarity (or distance) measure for objects with ordered values of attributes. The approach has been called rough clustering.

Rough Clustering

Rough clusters are a simple extension of the notion of rough sets. A measure of the distance between each object is defined, and clusters of objects are formed on the basis of this distance measure. The value set needs to be ordered to form a meaningful distance measure. Clusters are formed in a similar manner to agglomerative hierarchical clustering (Hair et al., 1998). However, an object can belong to more than one cluster. Clusters can then be defined by a lower approximation (objects exclusive to that cluster) and an upper approximation (all objects in the cluster which are also members of other clusters), in a similar manner to rough sets.

Let $S = (U, A)$ be an information system, where U is a non-empty finite set of M objects ($1 \leq i \leq M$), and A is a non-empty finite set of N attributes ($1 \leq j \leq N$) on U. The jth attribute of the ith object has value $R(i, j)$ drawn from the ordered value set Va.

For any pair of objects, p and q, the distance between the objects is defined as:

$$D(p,q) = \sum_{j=1}^{N} |R(p,j) - R(q,j)|$$

That is, the absolute differences between the values for each object pair's attributes are summed. The distance measure ranges from 0 (indicating indiscernible objects) to a maximum determined by the number of attributes and the size of the value set for each attribute. In the example discussed in this chapter, there are five attributes ranging from 1 to 7, so the maximum possible distance between any two objects would be 30. As will be seen later, considering small distances up to 5 can form viable clusters.

The algorithm for producing rough clusters is as follows. Initially, a distance matrix for all paired object comparisons is calculated. As this matrix is symmetric, only the lower triangle is calculated, reducing computational load. All object pairs at interobject distance *D*, where *D* steps from 1 to a determined maximum, are identified. Each object pair (a_j, a_k) can be in one of three situations in relation to current cluster membership, with the following consequences:

1. Both objects have not been assigned to any prior cluster. A new cluster is started with a_j and a_k as the first members.
2. Both objects are currently assigned to clusters. Object a_j is assigned to object a_k's earliest cluster, and object a_k is assigned to object a_j's earliest cluster. The earliest cluster is the first cluster the object was assigned to.
3. One object, a_j is assigned to a cluster and the other object, a_k is not assigned a cluster. Object a_k is assigned to object a_j's earliest cluster.

Increasing the maximum inter-object distance in this systematic way leads to a lowering of precision in a controlled manner, with objects becoming members of rough clusters. An application of this simple algorithm to the survey data is reported in the following section.

RESULTS

Description of the Data Set

The comparison between *k*-means cluster analysis and rough set cluster analysis was conducted on a sample of 437 useable responses from a larger study of the relationship between shopping orientation, perceived risk and intention to purchase products via the Internet (Brown, 1999; Brown, Pope & Voges, 2001). The cluster analysis presented here was based on five measures of shopping orientation: enjoyment, personalization, convenience, loyalty, and price. All measures were constructed as Likert-type statements with responses ranging from strongly disagree (1) to strongly agree (5). A brief description of the operationalization of these five psychographic measures follows. More detail is available in Brown (1999).

Shopping enjoyment measures the degree to which individuals take pleasure from the shopping process itself, regardless of whether they purchase or not. The construct was measured with a five-item summated scale based on the original scale of Bellenger and Korgaonkar (1980). A modified three-item scale developed by Hawes and Lumpkin (1984) was used to assess the extent to which individuals possess a *personalizing orientation* toward shopping. *Convenience* was measured using a three-item scale initially employed by Shamdasani and Ong (1995) to measure the importance of convenience to in-home shoppers. *Loyalty* was measured by a three-item scale developed by Hawes and Lumpkin (1984). Consumers' propensity to be *price conscious* when engaging in shopping activities was measured using a four-item scale developed by Tat and Schwepker (1998).

As rough clustering requires ordered discretized data, the factor score data was mapped onto an ordered variable with a range of seven. Discretization is the process by which a set of values is grouped together and is an ongoing problem in data mining approaches (Komorowski, Pawlak, Polkowski & Skowron, 1999). In this research, each variable (originally obtained as factor scores) was mapped onto an ordered variable with each value for the variable representing 14 to 15 percent of the data set. This discretization scheme was found to provide a rough cluster solution that was most comprehensible.

Non-Hierarchical Cluster Analysis

Non-hierarchical cluster analysis of the sample was undertaken, based on responses to the core shopping orientation scales (Brown, 1999). Testing the stability of the cluster solution was conducted in accordance with the recommendations of Punj and Stewart (1983). The seven-cluster solution was used as it had the highest kappa value of 0.90, and this was subsequently found to provide more efficient and interpretable results than the six-cluster solution, which was used as a comparison. After choosing the most appropriate number of clusters, the entire sample was then subjected to *k*-means analysis to obtain a final cluster solution. These clusters were interpreted according to the centroids of the shopping orientations within each cluster. A centroid of zero represented a neutral position toward a shopping orientation.

Positive centroids were considered to be indicative of the nature of a cluster. The higher the centroid, the more a cluster was interpreted as being oriented toward that construct. Conversely, negative centroids indicated the degree to which a construct was unimportant to members of that cluster. Table 1 shows the cluster centroids for the seven-cluster solution. As expected, multiple shopping orientations are identifiable. These shoppers can be differentiated by the degree to which

Table 1: Cluster means and standard deviations for shopping orientation variables k-means clustering

	Shopping Orientation				
	Enjoyment	Loyalty	Price	Convenience	Personalizing
	Mean (sd)	Mean (sd)	Mean (sd)	Mean (sd)	Mean (sd)
Cluster					
1	*-1.30*(1.66)	-0.60 (1.55)	*-2.76* (0.48)	0.02 (1.57)	*0.90* (1.78)
2	*1.92*(1.09)	-0.44 (1.70)	-0.33 (1.82)	*-2.11* (1.02)	0.14 (1.72)
3	*-2.13*(0.96)	-0.47 (1.73)	*1.12* (1.42)	*-1.72* (1.20)	-0.37 (1.80)
4	0.42(1.77)	*1.31* (1.37)	*1.12* (1.65)	0.88 (1.69)	*2.34* (0.75)
5	*1.33*(1.57)	-0.15 (1.59)	*1.10* (1.65)	*1.46* (1.15)	*-1.79* (1.18)
6	-0.31(1.64)	*2.41* (0.75)	*-1.03* (1.52)	0.33 (1.77)	*-1.36* (1.41)
7	-0.30(1.72)	*-2.61* (0.56)	0.22 (1.79)	*1.85* (1.25)	0.15 (1.88)

Centroids equal to or greater than (+ or -) 0.90 are shown in italics they responded to shopping orientation statements. To aid in interpretation, centroids equal to or greater than ± 0.90 have been highlighted. Positive numbers equal to or above 0.90 suggest a strong positive orientation towards that particular aspect of shopping. Negative numbers equal to or below –0.90 suggest a strong negative orientation towards that aspect. In other words, that particular aspect would be sacrificed. For example, Cluster 3 in Table 1 could be interpreted as a shopper looking for the best price, who is prepared to sacrifice convenience and enjoyment to obtain that price.

Cluster 1 members showed moderately high values on the personalising shopper dimension, with negative responses to enjoyment and price. A high value on the shopping enjoyment factor and a strong negative value for convenience characterise Cluster 2. Price consciousness was the positive factor in Cluster 3, with enjoyment and convenience strongly negative. Values for Cluster 4 were moderate to high positive on the loyalty, price and personalising shopping orientations. Cluster 5 had similar positive values on the convenience, shopping enjoyment, and price dimensions and a strongly negative value on the personalising dimension. Cluster 6 was characterised by their preference for patronising local merchants and low concern for price and personal shopping. Cluster 7 was characterised by its high score on the convenience orientation and a low score on loyalty.

Rough Cluster Analysis

Rough clusters are based on a simple distance measure between objects in the data set as described above. Objects can belong to more than one cluster. Consequently, rough clustering produces more clusters than standard cluster analysis. The number of clusters that is required to account fully for the data is a function of the interobject distance. In addition, the lower approximation of each cluster is dependent on the number of clusters selected for the solution. More clusters in the solution means an object has more chance of being in more than one cluster, therefore moving from the lower approximation to the boundary region and reducing the size of the lower approximation.

As outlined above, the algorithm steps the interobject distance D from 1 to a set maximum. Table 2 shows the number of objects assigned to clusters, the percentage of the data set that this represents, and the number of clusters obtained

*Table 2: Data set coverage for different values of interobject distance (*D*)*

D	Number of Objects	% of Data	Number of Clusters
1	69	15.8	31
2	291	66.6	98
3	424	97.0	76
4	437	100.0	39
5	437	100.0	24
6	437	100.0	18

for each maximum *D* value from 1 to 6. As would be expected, using a maximum *D* of 1 only identifies a limited number of similar objects. As shown in Table 2, a maximum *D* of 4 fully accounts for the data set in 39 clusters. Increasing the maximum *D* to 5 and 6 fully accounts for the data set in 24 and 18 clusters, respectively.

A number of factors need to be considered when determining the best maximum value for *D* and the best number of clusters to include in the solution. The number of clusters chosen needs to cover a reasonable proportion of the data set. As described below, we defined a "reasonable proportion" as over 90% of the objects in the information system. A solution with too few factors will not provide a particularly useful interpretation of the partitioning of the data. On the other hand, too many clusters will make interpretation difficult. In addition, the degree of overlap between the clusters should be minimised to ensure that each cluster identified provides additional information to aid with interpretation. One way to achieve this is to maximise the sum of the lower approximations of the clusters being used to provide a possible solution. Determining a good solution requires a trade-off between these factors. Tables 3 and 4 and related discussion report an analysis of a number of possible solutions to determine the best solution for the current data set.

Table 3 shows what happens to the cluster solution as *D* is progressively increased from 2 to 6. For clarity, only values for the first twelve clusters obtained from these five separate maximum *D* solutions are shown. The table shows the size of the upper approximation for these twelve clusters for each value of maximum *D*.

Table 3: Size of upper and lower approximations for values of maximum D *from 2 to 6*

	Max D = 2			Max D = 3			Max D = 4			Max D = 5			Max D = 6		
	$\|B^*(k)\|$	$\|B_*(k)\|$		$\|B^*(k)\|$	$\|B_*(k)\|$		$\|B^*(k)\|$	$\|B_*(k)\|$		$\|B^*(k)\|$	$\|B_*(k)\|$		$\|B^*(k)\|$	$\|B_*(k)\|$	
		12	6		12	6		12	6		12	6		12	6
1	11	11	11	40	14	26	85	8	15	139	4	17	186	2	10
2	10	7	10	35	25	29	69	28	47	116	12	54	186	0	0
3	10	10	10	30	23	28	82	2	23	131	0	14	160	4	43
4	8	8	8	25	6	18	57	22	35	121	1	8	186	0	9
5	7	7	7	25	15	18	62	18	27	113	9	19	161	4	15
6	7	7	7	19	14	17	61	6	19	94	8	38	171	7	23
7	7	7		34	6		62	1		96	8		141	4	
8	6	6		25	14		54	10		105	2		114	5	
9	6	6		17	14		40	16		105	2		133	0	
10	6	6		23	12		50	7		87	8		130	2	
11	8	3		20	13		48	11		83	10		122	3	
12	7	5		17	12		46	9		81	6		137	1	
$\Sigma\ \|B_*(k)\|$															
		83	53		168	136		138	166		70	150		32	100

Two values of the lower approximation are shown; the first is the value when all twelve clusters are considered, and the second is the value when only the first six clusters are considered. For the first two solutions, the sum of the lower approximations is larger for the 12 clusters than for the 6 clusters. Further investigation showed that for these two values of maximum *D*, the sum of the lower approximations continues to increase as the number of clusters is increased. This shows that for these two solutions, the algorithm is mainly performing Step 1, that is, as new objects are being incorporated into the solution they are being assigned to new clusters.

For a maximum *D* value of 4, this pattern reverses. That is, the sum of the lower approximations for the first six clusters is larger than the sum of the lower approximations for the first twelve clusters. The sum of lower approximations then decreases as the value of maximum *D* increases. This shows that for maximum values for *D* of 4 and above, the rough clustering algorithm is starting to assign objects based on Steps 2 and 3, that is, as new objects are being incorporated into the solution they are being assigned to existing clusters. This suggests that the best maximum value for *D* is at least 4.

Table 4 shows the results of applying varying criteria relating to the percentage of objects assigned to the cluster solution. The table shows the percentage of data assigned to clusters, the number of clusters, and the sum of the lower approximations for minimum data set coverage of 70%, 80%, and 90%. The maximum *D* value of 3 has been shown for comparison only. While this produces the largest sum of lower approximations for all percentages of data set coverage shown, it requires a large number of clusters to achieve this result. Only maximum *D* values of 4, 5, or 6 will be considered as providing viable solutions. Table 4 shows that if we account for only 70% of the data set, a maximum *D* of 4 and nine clusters produces the best solution (defined as maximising the sum of lower approximations).

When accounting for at least 80% or 90% of the coverage, the best solution is obtained with a maximum *D* of 5. For a minimum of 80% data set coverage, six clusters with a sum of lower approximations of 150 is obtained. For a minimum of 90% data set coverage, nine clusters with a sum of lower approximations of 108 is obtained. The analysis presented in Tables 3 and 4 suggests that a nine-cluster solution with a maximum *D* of 5 produces the best solution, taking into account the factors suggested above. This solution accounts for over 90% of the data set and produces a solution that does not contain too many clusters.

Table 5 shows the cluster centroids for the rough cluster solution for interobject distance 5, set out in a similar manner to Table 1. As with the *k*-means solution, multiple shopping orientations are identifiable, with shoppers being differentiated by the degree to which they responded to statements regarding shopping orientation. Again, positive numbers above 0.90 suggest a strong positive orientation towards that particular aspect of shopping. Negative numbers below –0.90 suggest a strong negative orientation towards that aspect. Cluster 2 in Table 5, for example, could be interpreted as a shopper looking for the best price, who is prepared to sacrifice convenience to obtain that price.

Cluster 1 shows a strong loyalty orientation, Cluster 3 shows loyalty at the expense of convenience, and Cluster 4 shows loyalty at the expense of price. Cluster 5 shows positive scores on enjoyment, loyalty and convenience. Three of the clusters show complex combinations of orientations, including Cluster 5 described above. Cluster 6 shows a concern for price and convenience at the expense of loyalty, while Cluster 7 shows concern for enjoyment, price and personal shopping. Two of the clusters form combinations of orientations that might be considered more difficult to interpret or, at best, reflections of a negative shopping orientation. Cluster 9 has a negative value for convenience, and Cluster 8 shows concern for personal shopping at the expense of enjoyment.

CONCLUSION

Comparison of Cluster Solutions

To distinguish between the two solutions, clusters from the *k* -means solution (Table 1) are referred to as Clusters, while clusters from the rough clustering solution (Table 5) are referred to as Rough Clusters. Cluster 1 showed a moderately high positive score on the personalizing dimension. Two of the nine rough clusters also had high scores on personalizing, but neither of them matched the negative

Table 4: Number of clusters and sum of lower approximations for different percentage coverage of the data set and different values of D.

	D	% of Data	Number of Clusters	$\Sigma\,|B_*(k)|$
At least 70% coverage				
	3	70.3	22	203
	4	73.2	9	155
	5	74.6	5	144
	6	78.0	4	131
At least 80% coverage				
	3	80.5	31	208
	4	80.5	12	138
	5	80.8	6	150
	6	85.1	5	107
At least 90% coverage				
	3	90.4	49	197
	4	90.6	18	83
	5	90.2	9	108
	6	90.8	6	100

values for enjoyment and price of Cluster 1. The closest match was Rough Cluster 8, which had a negative value for enjoyment. In contrast, Rough Cluster 7 had positive values for enjoyment and price as well as personalizing.

Cluster 2 showed a high positive value for enjoyment. Rough Cluster 5 also had a high positive value for enjoyment, but it did not show the high negative value for convenience characteristic of Cluster 2. Rough Cluster 9 shows this high negative value for convenience but does not match Cluster 2 on the other dimensions. As mentioned above, Rough Cluster 9 presents difficulties in interpretation. Cluster 3 showed a positive value for price, with negative values for enjoyment and convenience. Two of the rough clusters had positive values for price and two had negative values on other dimensions, convenience for Rough Cluster 2, and loyalty for Rough Cluster 6. It appears that the rough cluster analysis has segmented the price-conscious shopper into more specific subsegments of consumers, prepared to sacrifice different aspects of the shopping experience in exchange for price savings.

Cluster 6 had a high positive value for loyalty and negative values for price and personalizing. Three of the rough clusters also had positive values for loyalty. Rough Cluster 1 is a loyalty cluster, and all other dimensions are neutral. Rough Cluster 4 had a negative value for price and Rough Cluster 3 had a negative value for convenience. The rough clustering analysis, in addition to finding similar trade-offs for loyalty as the *k*-means approach, has found a major cluster defined by only the loyalty dimension. Cluster 7 had a positive value for convenience and a negative value for loyalty. Two rough clusters had positive value for convenience, one (Rough Cluster 5) with a positive value for loyalty and one (Rough Cluster 6) with a negative value for loyalty and positive for price. These rough clusters appear to be identifying different sub-segments of the data to those identified by the *k*-means approach.

Table 5: Cluster means and standard deviations for shopping orientation variables rough clustering - Interobject distance of 5

	Shopping Orientation				
	Enjoyment	Loyalty	Price	Convenience	Personalizing
	Mean (sd)	Mean (sd)	Mean (sd)	Mean (sd)	Mean (sd)
Cluster					
1	-0.22(1.70)	*1.48* (1.39)	-0.64 (1.77)	0.23 (1.77)	0.58 (1.67)
2	-0.01(1.98)	-0.69 (1.64)	*1.03* (1.58)	*-1.84* (1.19)	-0.33 (1.73)
3	-0.05(1.93)	*1.07* (1.43)	-0.71 (1.52)	*-1.13* (1.32)	0.54 (1.57)
4	0.21(1.69)	*0.93* (1.35)	*-0.91* (1.48)	0.38 (1.64)	-0.17 (1.52)
5	*1.26*(1.43)	*0.95* (1.46)	-0.06 (1.43)	0.93 (1.74)	0.07 (1.99)
6	-0.26(1.48)	*-1.28* (1.45)	1.10 (1.63)	*1.27* (1.41)	-0.68 (1.67)
7	*0.92*(1.51)	0.15 (1.82)	*1.60* (1.20)	0.84 (1.59)	*1.56* (1.41)
8	*-1.32*(1.47)	-0.01 (1.56)	-0.66 (1.80)	-0.39 (1.90)	*1.50* (1.23)
9	-0.86(1.50)	0.64 (1.54)	-0.82 (1.55)	*-1.57* (1.25)	0.53 (1.67)

Centroids equal to or greater than (+ or -) 0.90 are shown in italics

More complex combinations of dimension values were found in Clusters 4 and 5 of the solution obtained via *k* -means. Cluster 4 had a strongly positive value for the personalizing dimension, coupled with moderately high values for loyalty and price. The closest match was Rough Cluster 7, with moderately high values on the personalizing, price and enjoyment dimensions. Cluster 5 showed a positive value for enjoyment, coupled with positive values for price and convenience and a strongly negative value for personalizing. The closest match was Rough Cluster 5, with positive values for enjoyment, loyalty and convenience.

As expected, rough clustering and *k*-means clustering have resulted in different interpretations of the data set, but with some degree of overlap in these interpretations. The rough clustering solution is necessarily different because of the possibility of multiple cluster membership of objects. Rough clustering provides a more flexible solution to the clustering problem. We suggest that the data regularities found by the rough clustering technique should be conceptualized as extracting *concepts* from the data, rather than strictly delineated subgroupings (Pawlak, 1991). The concepts provide interpretations of different shopping orientations present in the data without the restriction of attempting to fit each object into only one subsegment. Such concepts can be an aid to marketers attempting to uncover potential new segments of consumers. As such, it is a promising technique deserving further investigation.

Extensions

In a paper relating to the use of models in marketing decision making, Lilien and Rangaswamy (2000, p. 234) suggested that “we should explore alternative approaches, like Artificial Intelligence (AI), that are beyond our traditional toolkits derived from statistics, operations research, and psychology.” Many of these approaches, such as neural networks and evolutionary algorithms, are already being regularly used in marketing research applications. This chapter has outlined the use and extension of another computational intelligence technique. This technique, rough sets, has been applied to a marketing problem, the clustering of a data set into homogeneous clusters. Rough clustering derives concepts from the data set as an aid to interpretation and is a valuable addition to the marketer’s attempt to identify subgroups of consumers. The technique could be used with other demographic and psychographic measures to search for possible niche markets.

The research reported here is at the beginning of a program to explore the applicability of rough sets theory to problems in marketing research. A number of extensions are planned. One future extension of the work is to produce more efficient methods for the discretization of intervals of real data (Komorowski et al., 1999) rather than the simple method that was used in this example. Hybrid rough-fuzzy techniques are one way of addressing this problem (Pal & Skowron, 1999). A second extension of this research would be to compare the rough clustering solutions with solutions obtained by the more established technique of clustering using fuzzy sets. A third extension relates to the algorithm itself. When an object pair is identified, the current algorithm assigns one or both objects to the other object's earliest cluster. If the lower approximations are not maximized, there is likely to be a loss of information. Assigning an object to other than the earliest cluster may result in "less rough" clusters, that is clusters that maximize lower approximations and retain more information about the data set. Such clusters will have maximum uniqueness, but still allow the possibility of multiple cluster memberships. Because of the variety of possible solutions that could be found if this change to the clustering algorithm is made, a suitable optimizing algorithm is needed. We suggest that the use of evolutionary algorithm techniques would be most suitable for this optimizing task. Using such techniques to develop better cluster solutions is the next major extension of our research program.

REFERENCES

Arabie, P., & Hubert, L. (1994). Cluster analysis in marketing research. In R. P. Bagozzi (Ed.), *Advanced methods of marketing research*, Cambridge, MA: Blackwell, 160-189.

Beane, T. P., & Ennis, D. M. (1987). Market segmentation: A review. *European Journal of Marketing, 21*, 20-42.

Bellenger, D. N., & Korgaonkar, P. K. (1980). Profiling the recreational shopper. *Journal of Retailing,* 56(3), 77-92.

Berry, M. J. A., & Linoff, G. (1997). *Data mining techniques: For marketing, sales, and customer support*. New York: Wiley.

Blozan, W., & Prabhaker, P. (1984). Notes on aggregation criteria in market segmentation. *Journal of Marketing Research, 21*, 332-335.

Brown, M. R. (1999). *Buying or browsing? An investigation of the relationship between shopping orientation, perceived risk, and intention to purchase products via the Internet*. Unpublished doctoral dissertation, Griffith University, Brisbane, Australia.

Brown, M. R., Pope, N. K. Ll., & Voges, K. E. (2001). *Buying or browsing? An analysis of the effects of shopping orientation, gender, product type, and prior purchase on online purchase intentions*. Manuscript in preparation, Griffith University, Brisbane, Australia.

Currim, I. S., & Schneider, L. G. (1991). A taxonomy of consumer purchase strategies in a promotion intensive environment. *Marketing Science, 10*, 91-110.

Darden, W. R., & Reynolds, F. D. (1971). Shopping orientations and product usage rates. *Journal of Marketing Research, 8* (November), 505-508.

Dholakia, R. R., Pedersen, B., & Hikmet, N. (1995). Married males and shopping: Are they sleeping partners? *International Journal of Retail Distribution Management, 23(3)*, 27-33.

Dickson, P. R., & Ginter, J. L. (1987). Market segmentation, product differentiation, and marketing strategy. *Journal of Marketing, 51*, 1-11.

Furse, D. H., Punj, G. N., & Stewart, D. W. (1984). A typology of individual search strategies among purchasers of new automobiles. *Journal of Consumer Research, 10*, 417-431.

Gehrt, K. C., Yale, L. J., & Lawson, D. A. (1996). The convenience of catalog shopping: Is there more to it than time? *Journal of Direct Marketing, 10(4)*, 19-28.

Grover, R., & Srinivasan, V. (1992). Evaluating the multiple effects of retail promotions on brand loyal and brand switching segments. *Journal of Marketing Research, 29*, 76-89.

Hair, J. E., Anderson, R. E., Tatham, R. L., & Black, W. C. (1998). *Multivariate data analysis* (5th ed.). London: Prentice-Hall International.

Hartigan, J. A. (1975). *Clustering algorithms*. New York: Wiley.

Hashemi, R. R., Pearce, B. A., Arani, R. B., Hinson, W. G., & Paule, M. G. (1997). A fusion of rough sets, modified rough sets, and genetic algorithms for hybrid diagnostic systems. In T. Y. Lin & N. Cercone (Eds.), *Rough sets and data mining: Analysis of imprecise data*, Boston: Kluwer, 149-175.

Hawes, J. M., & Lumpkin, J. R. (1984). Understanding the outshopper. *Journal of the Academy of Marketing Science*,12(4), 200-218.

Hruschka, H. (1986). Market definition and segmentation using fuzzy clustering methods. *International Journal of Research in Marketing, 3*, 117-134.

Hu, K., Lu, Y., Zhou, L., & Shi, C. (1999). Integrating classification and association rule mining: A concept lattice framework. In N. Zhong, A. Skowron, & S. Ohsuga (Eds.), *New directions in rough sets, data mining, and granular-soft computing*, Berlin: Springer, 443-447.

Komorowski, J., Pawlak, Z., Polkowski, L., & Skowron, A. (1999). Rough sets: A tutorial. In S. K. Pal & A. Skowron (Eds.), *Rough-fuzzy hybridization: A new trend in decision making*. Singapore: Springer.

Kowalczyk, W., & Slisser, F. (1997). Modelling customer retention with rough data models. In J. Komorowski & J. Zytkow (Eds.), *Principles of data mining and knowledge discovery*, Berlin: Springer, 7-13.

Lilien, G. L., & Rangaswamy, A. (2000). Modeled to bits: Decision models for the digital, networked economy. *International Journal of Research in Marketing, 17*, 227-235.

Lingras, P., & Davies, C. (1999). Rough genetic algorithms. In N. Zhong, A. Skowron, & S. Ohsuga (Eds.), *New directions in rough sets, data mining, and granular-soft computing*, Berlin: Springer, 38-46.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam, & J. Neyman (Eds.), *Proceedings of the fifth Berkeley symposium on mathematics, statistics and probability,* Volume 1, Berkeley, CA: University of California Press, 281-298.

Mahajan, V., & Jain, A. K. (1978). An approach to normative segmentation. *Journal of Marketing Research, 15*, 338-345.

Mazanec, J. (1992). Classifying tourists into market segments: A neural network approach. *Journal of Travel and Tourism, 1(1)*, 39-59.

Munakata, T. (1998). *Fundamentals of the new artificial intelligence: Beyond traditional paradigms*. New York: Springer-Verlag.

Oosthuizen, G. D. (1994). Rough sets and concept lattices. In Ziarko, W. P. (Ed.), *Rough sets, fuzzy sets and knowledge discovery*, London: Springer-Verlag, 24-31.

Pal, S. K., & Skowron A. (Eds.), (1999). *Rough-fuzzy hybridization: A new trend in decision making*. Singapore: Springer.

Pawlak, Z. (1981). Information systems—Theoretical foundations. *Information Systems, 6(3)*, 205-218.

Pawlak, Z. (1982). Rough sets. *International Journal of Information and Computer Sciences, 11(5)*, 341-356.

Pawlak, Z. (1984). Rough classification. *International Journal of Man-Machine Studies*, 20, 469-483.

Pawlak, Z. (1991). *Rough sets: Theoretical aspects of reasoning about data*. Boston: Kluwer.

Pawlak, Z. (1999). Decision rules, Bayes' rule and rough sets. In N. Zhong, A. Skowron, & S. Ohsuga (Eds.), *New directions in rough sets, data mining, and granular-soft computing*, Berlin: Springer, 1-9.

Pawlak, Z., Wong, S. K. M., & Ziarko, W. (1988). Rough sets: Probabilistic versus deterministic approach. *International Journal of Man-Machine Studies, 29*, 81-95.

Peppers, D., & Rogers, M. (1997). *The one to one future: Building relationships one customer at a time*. New York: Currency/Doubleday.

Punj, G., & Stewart, D. W. (1983). Cluster analysis in marketing research: Review and suggestions for application. *Journal of Marketing Research, 20*(May) 134-148.

Shamdasani, P. N., & Ong, G. Y. (1995). An exploratory study of in-home shoppers in a concentrated retail market. *Journal of Retailing and Consumer Services, 2 (1)*, 15-23.

Shim, S., & Mahoney, M. Y. (1992). The elderly mail-order catalog user of fashion products: A profile of the heavy purchaser. *Journal of Direct Marketing, 6(1)*, 49-58.

Smith, W. (1956). Product differentiation and market segmentation as alternative marketing strategies. *Journal of Marketing, 21*, 3-8.

Tat, P. K., & Schwepker, C. H. Jr. (1998). An empirical investigation of the relationships between rebate redemption motives: Understanding how price consciousness, time and affort, and satisfaction affect consumer behavior. *Journal of Marketing Theory and Practice, 6 (2)*, 61-71.

Tauber, E. M. (1972). Why do people shop? *Journal of Marketing, 36*(October), 46-49.

Van den Poel, D., & Piasta, Z. (1998). Purchase prediction in database marketing with the ProbRough system. In L. Polkowski & A. Skowron (Eds.) *Rough sets and current trends in computing*, Berlin: Springer, 593-600.

Venugopal, V., & Baets, W. (1994). Neural networks and statistical techniques in marketing research: A conceptual comparison. *Marketing Intelligence and Planning,* 12(7), 30-38.

Voges, K. E. (1997). Using evolutionary algorithm techniques for the analysis of data in marketing. *Cyber-Journal of Sport Marketing, 1(2)*, 66-82.

Voges, K. E. & Pope, N. K. Ll. (2000). An overview of data mining techniques from an adaptive systems perspective. In A. O'Cass (Ed.), *Visionary marketing for the twenty-first century: Facing the challenge—Proceedings of ANZMAC 2000*, Brisbane, Australia: Australian and New Zealand Marketing Academy, 1323-1329.

Wind, Y. (1978). Issues and advances in segmentation research. *Journal of Marketing Research, 15*, 317-337.

Yao, Y. Y., Wong, S. K. M., & Lin, T. Y. (1997). A review of rough set models. In T. Y. Lin & N. Cercone (Eds.), *Rough sets and data mining: Analysis for imprecise data*, Boston: Kluwer, 47-75.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8*, 338-353.

Chapter XIII

Heuristics in Medical Data Mining

Susan E. George
University of South Australia, Australia

This chapter presents a survey of medical data mining focusing upon the use of heuristic techniques. We observe that medical mining has some unique ethical issues because of the human arena in which the conclusions are outworked. The chapter proposes a forward looking responsibility for mining practitioners that includes evaluating and justifying data mining methods–a task especially salient when heuristic methods are used. We define heuristics broadly to include those methods that are applicable for large volumes of data, as well as those specifically directed to dealing with uncertainty, and those concentrated upon efficiency. We specifically consider characteristics of medical data, reviewing a range of mining applications and approaches. We conclude with some suggestions directed towards establishing a set of guidelines for heuristic methods in medicine.

INTRODUCTION

Deriving—or discovering—information from data has come to be known as data mining. It is a popular activity in many domains, from stock market prediction to healthcare. There are varied and diverse applications of the knowledge derived from mining as conclusions are utilised in various capacities. Knowledge derived from medical mining has been used to assist tasks as diverse as patient diagnosis and inventory stock control; it has formed the knowledge behind intelligent interfaces for patient record systems and been the tool of medical discovery. This chapter reviews some of the applications that have been made and identifies the need for a set of guidelines for heuristic methods in medicine.

Data mining presents many challenges since data is often present in huge volumes, distributed across many sources and highly complex in nature with many hundreds of variables and relationships among those variables varying in time, space or both often with a measure of uncertainty. Data mining is sometimes distinguished from statistics because of the vastness of the data sets with which data miners are concerned. Large data sets mean an increased algorithmic complexity and with that complexity comes the corresponding need to address issues in efficiency, hence the focus of this volume on a synergy of heuristics and data mining.

Artificial intelligence gives a very specific meaning to the term "heuristic." A heuristic is something that aids discovery of a solution. Heuristic methods are typically employed within the context of problem solving, where the solution to a problem must be found via a process of graph search (elements in the graph represent problem states, or operations that transform problem states). In search heuristic methods are able to guide exploration in an appropriate way and achieve faster solutions or more optimal solutions; occasionally the heuristic prevents a solution. Thus there are classic graph search algorithms such as the A* algorithm which is heuristic search (under the right conditions). However, we may broaden the definition of heuristics to include those techniques that are specifically relevant to dealing with large volumes of data or uncertainty within data. In doing so we can move away from traditional heuristic methods and encompass a wider range of techniques that may be regarded as "heuristic" because they "aid discovery," the crux of data mining.

In the remainder of this chapter we consider the nature of medical data mining and identify it as a unique arena for heuristic techniques. This is not just because the data is sensitive or highly private or that ethical concerns shadow every stage of mining from data collection to analytic procedures, from storage to appropriate data access; privacy, accuracy and security are issues in other domains (Walhstrom, Roddick, & Sarre, 2000). Rather, medical data mining is unique because of the implications for human life that may arise from the conclusions drawn. We propose that a set of guidelines for medical data mining is necessary, and that this is a natural consequence of forward-looking responsibility within this field. Forward-looking responsibility is accountable for high quality products and methods and requires appropriate evaluation of results and justification of conclusions. The proposed guidelines relate to the evaluation and justification of data mining results (so important when heuristic "aids to discovery" are utilised that "may" benefit a solution) and extend to both where and how the conclusions may be utilised and indeed where heuristic techniques are relevant in this field.

This chapter continues by providing a short review of some techniques that may be regarded as heuristic, including the classic A* search algorithm, tabu search, genetic approaches, fuzzy logic, rough sets, case-based reasoning and neural networks. We also examine the nature of medical data that present particular challenges for mining tasks and the diverse range of medical mining applications that have been made linking the mining application with the particular challenges of the data. In most instances we observe that the heuristics methods are not evaluated, justified or explained, but simply employed–and this does not demonstrate good forward looking responsibility. We conclude the chapter with the guideline suggestions in this important human arena in which conclusions are outworked.

HEURISTICS

"Heuristic" comes from the Greek word heuriskein, meaning "to discover," and refers to theories that describe how humans make discoveries. Plesk (1997) reports that heuristics are key to good higher-order thinking in human problem solving and creativity, whether we are trying to play chess or plan a business. Eight basic heuristics for human creativity are suggested. These heuristics of human discovery are translated into computer discovery, where they conventionally function as 'rules of thumb.' Artificial intelligence popularised the heuristic as something that captures, in a computational way, the knowledge which people use to solve everyday problems.

The A* state-space search algorithm (Hart, Nilsson, & Raphael, 1975) is sometimes known as heuristic search because it incorporates a way for domain-specific knowledge to be incorporated into a search process. The A* graph search algorithm guides the exploration of the search process using a heuristic evaluation function which assigns a numeric value to quantify the goodness of a node in terms of a problem state. Built into this function is domain-specific knowledge about how close a given node is to the goal node. This knowledge is only approximate and may or may not work in guiding the search to a solution or to the best solution (in optimisation terms we may reach a local rather than a global minimum/maximum).

More recently Tabu search (Glover, 1986) has been used on combinatorial optimization problems. It is regarded as "a meta-heuristic" superimposed on other heuristics. It forbids moves (hence, tabu) which take the solution to states in the solution space previously visited, but rather it forces new regions to be investigated, possibly accepting a new poor solution to avoid a path already investigated. The Tabu method was partly motivated by the observation that human behaviour appears to operate with a random element that leads to inconsistent behaviour given similar circumstances—but that this can often be fruitful! Forcing the search to a particular non-promising solution can ultimately lead to finding the desired solution, even though it may appear a random move in contrast to a move to the local optimum.

Increasingly, heuristics are used to refer to techniques that are inspired by nature, biology and physics. The genetic search algorithm (Holland, 1975) may be regarded as a heuristic technique. In this search approach, populations of solutions (represented as character strings) are evolved over a number of generations until the optimal one is located (the strings are evolved by a technique mimicking the crossover in the chromosomes of human reproduction). A "fitness" function determines the quality of a string and is based upon domain-specific knowledge. Additionally, the more recent Memetic Algorithm (Moscato, 1989) approach also demonstrates a population-based paradigm; combining local search heuristics with the crossover operations that produce new solution strings.

Brameier and Banzhauf (2001) provide the major example of a general heuristic technique being successfully utilised for mining medical data. They use a population-based paradigm but make an interesting modification to the genetic programming based on an inspiration from nature about how ‘introns’ (denoting DNA segments with information removed before proteins are synthesised) are used in generating new strings. They suggest that introns may help to reduce the number of destructive recombinations between chromosomes by protecting the advantageous building blocks from being destroyed by crossover. Complete protein segments are more frequently mixed than interrupted during evolution and this biological finding can be used as a heuristic approach within the genetic algorithm. Massive efficiency improvements in the algorithm are reported compared to neural networks as this general heuristic mechanism is applied to the genetic algorithm.

There are also a number of methods that are particularly useful when dealing with data that contains uncertainty. Fuzzy logic assesses data in terms of possibility and uncertainty. A fuzzy system helps to improve cognitive modelling of a problem because fuzzy logic encodes knowledge directly in a form that is very close to the way experts themselves think. It has the power to represent multiple cooperating, collaborating and even conflicting experts. However, using fuzzy logic requires a careful analysis of the value range of fuzzy variables. Rough set modeling can also aid uncertainty. Rough sets use a pair of standard sets, the lower approximation *S* and the upper approximation *‘S*, to represent uncertainty and vagueness. The difference between upper and lower approximation *‘S* - *S* is called a boundary region, or the area of uncertainty of a rough set. In contrast, case-based reasoning stores past examples and assigns decisions to new data by relating it to past examples. Here, each of the past examples refers to a case that can be defined as the description of a problem that has been successfully solved in the past, along with its solution. When a new problem is encountered, case-based reasoning recalls similar cases and adapts the solutions that worked in the past. Finally, artificial neural networks classify or cluster sets of data using a network structure of simple processing units connected by weighted links. The weights are established by a process of training from initially random values, and the network comes to represent a potentially complex statistical model of some aspect of the world. This range of methods (sometimes regarded as “heuristic”) has all been applied to medical data. We shall review some, especially in light of dealing with large data sets.

ISSUES IN MEDICAL DATA MINING

Responsibility in Medical Data Mining

Data security, accuracy and privacy are issues within many domains and ethical responsibility in these areas is not restricted to practitioners within the medical domain. Nor are data miners isolated from the even more general responsibilities faced by computing/engineering professionals, who produce software and systems upon which people come to depend. Unfortunately, responsibility for quality is frequently only highlighted when there is some breakdown or malfunction resulting in devastation to human life, but it need not be like this. Johnson and Nissenbaum (1995) distinguish "backward-looking" responsibility from "forward-looking" responsibility.

A "backward-looking" responsibility asks questions in the wake of a harmful event and seeks to discover who is to blame for the harm and who should be punished. It is often conducted in the context of discovering legal liability. The Therac-25 computer-controlled radiation treatment is a shocking example of a malfunction disaster that resulted in loss of life for people who were receiving computer-controlled radiation treatment for cancer. If medical data mining products are ever produced by "professionals" or are ever exploited "commercially," there may be the same serious legal consequences for their creators in the wake of harmful consequences from information produced. In contrast a "forward-looking" responsibility addresses the particular responsibilities in advance; it defines guidelines for creating quality products, measures the quality of the product, and defines the method of evaluation, the limitations and scope of operation in advance of harmful incidents. Any computer medical data mining "product" must have this forward-looking responsibility. In the context of software engineering, the computer field seeks to promote high quality software products, so too data miners should be seeking to guarantee high quality data mining techniques. Later we shall consider some guidelines for forward-looking responsibility in medical data mining. First we review the nature of medical data and data mining applications.

The Nature of Medical Data

Cios and Moore (2001) identify a number of unique features of medical data mining, including the use of imaging and need for visualisation techniques, the large amounts of the unstructured nature of free text within records, data ownership and the distributed nature of data, the legal implications for medical providers, the privacy and security concerns of patients requiring anonymous data used where possible together with the difficulty in making a mathematical characterisation of the domain. We concur with this view that medical data presents some particular mining challenges, particularly because of the human situation within which the data occurs and in which the derived information is used. Its human context is unique. It is perhaps one of the last areas of society to be "automated," with a relatively recent increase in the volume of electronic data and many paper-based clinical record systems in use. It shares difficulties of a young domain with lack of standardisation (for example, among coding schemes) and still some reluctance among healthcare providers to use computer technology. Nevertheless, the rapidly increasing volume of electronic medical data is perhaps one of the domain's current distinguishing characteristics, as one of the last components of society to be "automated." Large electronic medical data sets arise from the slow but sure penetration of technology into practice with the increasing use of electronic record keeping systems, digital medical imaging, remote video surgery, the database of genetic materials from the human genome research project and other ventures that produce data in volumes not previously encountered within the domain. With large datasets come all the issues in efficient storage, communication across networks, transfer to and from main memory, optimal processing and the complexities of dealing with distributed data dispersed over disparate sites.

Beyond the challenges of the sheer volume of medical data come the additional complexities that arise from highly structured data, with inter-related components, perhaps changing over time in multi-dimensional space. For example, the interrelations between patient symptom and diagnosis code, range over many hundreds of variables, often time-dependent and sparse—with any inherent structure potentially empty for a large portion of the population making it particularly difficult to detect structure. We face particular challenges with incremental data and time-series data, complexities in two or three-dimensional space with spatiotemporal pattern recognition increasingly the complexity. Medical data perhaps also faces some of the greatest uncertainty, inaccuracy and incompleteness that we may find. Natural patient variability contributes to "noise" that may be present in the data as inherent differences between people mean one miraculously recovers and another deteriorates. We must deal with missing records, erroneous values, nonstandard nomenclatures – with data in slightly different formats, using different coding, different symbols used with the same meaning and a variety of meanings for the same symbol. Medications may be referred to by a variety of names (generic or tradename), there may be duplication, error and redundancy. For example, patient age appearing in several places, medical terms misspelled and other noise present that calls for data preprocessing and cleaning. In short, medicine faces a rapidly increasing volume of complex, structured data without the accompanying rigour that might be expected from domains with established electronic data sets.

APPLICATIONS OF MEDICAL DATA MINING

Many ventures within medical data mining are better described as machine learning exercises where the main issues are, for example, discovering the complexity of relationships among data items or making predictions in light of uncertainty rather than dealing with the volume of data. With the exception of some medical imaging applications and mining of electronic medical records, the databases are small. There are even fewer instances where traditional heuristics from artificial intelligence (e.g., *A** search) have been applied to handling data volume, although there are many examples of case based reasoning, neural networks, fuzzy logic and other approaches that may be regarded as "heuristic" and "data mining." We review some data mining applications before considering those techniques and applications areas specifically relevant to data volume. We broadly categorise applications as clinical, administrative and research according to whether they are used (or potentially used) in a clinical context, are infrastructure related or exploratory in essence.

Clinical Applications of Data Mining

There are a wide variety of automated systems that have been designed for diagnosis–systems that detect a problem, classify it and monitor change. Brameier and Banzhauf (2001) describe the application of linear genetic programming to several diagnosis problems in medicine, including tests for cancer, diabetes, heart condition and thyroid. Their focus is upon an efficient algorithm that operates with a range of complex data sets. These diagnosis systems may be used in decision support or for a training tool. Decision support systems aid human judgement and decision making via statistical summaries, visualisations, and other forms of data presentation and analysis that highlight potentially important trends within the data. The National University of Singapore (Cao, Leong, Leong, & Seow, 1998) uses a data-driven approach based upon Bayesian probabilities to address time dependent conditional probabilities for follow-up of patients who have undergone cancer treatment. One of the main issues in this context was the temporal nature of the data. The Department of Optometry and Vision Sciences at Cardiff University of Wales, (Henson, Spenceley, & Bull, 1996) also demonstrated a decision-making aid in the context of managing the treatment of glaucoma patients, using an artificial neural network model derived from data mining a patient database of visual fields. In this particular work, many problems of medical data mining were addressed including spatiotemporal data, variation from psychophysical measurement, difficulties in diagnosis and lack of gold standard.

Carnegie Mellon University studied a medical database containing several hundred medical features of some 10,000 pregnant women over time. They applied data-mining techniques to this collection of historical data to derive rules that better predict the risk of emergency caesarian sections for future patients. One pattern identified in the data predicts that when three conditions are met, the patient's risk of an emergency caesarian section rises from a base rate of about seven percent to approximately 60 percent. Again the justification and explanation of the data mining conclusions have particular importance as there are definite clinical outcomes associated with the results of the mining.

Mani, Shankle, Dick, and Pazzani (1999) report on the use of machine learning as a data mining method to develop practice guidelines for managing dementia in Alzheimer's disease and determining its severity. They measure memory, orientation, judgment and problem solving, community affairs, home and hobbies and personal care in order to assign a global level of impairment based on the individual indicators. They hope to determine appropriate drug treatment and plan for future methods of patient care. Early identification of dementia can optimise quality of life and reduce costs. Dementia staging is not often used in practice because of its cost and the complexity of logic for scoring a patient. They used decision tree learners and rule inducers comparing both approaches with a Bayesian approach, concluding that a two-stage learning model improves classification accuracy over a single stage one. In applications such as this one, where the results of data mining have direct implications for patients, it is clearly vital that the methodologies and techniques that are applied are properly evaluated.

Administrative Applications of Data Mining

An interesting administrative application of data mining in a medical context comes in the area of interfaces for electronic medical records systems, which are appropriate for speedy, accurate, complete entry of clinical data. At the University of South Australia, George and Warren (2000) report on the use of a data mining model underlying an adaptive interface for clinical data entry. As records are entered a database is established from which predictive Bayesian models are derived from the diagnosis and treatment patterns. This model is used to predict the treatment from new diagnoses that are entered, producing intelligent anticipation. The predictive model is also potentially incremental and may be re-derived according to physician practice. One interesting issue in the data of this application is the discovery of appropriate mining methodologies that extract the valuable relationships between the data items. Various statistical and other techniques were reported, with the Bayesian being the most productive. This application addresses issues in incremental mining, temporal data and highly complex data with duplication, error and nonstandard nomenclatures. The more data available the better the models.

Epidemiology is concerned with the incidence of disease within a population and necessarily involves combining information from various sources. One obstacle to nationwide studies is the absence of standard nomenclatures. For example, there are several different coding schemes that may be used for recording problem diagnosis codes in general practice, while medications may be referred to by generic name or trade name. Combining data in epidemiology requires standardisation of coding and nomenclature. Such re-coding for standardisation in epidemiology may also be regarded as a form of "data compression," where the granularity of different diagnoses can be controlled (e.g., trauma to left and right hand may have separate codes which can be combined in some contexts), and "data cleaning," where erroneous values are removed, and "data preprocessing," where significant relationships are identified and enhanced.

Dawson (1998) reports on the software package from Orlando's MedAI called "Chronic Disease Identification," which directly uses data mining to analyse large volumes of data in order to predict life-threatening diseases. In particular it uses a neural network to examine a health provider's claims data, medical records information and other healthcare data to identify which individuals within a patient population will develop severe medical problems over the next three years. The system also predicts the costs of treatment. In the American insurance context this is valuable information for now it is possible to determine who is most at risk of given conditions with huge implications for insurance premiums. This application of mining must address some of the ethical issues associated with medical data and the need to respect individual confidentiality.

Bansal, Vadhavkar, and Gupta (2000) report on an important application of data mining to medical inventory control to forecast sales demand. "Medicorp" is a large retail distribution company that dispenses pharmaceutical drugs to customers in a number of states in the US. Medicorp is forced to have a large standing inventory of products ready to deliver on customer demand. The problem is how much quantity of each drug should be kept in the inventory at each store and warehouse. Significant financial cost is incurred if excess quantities are carried compared to customer demand, but too little supply leads to unsatisfied customers. Neural networks are used to optimize the inventory in a large medical distribution using traditional statistical techniques to evaluate the best neural network type. Serious experimentation was conducted with parameters for multi-layer perceptrons, time delay neural networks and recurrent neural networks. The first two were especially promising in forecasting sales demand, with the discovery that short interval predictions were particularly hard to make, although they gave more useful predictions. This application must cope with issues in epidemiology as temporal (seasonal and other) variations in demand of drugs are present.

Medical Research

Medical data mining is a natural method of performing medical research where new relationships and insights are discovered in human health. One such aid to medical research is the visualisation of data, producing digital images, perhaps 3-dimensional reconstruction of shapes (of organs, bones, tumors etc.) to aid investigation. For example, combining information from computerised tomography and Magnetic Resonance Imaging. Image fusion considers combining images of a single object taken by different sensors under different illumination conditions to create a single image that is better than any of the individual images. Stevan, Brooks, Downing, and Katz (1996) have used matching/fusion strategies are based on rule-based knowledge representation and tree search. The techniques are loosely within what might be defined as data mining in the sense that they extract information from the collection of points and produce a single fused image, importantly using search – with all the potential for conventional heuristics of artificial intelligence. The two-dimensional reference and target images are segmented into non-touching and non-overlapping regions (often using domain-specific information of knowledge about the shape of human organs). Region matches are then obtained using a tree search maximising a goodness of match. This application addresses a highly challenging area of 3-dimensional spatiotemporal change.

The application of artificial neural networks to medical imaging is also important. The University of Aberdeen addresses the problem of mammographic image analysis using neural nets together with conventional image analysis techniques to assist the automated recognition of pathology in mammograms (Undrill, Gupta, Henry, Downing and Cameron, 1996). The group also address the use of genetic algorithms for image analysis, applying this powerful general optimisation technique to a variety of problems in texture segmentation and shape analysis in two-dimensional and three-dimensional images (Delibassis and Undrill, 1996). Mining information from the data in these tasks must address many of the problems of finding patterns within large volumes of highly complex data.

Banerjee (1998) describes the use of data mining in medical discovery, reporting on a data mining tool which uncovered some important connections between diseases from mining medical literature. The data mining tool compared the article titles in various medical journals. Medical discoveries were made, such as the connection between estrogen and Alzheimer's disease and the relationship between migraine headaches and magnesium deficiency. Ngan, Wong, Lam, Leung, and Cheng (1999) have reported on medical discovery using data mining based upon evolutionary computation search for learning Bayesian networks and rules. They were able to discover new information regarding the classification and treatment of scoliosis as well as knowledge about the effect of age on fracture diagnoses and operations, and length of hospital stay.

LARGE SCALE MEDICAL DATA MINING

There are a few medical data mining applications that are specifically aimed at dealing with large volumes of data and scope within this context for drawing upon heuristics that specifically handle that data volume. Dealing with large-scale databases has its own unique difficulties that are simply not present in smaller scale machine learning projects. We continue to review some such mining applications and make suggestions for where heuristics will be of value. Applications have had to address issues such as distributed data, incremental mining and massive databases.

Distributed Data

Combining data from heterogeneous sites is frequently an issue with very large databases which cannot be combined into data within one location. Kargupta, Park, Hershberger and Johnson (1999) are interested in an epidemiological study that involves combining data from distributed sources. Their study investigates what it is that affects the incidence of disease in a population, focusing upon hepatitis and weather. They illustrate the collective data mining approach, emphasising the importance within medicine of merging data from heterogeneous sites. Their solution minimises data communication using decision tree learning and polynomial regression. As more and more hospitals and general practitioners, pharmacists and other healthcare related professions utilise electronic media, mining ventures are going to have to cope with mining across data sources. They will have to address issues as this study addresses, such as minimising data exchange, adopting suitable heuristic approaches.

Incremental Mining

One interesting on-going database mining project at Rutgers is the development of efficient algorithms for query-based rule induction, where users have tools to query, store, and manage rules generated from data. An important component of the research is a facility to remember past mining sessions producing an incremental approach. They are using heuristics for efficiently "re-mining" the same or similar data in the face of updates and modifications. In their trials a major insurance company was trying to explore anomalies in their medical claims database. The new data mining techniques aided the isolation of high cost claims and scenarios in each disease group that would lead to high cost claim. They also identified characteristics of people who were likely to drop out of their health plan and locations where there were higher dropout rates. In this general approach to mining, information from prior mining is utilised in new mining to prevent the need to compute relationships from scratch every time data is added to the database. This is naturally a general approach to mining large-scale changing databases that may be considered in a variety of fields.

Massive Databases

In June 2000 the completion of a rough draft of the human genome was announced. This landmark achievement promises to lead to a new era of molecular medicine, an era that will bring new ways to prevent, diagnose, treat and cure disease. The ultimate goal of genome research is to find all the genes in the genetic sequence and to develop tools for using this information in the study of human biology and medicine. The analysis firstly involves dividing the chromosomes into smaller fragments that can be propagated and characterized and then ordering (mapping) them to correspond to their respective locations on the chromosomes. After mapping is completed, the next step is to determine the base sequence of each of the ordered genetic fragments. The result of this multinational project is a sequence of approximately three billion base pairs of genetic material that make human beings what they are. New tools will be needed for analyzing the data from genome maps and sequences. Data mining technology is already pinpointing tiny differences in the genetic sequence and defects that cause disease. Identifying such anomalies is the first step toward finding cures. Musick, Fidelis, and Slezak (2000) describe an effort in large-scale data mining at Lawrence Livermore National Labs. They adapt current data mining techniques to the genome domain and lay the groundwork for a more extensive research and development effort in large-scale data mining. The results will be directly applicable to other large spatial, multi-dimensional data.

Data Compression

The American National Academy of Science identifies scalability of algorithms as a key research issue and suggests approaches that include selecting data subsets by random sampling and summarizing them. In particular they suggest the compression of images, working with reduced versions of the data, and the derivation of smaller data sets by performing regression. A four-terabyte database may be sampled down to 200 gigabytes, aggregated and the results filtered down to ten gigabytes. In the context of mining medical images it may be possible to utilise some of these heuristic approaches to data compression and reduction.

GUIDELINES FOR HEURISTIC APPLICATION IN MEDICAL DATA MINING

One of the biggest ethical issues in medical mining concerns what is done with the knowledge derived. There is tremendous potential for good in improving quality of human life, managing disease effectively, efficiently administering programs and preserving life, but the same knowledge can also be put to less constructive ends, or benefit only a few, or conform to the contemporary political agendas influenced by the philosophy of the age. For example, the mapping of the human genome presents unique possibilities and challenges for ethically employing the knowledge derived. There is the potential to predict genetic diseases in unborn children or the development of cancer in an elderly person, and with that knowledge comes a new responsibility, to appropriately employ it. The National Human Genome Research Institute has already issued statements concerning, for example, the need not to discriminate in issues of insurance and employment based on genetic information. There has never been quite the same potential and responsibility for using the knowledge we derive or discover.

Forward-looking responsibility requires ensuring the quality of automated techniques and knowing the limitations and scope of methods in advance of harmful consequences. It asks about the reliability of the derived knowledge and defines a way to evaluate products. This is none so pertinent as when heuristic methods are utilised to derive that knowledge. Whatever is ultimately done with the conclusions we know that heuristics do not guarantee “optimality” or even the “accuracy” or “validity” of the conclusion. Forward looking responsibility within medical data mining will address, among other things, how knowledge is evaluated, how conclusions are justified, and what are the scope of validity and the limitations of “products.”

Evaluation

Evaluation is a notoriously difficult component of many ventures. Evaluation in a medical context has its own complications. One of the best forms of evaluation for clinical data mining solutions is a clinical trial where the information derived from data mining solutions is used by experts; it is subsequently possible to compare and contrast the automated solutions with a range of expert opinion. However, such evaluation can be complicated by lack of consensus among medical experts, including differences of opinion in diagnosis statements, treatment preferences and other areas. Clinical trial is notoriously difficult to organise due to various constraints upon medical practitioners. Even with appropriate opportunity and incentive there are barriers to genuine evaluation that medical data mining projects would do well to anticipate and ensure that this vital evaluation stage is not omitted.

Clinical trial is not the only form of evaluation that is desirable for data mining solutions. Even before products are clinically trialed there are many other methods of ensuring the reliability and quality of knowledge derived. One such approach makes use of benchmark data sets where various techniques (heuristic and other) could be compared to assess quality and efficiency of solutions. Additionally some types of specialist data may be invaluable resources for data mining researchers. The American National Health and Nutrition Examination Survey X-ray Image Archive has made some effort to make substantial X-ray data available. The data set contains more than 17,000 X-ray cervical and lumbar spine digitised images, accessible online under controlled circumstances via Web tools.

Justification and Explanation

Explanation and justification of a conclusion and process that led to that conclusion are important components of automated reasoning systems that would also benefit automated discovery systems. Traditional rule-based expert systems have a “how and why” explanation facility, where it is possible to ask “how” a conclusion was obtained and follow back the chain of rule applications or ”why” a particular fact is being requested and explore the proposed chain of reasoning to see what the system is attempting to prove. Data mining would benefit from a similar facility. Indeed if we cannot verify the information discovered we may as well not utilise data mining since the conclusions derived would not be reliable.

Scope and Limitations

It is important to define the scope and limitations of systems. There may be constraints in many areas from clinical applicability to the efficiency of algorithms/ storage requirements of databases. We also need to define the time and space complexity of the algorithm especially whether the algorithm is scalable for large amounts of data, whether it is robust to errors, invalid input, and memory shortage, and to consider whether the output of the system is interpretable for humans or is itself input to another procedure. The scope of clinical applicability is important to define, since an algorithm may only be suitable for certain types of data or models built from certain populations. For example, an algorithm that can predict diagnosis/ treatment patterns among adult diseases may be unsuitable for prediction among children. Whether the system can reason with missing data, uncertainty or erroneous data influences its scope of operation as does whether it requires static or incremental data sets of patient data. Prior knowledge may be necessary for the algorithm or a certain amount of data may need to be available before mining, which again influences the scope of applicability of the method. Knowing the scope and limitations of a system is none the more poignant than when heuristics are used, since there is even more potential that a heuristic method is specific to a particular population and context.

CONCLUSION

This chapter has reviewed some methods and techniques of medical data mining relating them to heuristic techniques. We have identified the need for forward-looking responsibility as a unique concern of medical mining that is especially relevant given the application of heuristic techniques. We have identified some of the particular problems of medical data and related these to some of the applications of medical mining that have been made, and we have also identified where heuristic techniques are applied. We have found that many heuristic techniques are essentially small-scale machine learning methods rather than approaches related to the management of large volumes of medical data, although a few heuristic methods specifically related to volume of data have been reviewed. The chapter advocates a forward-looking responsibility, with the data mining practitioner aware of the guidelines for medical mining in regard to quality, its evaluation, the constraints on operation and the limitations of algorithms and even when mining should not be applied due to the inability to verify the information discovered, or other reasons. These guidelines would be very timely in a field with some unique ethical considerations, at a point when the amount of electronic medical data is rapidly increasing and heuristic techniques may be employed to "speed the quality" of solution or "improve a solution" in the information discovered in these vast datasets.

REFERENCES

Banerjee, K. (1998). Is datamining right for your library?, *Computers Libraries, 18(10)*, Nov/Dev 1998. Retrieved September 21, 2000 from http://www.infotoday.com/cilmag/nov98/story1.htm.

Bansal, K., Vadhavkar, S. & Gupta, A. (2000). *Neural Networks Based Data Mining Applications for Medical Inventory Problems*. Retrieved September 21, 2000 from http://scanner-group.mit.edu/htdocs/DATAMINING/Papers/paper.html.

Brameier, M. & Banzhaf, W. (2001). A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining, *IEEE Transactions on Evolutionary Computation, 5*, 17-26.

Cao, C., Leong, T. Y., Leong, A. P. K., & Seow, F. C. (1998). Dynamic decision analysis in medicine: A data driven approach. *International Journal of Medical Informatics, 51(1)*:13-28.

Cios, K. & Moore, G. W. (2001). *Medical Data Mining and Knolwedge Discovery: Overview of Key Issues*, in Medical Data Mining and Knowledge Discovery, Springer-Verlag, Germany.

Dawson, B. (1998). The digital Race, *Stay Free Magazine, Issue 14.* Retrieved September 21, 2000 from http://metalab.unc.edu/stayfree/14/datamining.html.

Delibassis, K. & Undrill, P.E. (1996). Genetic algorithm implementation of stack filter design for image restoration, *IEE Proc. Vision, Image & Signal Processing, 143(3)*, 177-183.

George, S.E. & Warren, JR. (2000). Statistical Modelling of General Practice Medicine for Computer Assisted Data Entry in Electronic Medical Record Systems, *International Journal of Medical Informatics. Vol. 57(2-3)*, 77-89.

Glover, F. (1986). Future paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research, 5:533-549*, 1986.

Hart, P. E., Nilsson, N. J. & Raphael B. (1975). A formal basis for the heuristic determination of minimum cost paths. I*EEE Transactions on Systems Science and Cybernetics 4*, 100-107.

Henson, D.B., Spenceley, S.E., & Bull, D.R. (1996). FieldNet: Package for the spatial classification of glaucomatous visual field defects, Perimetric Update 1995 / 96, *Proceedings of the XIIth International Perimetric Society Meeting*, Germany, pp 289 – 298.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems : an introductory analysis with applications to biology, control and artificial intelligence*, University of Michigan Press, Ann Arbor, MI, 1975.

Johnson, D. & Nissenbaum, H. (1995). *Computers, Ethics and Social Values*, Prentice-Hall, Englewood Cliffs, New Jersey.

Kargupta, H., Park, B., Hershberger, D., & Johnson, E.,(1999). Collective Data Mining: A New Perspective Toward Distributed Data Mining. *Advances in Distributed and Parallel Knowledge Discovery*, Eds: Hillol Kargupta and Philip Chan. MIT/AAAI Press.

Mani, S., Shankle, W., Dick, M. & Pazzani, M.(1999). Two-stage Machine Learning model for guideline development, *Artificial Intelligence in Medicine 16*, 51 – 71.

Moscato, P. (1989). *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*, P. Moscato, Caltech Concurrent Computation Program, C3P Report 826.

Musick, R., Fidelis, K. & Slezak, T. (2000). *Large-Scale Data Mining; Pilot Project in Human Genome*, Center for Applied Scientific Computing, Lawrence Livermore National Lab. Retrieved September 21, 2000 from http://www.isi.edu/nsf/papers/musick.htm.

Ngan, P. S., Wong, M. L., Lam, W., Leung, K. S. & Cheng. J. C. Y. (1999). Medical data mining using evolutionary computation, *Artificial Intelligence in Medicine 16, 73 – 96.*

Plesk, P. E. (1997). *Creativity, Innovation, and Quality.* Milwaukee, WI: ASQC Quality Press.

Stevan, M. V., Brooks, M. J., Downing, A. & Katz, H. E. (1996). *AI and Medical Imagery: Strategy and Evaluation of Inexact Relational Matching.* Retrieved September 21, 2000 from http://www.csu.edu.au/ci/vol2/vajdic/vajdic.html.

Undrill, P.E., Gupta, R., Henry, S., Downing M. & Cameron G.G. (1996). *Outlining suspicious lesions in mammography by texture focussing and boundary refinement*, Proc. 'SPIE Medical Imaging: Image Processing', 2710, pp.301-310, Newport Beach, CA, 1996.

Walhstrom, K., Roddick, J. F., & Sarre, R., (2000) *On the Ethics of Data Mining*, Research Report ACRC-00-003, January 2000, School of Computer and Information Science, University of South Australia, 2000.

Chapter XIV

Understanding Credit Card Users' Behaviour: A Data Mining Approach

A. de Carvalho
University of Guelph, Canada and
University of São Paulo, São Carlos,Brazil

A. P. Braga
Federal University of Minas Gerais, Pampulha, Brazil

S.O. Rezende and E. Martineli
University of São Paulo, São Carlos,Brazil

T. Ludermir
Federal University of Pernambuco, Brazil

In the last few years, a large number of companies are starting to realize the value of their databases. These databases, which usually cover transactions performed over several years, may lead to a better understanding of the customer's profile, thus supporting the offer of new products or services. The treatment of these large databases surpasses the human ability to understand and efficiently deal with these data, creating the need for a new generation of tools and techniques to perform automatic and intelligent analyses of large databases. The extraction of useful knowledge from large databases is named knowledge discovery. Knowledge discovery is a very demanding task and requires the use of sophisticated techniques. The recent advances in hardware and software make possible the development of new computing tools to support such *tasks. Knowledge discovery in databases comprises a sequence of stages. One of its main stages, the data mining process, provides efficient methods and tools to extract meaningful information from large databases. In this chapter, data mining methods are used to predict the behavior of credit card users. These methods are employed to extract meaningful knowledge from a credit card database using machine learning techniques. The performance of these techniques are compared by analyzing both their correct classification rates and the knowledge extracted in a linguistic representation (rule sets or decision trees). The use of a linguistic representation for expressing knowledge acquired by learning systems aims to improve the user understanding. Under this assumption, and to make sure that these systems will be accepted, several techniques have been developed by the artificial intelligence community, using both the symbolic and the connectionist approaches.*

INTRODUCTION

The widespread use of databases and the fast increase in volume of data in these databases are creating a problem and a new opportunity for a large number of companies. These companies are realizing the necessity of making an efficient use of their stored databases.

Moreover, as a result of the "information technology revolution," storage and processing capabilities have faced an explosive increase in the last decades. Today, commercial and scientific applications easily produce gigabytes or terabytes of data in a few hours. These data hold variable information, e.g., trends and patterns, which can be used to improve business decisions and to optimize performance.

However, today's databases contain so much data that it has become almost impossible to manually analyze them for valuable decision-making information. In many cases, hundreds of independent attributes need to be simultaneously considered in order to accurately model systems behavior. Nowadays, this need for automatic extraction of useful knowledge from large amounts of data is widely recognized.

Data mining (DM) techniques are employed to discover strategic information hidden in large databases. Before they are explored, these databases are cleaned. Next, a representative set of samples is selected. Machine learning techniques are then applied to these selected samples.

This chapter investigates three different machine learning techniques. Two of these techniques are the symbolic learning algorithms C4.5 (Quinlan, 1993) and CN2 (Clark & Boswell, 1991). The other technique is a multilayer perceptron neural network (Rumelhart & McClelland, 1986) with a knowledge extraction technique, the TREPAN algorithm (Craven, 1996).

Despite their successful use on a large number of tasks, artificial neural networks (ANNs) have been much criticized for not presenting a clear indication of their decision process. It is usually very difficult to figure out what an ANN has learned and to understand how it generates answers to queries posed to it. This feature is found in functional models that are strongly based on the discovery of input-output relationships. For a large number of applications, the users should be able to perceive how this relationship is built internally and why the network has chosen one solution instead of another.

In contrast to ANNs, symbolic learning methods exhibit a high degree of understanding to the external user. They are able to explain the path from the query to the answer in an easy-to-understand language. Some examples of symbolic learning methods are the C4.5 (Quinlan, 1993), CN2 (Clark & Boswell, 1991) and CART (Breiman, Friedman, Olshen & Stone, 1984) learning algorithms.

Systems that have both the ability of generalization found in ANNs and the external understanding of symbolic learning algorithms are now being sought. This class of systems includes the so-called hybrid intelligent systems, whose main idea is to put together the abilities of two or more artificial intelligence paradigms, such as symbolic learning algorithms and ANNs, to improve the overall performance obtained.

In this direction, a large effort has recently been made to develop techniques able to extract knowledge from ANNs. It is common sense that, in order to have high user acceptance and to improve their usefulness as learning tools, it is necessary that the explanation ability becomes an integral part of trained ANNs, allowing them to explain their structure in an easy language.

Algorithms used to extract knowledge from trained ANNs allow the description of this knowledge as a decision tree, a information path through the network or a set of rules generated in a language externally understandable. Classical examples of these algorithms are SUBSET (Towell & Shavlik, 1993), MofN (Towell & Shavlik, 1993), Rulex (Andrews & Geva, 1994) and TREPAN (Craven, 1996).

THE KNOWLEDGE EXTRACTION PROCESS

In building intelligent systems, the most important and difficult step is the extraction of patterns, which is a process that involves the extraction itself and the interpretation and expression of knowledge. It is a time-consuming task and, as a result, the most expensive step in building knowledge systems in general.

The term "knowledge" has been largely discussed from a philosophical point of view. It can be either explicitly acquired from specialists or implicitly extracted from data sets representing the application domain.

Most of the machine learning experiments use data sets derived from a database that have been previously cleaned and treated. Real-world applications need to deal with data coming from large databases with a large amount of noise, missing data and irrelevant features.

Knowledge discovery in databases (KDD) is the process of knowledge extraction from large volumes of data. Fayyad (Fayyad, Piatetsky-Shapiro, Amith &

Smyth, 1996; Fayyad, Piatetsky-Shapiro, & Amith, 1996; Fayyad, 1996) defines the KDD process as a nontrivial process of identification of valid, new, potentially useful and understandable patterns present in the data. This definition is easier to understand by taking each of its components individually:

- Data: The set of facts or cases in a data repository. As an example, consider the field values of a sale record in a database.
- Pattern: Refers to the set of models or information that represents some abstraction of a data subset in any descriptive language.
- Process: The KDD process involves several stages, such as data preparation, search for patterns and knowledge evaluation. This process is usually very complex;
- Valid: The patterns discovered must have a certainty degree. They must satisfy functions or thresholds that assure that the samples covered and the cases related to the patterns found are acceptable.
- New: A pattern found must provide new information about the data. The novelty degree is used to define how new a pattern is. It can be measured through either comparisons among the data changes (how much a defined value differs from the expected or predicted value) or absence of previous knowledge (the relationship between a new discovery and older discoveries).
- Useful: The patterns discovered must be incorporated in order to be used;
- Understandable: One of the main goals of KDD is to allow the user to understand the patterns through a descriptive language, thus allowing a deeper understanding of the data.
- Knowledge: Knowledge is defined according to the domain, considering usefulness, originality and understanding.

It is then important to stress the different meanings of the terms KDD and DM. While KDD denotes the whole process of knowledge discovery, DM is a component of this process. The data mining stage is used as the extraction of patterns or models from observed data. Knowledge discovery from databases can be understood as a process that contains, at least, the steps of application domain understanding, selection and preprocessing of data, data mining, knowledge evaluation and consolidation and use of the knowledge.

The KDD process begins with the understanding of the application domain, considering aspects such as the objectives of the application and the data sources. Next, a representative sample, selected according to statistical techniques, is removed from the database, preprocessed and submitted to the methods and tools of the DM stage with the objective of finding patterns/models (knowledge) in the data. This knowledge is then evaluated as to its quality and/or usefulness so that it can be used to support a decision-making process.

The search for patterns, as well as the understanding of the results from all the phases that compose the KDD process, can be accomplished with the aid of visualization techniques. These techniques facilitate the understanding, on the part of the users, of the knowledge extracted from the data, which can be accomplished by identifying structures, characteristics, tendencies, anomalies and relationships in the data (Rezende, Oliveira, Félix & Rocha, 1998).

Any realistic knowledge discovery process is not linear, but rather iterative and interactive. Any step may result in changes in earlier steps, thus producing a variety of feedback loops. One cannot simply expect an extraction of useful knowledge by submitting a group of data to a "black box."

Although at the core of the knowledge discovery process, the DM step usually takes only a small part (estimated at 15% to 25%) of the overall effort (Brachman & Anand, 1996).

Since the main focus of this chapter is on data mining, the next section concentrates on its main features.

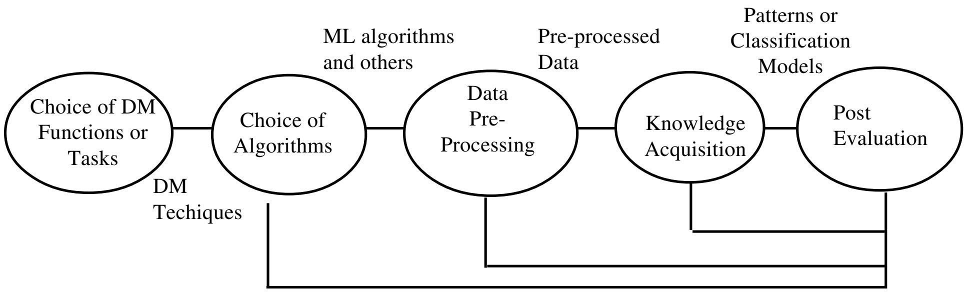
DATA MINING

By representing the main stage in the KDD process, the DM task can also be divided into several stages. A representative outline containing all these steps is illustrated in Figure 1.

As mentioned in the previous section, DM involves the following activities:

- Choice of DM functions or tasks: Analyze the task to define if the goal of the KDD is classification, grouping, regression, segmentation, etc.
- Choice of algorithms: Select the methods that will be employed in order to find patterns in the data. This may include the definition of which models and parameters would be more appropriate and the choice of a particular method from the criteria defined in the domain definition stage. The algorithms may be either supervised or non-supervised and they can follow the symbolic, statistical, instance-based, connectionist or genetic paradigms. Examples of algorithms used for data mining are decision trees (Quinlan, 1993), Bayesian networks (Russel & Norvig, 1995), clustering algorithms (Kohonen, 1984) and artificial neural networks (Rumelhart & McClelland, 1986). The algorithm choice is considered as an analytical process, since no one algorithm has an optimum performance in every application domain.
- Data preparation: Even after the data have been prepared and selected in the

Figure 1: Main stages in a data mining process



previous stages in the KDD process, they may still be too large to be used by the algorithms. A large data volume might even increase the complexity of the knowledge extracted representation, making the understanding harder. The data preparation involves the following steps:

1. Preprocess the data to the format specified by the algorithm to be used.
2. Reduce the number of samples/instances. This can be achieved either by simple reduction (random choice of samples) or by sampling techniques.
3. Reduce the number of features/attributes. This can be carried out either through suggestions from specialists or by using FSS (Feature Subset Selection—embedded filters or wrappers).
4. Features construction, which is the combination of one or more attributes in order to transform irrelevant attributes to more significant attributes. It is worth to mentioning that this manipulation may result in both the reduction and the increase in the number of features (becoming interesting in cases where there is a small number of features and a large number of examples).
5. Noise elimination, missing values treatment, etc.

- Knowledge Extraction: This activity involves a search for relevant patterns in one or more particular representations, like rules and decision trees, regressions, clusterings, among others. The success of this search depends on the two previous activities.

  In this stage, the data already prepared by the previous stage are used as input for the chosen algorithm. The extraction of patterns is then responsible to find patterns, models or classifications inside the data set.
- Pre-evaluation: Verifies the algorithm's accuracy, the model representation, the complexity of the rule sets or decision trees produced to represent knowledge, etc. Evaluates how difficult it is to understand the knowledge extracted. Checks what was learned, looking for new pieces of information and overfitting avoidance. Filters the information, eliminating useless and too obvious knowledge. For example, using a cattle database, produced rules like "only cows produce milk" should be ignored.

Figure 1 illustrates these steps. Once the patterns are discovered, they should be interpreted, analyzed and validated by the users of the KDD process.

Several different methods and techniques can be used to extract knowledge from data. The variety of methods and techniques currently used for extracting knowledge from databases comes from diverse disciplines such as statistics, machine learning, database, and visualization.

In order to evaluate the effectiveness of the knowledge extracted from different machine learning approaches, this work compares the knowledge extracted from ANNs using TREPAN to that extracted by two symbolic learning algorithms. The quality of a knowledge extraction technique can be evaluated by three different measures:

- Accuracy;
- Comprehensibility;
- New useful knowledge.

This chapter evaluates the performance of the three knowledge extraction techniques, measuring their classification accuracy and comprehensibility of the knowledge extracted.

The next section will briefly discuss different techniques that can be used for the knowledge extraction process.

KNOWLEDGE EXTRACTION TECHNIQUES

Several techniques have been successfully used to extract knowledge from databases. They range from numeric to symbolic, parametric to non-parametric and supervised to nonsupervised methods. In this section, three techniques are described and evaluated.

The first two techniques, the C4.5 (Quinlan, 1993) and CN2 (Clark & Boswell, 1991) symbolic learning algorithms, extract knowledge directly from the data set. The last technique, the TREPAN algorithm (Craven, 1996) extracts knowledge from a previously trained neural network. The CN2 algorithm induces *if...then* rules from a given data set. The C4.5 algorithm extracts decision trees, although it can also extract a set of ordered rules from the data set. Decision trees are also the knowledge representation produced by the TREPAN algorithm.

Knowledge Extraction by ANNs

There are several advantages in the extraction of knowledge from ANNs. A few of them are discussed in Andrews and Diederich (1995). They include:

- ***Validation***: System interface and external understanding are essential factors to allow the analysis and understanding of the hypothesis learned by the system.
- ***Integration between connectionist and symbolic systems***: Besides providing a precise and short symbolic description of the neural network knowledge, the knowledge extracted in the format *if...then* rules or decision trees facilitates the integration of ANNs with symbolic knowledge based systems.
- ***Investigation of data and scientific theory induction***: The data analysis carried out by the knowledge extraction process may find out new theories previously unknown. Without this process, theories discovered by ANNs would not become explicit.
- ***Explanation ability***: One of the main criticisms of ANNs is their inability to explain their decision process. The main goal of explanation procedures is to provide the user with a capability to explore the knowledge embedded in a system. Explanations must be clear and provide answers to the most relevant aspects of the knowledge acquired.

The knowledge acquired by ANN is represented by its bias and weight values in a numerical format. This representation is, in principle, not readily understandable by users. Knowledge extraction algorithms try to analyze, either directly or indirectly, this numerical representation in order to explain the ANN behavior. This analysis explores three different aspects of ANNs: topology, activation function and bias and weight values. According to Andrews and Diederich (1995), three different approaches can be considered in this analysis.

The first approach, known as *decompositional* or *local*, extracts knowledge individually from the nodes of a trained network. The rules for each unit take into account all the units in the previous layers. This approach produces rules with simpler intermediate terms. On the other hand, it requires the hidden units to be approximated by *threshold* units. Among the algorithms following this approach, one can mention the SUBSET (Towell & Shavlik, 1993), MofN (Towell & Shavlik, 1993) and Rulex (Andrews & Geva, 1994) algorithms.

However, these algorithms have restricted generalization and scalability capabilities. Most of them have limited applicability because they either require the use of a special training procedure (Craven & Shavlik, 1994; Setiono & Liu, 1995) or impose some restrictions on the network architecture or transfer function (Andrews & Geva, 1994; Tan, 1994; Fu, 1991).

The second approach involves the so-called *global methods*. These methods describe the behavior of the output units as a function of the input units. The most known methods using this approach are the VIA (Thrun, 1994) and the TREPAN (Craven, 1996) algorithms. The TREPAN algorithm is described in the next section.

The third approach, named combinational, is a combination of the two previous approaches and includes techniques which use the network architecture and weight values in order to complement a symbolic learning algorithm.

The TREPAN Algorithm

The TREPAN (TREes PArroting Networks) algorithm (Craven, 1996) represents the knowledge acquired by a trained ANN as a decision tree. It generates the decision tree by taking as input the trained ANN together with the input data used for its training. This algorithm does not take into consideration the internal network structure, which makes it general enough to be used together with most of the ANN's models.

TREPAN uses the same principle that is found in a few well-known algorithms for induction of decision trees, like C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984). These algorithms build a decision tree through recursive partitioning of a dataset. In particular, TREPAN builds a decision tree by using the *best-first* search technique, instead of the traditional *depth-first* search technique, used by most of the other algorithms.

When TREPAN is used, the class assigned to each example used in the construction of the decision tree is defined by an oracle or classifier, which is the trained ANN itself. Thus, when an example is presented to the oracle, it returns its class as defined by the ANN. The TREPAN algorithm aims to obtain a decision tree that represents in an accurate way the knowledge obtained by a trained ANN.

The expansion of the decision tree by TREPAN is controlled by a list that contains only leaf nodes. For each node in this list, a subset of the training samples, a set of complementary samples and a set of constraints are stored.

A new node inserted in this list is later removed either by expanding into other offspring leaf nodes or to finally become a leaf node which cannot be expanded any further. Once removed, a node cannot return to the list. Instead, the offsprings created as a result of the expansion are inserted in the list. The expansion of a node is carried out by using a special test that selects the nodes that must be divided and generates leaf offsprings.

The training subset contains examples that reach the related leaf node. The subset of complementary examples is composed by randomly generated examples and by some other examples that were not part of the ANN training set. These examples also can reach the related leaf node. These two subsets are used either in the division test of the internal nodes or to help define the class of the leaf nodes. The set of constraints stored in each node is formed from tests for conditional node division. These tests are performed on the sample attributes. In order to reach the related leaf node, the training examples and the complementary examples must satisfy this set of constraints.

According to the decision function used by the tree nodes, the TREPAN algorithm can generate three different formats of decision trees.

- **Original or m-of-n**: Extracts trees whose internal nodes present tests of the m-of-n type.
- **Disjunctive or 1-of-n**: Variation of the original method, uses disjunctive (OR) tests to generate the trees.
- **Simple test**: The decision tests have only one attribute, which is evaluated as true or false.

The next section presents the CN2 technique, which extracts a group of rules from a set of training examples.

The CN2 Algorithm

Systems that induce rules or knowledge from examples are very useful in the building of knowledge based systems (Clark & Boswell, 1991). The CN2 algorithm was designed to induce rules with the format *if...then*. It combines the ability to process noisy data, which is an attribute of ID3 based algorithms, with the ability to generate *if...then* rules and use search strategies employed by AQ based algorithms. When a set of rules is used, each rule is associated with a specific class.

CN2 is composed of two functions: the first function searches for the best rule to be extracted from the data. The second function controls the search for the rules by performing each selected rule several times. During the search process, CN2 evaluates the rules currently generated and defines the best one of them. One of the possible choices for a quality evaluation strategy is to evaluate the rule precision on the training set. Another possibility is the use of an entropy measure. However, this technique has a tendency of selecting rules that are too much biased towards particular cases, thus covering a low number of examples. As a result, the rules may present a poor generalization ability.

In order to avoid the generation of rules too specialized, CN2 uses a significance test. This test guarantees a fair distribution of the examples among the existing classes. Thus, rules that cover only part of the examples are discarded. But even discarding these rules that are below a specified *threshold*, the precision of the rules may be reduced because the remaining rules will probably be more general.

Another problem with the original CN2 is that the set of rules produced is ordered (Clark & Boswell, 1991). This set is also known as a decision list (Rivest, 1987), and the ordering makes its analysis more difficult. A solution to these problems is the use of the Laplace error estimate. CN2 with Laplacian error estimate was developed by Clark and Boswell (1991) and uses the original CN2 algorithm with entropy. This estimation is obtained by using the following equation (Clark & Boswell, 1991):

$$LaplaceAccuracy = (n_c + 1)/(n_tot + k) \qquad (1)$$

where k is the number of classes in the domain, n_c is the number of examples in the class ***c*** covered by the rule and n_tot is the total number of examples covered by the rule.

When the list of rules is generated, the class ***c*** predicted by a rule is the one with the largest number of examples covered by this rule. One advantage of this method is that it can generate a non-ordered list of rules that is both smaller and simpler than the corresponding ordered set.

A rule in an ordered list can not be evaluated separately, since the order must be followed. Thus, the complexity of the evaluation of an ordered set of rules increases with the number of rules in the set.

The C4.5 Algorithm

The C4.5 algorithm (Quinlan, 1993) uses a decision tree to represent the knowledge extracted from a given data set. When C4.5 is used in a classification problem, the classes are represented by leaf nodes of the tree produced. The other nodes correspond to the attributes used for the classification. The tree branches are labelled with either discrete attribute values or continuous attribute intervals. In order to improve its generalization, a decision tree can later be pruned. It must be noticed that the knowledge represented by a tree can also be described by production rules.

The generation of decision trees by C4.5 is based on the Hunt algorithm (Hunt, Marin & Stone, 1966), which given a test set T containing examples from the classes C_1, C_2, ..., C_k, chooses one among three possible decisions:

- T contains one or more examples, all of them belonging to the class C_j. The decision tree generated for T is a leaf that identifies the class C_j.
- T does not contain examples. The decision tree is also a leaf, but the class

associated to this leaf must be defined through information extracted from other sources. As an example, the leaf node can be associated to the class with the largest number of occurrences.

- T contains examples belonging to more than one class. In this case, T is divided into subsets, where each subset should have examples from the smallest possible number of classes. A criterion to divide T is chosen and the division of T through this criterion generates the subsets $T_1, T_2, ..., T_n$. The decision tree for the set T is then composed by a decision node and a set of child nodes. For each of the child nodes, one of these three possible decisions is recursively applied.

Usually, among several trees generated from the same dataset, the smaller the tree, the better its generalization. This occurs due to the smaller number of tests required and because the leaf nodes can be applied to a larger number of examples.

Each rule extracted from a decision tree corresponds to a path between the tree root node and a leaf node. Thus, the premises may be very large, i.e., they might have a large number of tests. If there is more than one path leading to a leaf node representing the same class, all these paths represent OR disjunctions.

It should be observed that all the tree paths are mutually exclusive. Only one path is satisfied for each query.

The next section describes the application domain used for the experiments presented in this chapter, credit risk analysis.

CREDIT RISK ANALYSIS

Credit can be defined as the delivery of a value in exchange for a promise that this value will be paid back in the future. However, there is always the risk of this promise not being consummated. Formal contracts are usually employed in order to guarantee to the creditor the right of receiving the debt. But even these contracts do not ensure that the debt will be paid to the creditor, since the debtor may not have the resources necessary to make the payment.

In order to reduce the risks of unpaid debts, a technical analysis is advisable before a credit is granted to a client. This analysis is named credit risk analysis.

Credit risk analysis has attracted a large deal of attention lately. Credit analysis is essentially a classification task that involves evaluation of the reliability and profitability of a credit application. In most of the cases of credit assessment, bank managers must deal with a large variety of information from a large variety of sources. Much of this information can be characterized as being:

- Incomplete;
- Ambiguous;
- Partially incorrect;
- Possibly relevant.

The traditional approach employed by bank managers largely depends on their previous experience and does not follow the procedures defined by their institutions.

Credit risk analysis is concerned with the evaluation of the profit and guaranty of a credit application. Credit applications can either be in benefit of companies or people. Examples of personal credit applications are student loan, personal loan, credit card concession and home mortgage. Examples of company credits are loans, stocks and bonds.

Nowadays, a growing number of shops ask for personal data from clients who want to finance their purchase. These data are used to consult a credit database that informs if the client has any history of unpaid debts. In sectors where the credit value is high, like home loans, a large amount of information may be required of the client. If the database does not return any negative information about the client, the credit is usually authorized.

The higher the value of the credit asked, the more rigorous is the credit risk analysis. Some large companies have whole departments dedicated to credit risk analysis. These departments carry out several activities before a credit is conceded, like:

- Elaborate files with clients' identification data;
- Elaborate financial analysis about the clients;
- Collect information regarding unpaid bills and clients' bankruptcy;
- Visit clients to create a more solid concept of the amount of credit they are able to assume;
- Consult other suppliers/creditors of the clients.

The next section discusses the different approaches that can be followed for credit risk evaluation.

Methods for Credit Risk Evaluation

The approaches usually employed for credit risk evaluation may be basically divided into two groups of methods:

- Subjective methods;
- Quantitative methods.

Subjective Methods

Subjective methods are those methods based on previous experience. The use of previous experience is the oldest and still the most used method for credit risk analysis. It is a valuable method, mainly if the involved issues are subjective. These methods take into account information like: the potential management of a company, the age of the production machinery, the client honesty, the analysis of the country's current economic situation, etc. However, this previous experience is not easily acquired and only time and participation in relevant situations can provide it.

Since loan officers rely on their own heuristics (result of their previous experience) when assessing credit applications, different officers may arrive to different decisions for the same application. Several companies follow the five Cs of credit when a credit request risk is evaluated (Ross, Westerfield, & Jaffe, 1993):

- Character;
- Capability;
- Conditions;
- Capital;
- Collateral.

Quantitative Methods

These methods, among them credit scoring, are used to classify quantitatively if a credit should be granted or not to a company (client). They belong to a more general class of methods and techniques named pattern recognition.

There are a large number of methods to deal with pattern recognition problems and, in principle, all of them can be used for credit risk analysis.

Subjective Versus Quantitative Methods

Models developed by quantitative methods present the following advantages over the use of previous experience:

- Consistency: Since different analysts have different judgement criteria, the use of previous experience can lead to different decisions for the same credit application. The use of quantitative models eliminates this subjective factor, allowing a financial institution to obtain a higher consistency in its credit decisions.
- Flexibility: Since they can be adapted to changes in the economic scenario, the quantitative models provide more flexibility. Quantitative models can be redesigned using a new data set representing the current economic situation. Subjective methods do not have this flexibility. These methods demand a long period of time before the analyst acquires the knowledge necessary to deal with the new situation. Besides, the previous experience may not be useful in the next period. The ideal situation would be the use of quantitative methods by credit analysts in addition to their previous experience. This combination could proportionate more efficient evaluation of credit requests, leading to better decisions.
- Speed: Quantitative methods reduce the time spent in credit risk evaluations, allowing the daily analysis of a larger amount of credit requests by financial institutions. Moreover, rather than analyzing factors that can be analysed by the model, the analysts can use their time to analyse the most relevant requests, to keep up to date with the current economic situation, to work in more complex requests, etc.
- Confidence: the analyst can use the decision generated by quantitative methods to support his/her final decision.

Additional advantages are the lack of racial, regional or ethnical discriminations. But one has to be aware that the quantitative methods also have their disadvantages:

- As time goes by, the economic situation changes and the models defined with outdated samples loose their efficiency. However, this problem may be overcome by redefining the models using more up-to-date samples.

- In many situations, the precision is not the only issue of interest. In general, quantitative methods are known as “black boxes”, not explaining how they arrive at a particular decision. Financial institutions may want an explanation of the decision process. When neural networks are used, knowledge extraction techniques can explain the decisions taken by the network. However, these techniques are still restricted to academic environments.
- Quantitative models depend on the geographical region where the samples were generated. Thus, the data should be collected from all the regions where a financial institution has clients.
- The financial institution may become too dependent on the company who developed the quantitative model. Techniques to automate the model design could be used to reduce this dependency.

The next section presents the experiments performed with a credit risk analysis dataset and discusses the results obtained.

EXPERIMENTS

This section presents a set of experiments performed using a real dataset of credit risk assessment and discusses the results obtained. The dataset used in the classification and knowledge extraction experiments is comprised of 5,635 samples selected from a very large dataset containing 100,000 credit card users.

Initially, these experiments evaluate the classification accuracy presented by ANNs, CN2, C4.5 and TREPAN. Afterwards, they compare the knowledge extracted by CN2, C4.5 and TREPAN. The ANNs used in this article are MLP networks (Rumelhart & McClelland, 1986), trained by the Rprop algorithm (Riedmiller & Braun, 1993).

In the comparison of the knowledge extracted, the first comparison juxtaposes the rules set generated by the CN2 algorithm with that produced by the C4.5 algorithm. Since C4.5 can generate knowledge using both the rules set and decision tree representations, the decision tree produced by the C4.5 algorithm for the same dataset that produced its rules set is compared to the three formats of decision trees generated by the TREPAN algorithm.

The experiments were carried out using a real dataset obtained from a Brazilian bank. The experiments were performed according to the principles recommended by the neural networks benchmark Proben1 (Prechelt, 1994). The Proben1 benchmark consists of a set of rules and guidelines on how to perform experiments with ANNs.

The credit card dataset consists of 5,635 examples divided in two classes: good payers and bad payers. 5,414 of the examples (96.07%) belong to the good payer class and 221 of the examples (3.93%) come from the bad payer class. This is an unbalanced dataset. Each example has 31 input attributes and 2 output attributes.

The input attributes have continuous and discrete values. A number of examples present missing values for some of the attributes. Input attributes include data like client credit history, credit purpose, years with employer, credit cards, savings account/bonds, personal status, sex, age, salary, type of residency, properties, education, address, etc.

Symbolic data, like sex, personal status and type of residency, were converted to numerical data. Two approaches were adopted to convert the values of each symbolic attribute. When the attribute values follow an order, the symbolic values were converted to real numbers obeying that order. Otherwise, the attribute values were converted to unary vectors, where the vector size was equal to the number of values for that particular attribute.

Once all the attributes were converted to numeric values, they were normalized and randomly divided into three subsets: training, validation and test subsets. Proben1 suggests the following partition: 50% of the samples for the training subset, 25% for the validation subset and 25% for the test subset. However, the large unbalance of the credit card dataset, with less than 4% of the samples belonging to one of the classes, could bias the training process toward the majority class if these proportions were used. In order to overcome this problem, the following numbers of samples were used:

- Bad payers: 177 samples for training, 22 for validation and 22 for test;
- Good payers: 177 samples for training, 541 for validation and 541 for test.

The samples for the good payers' class were randomly selected from the set of all samples belonging to this class. The experiments were conducted three times for each network; each time the dataset was shuffled in order to generate different partitions. In these partitions, the same proportion of samples present in each class for the training, validation (for the experiments with ANNs) and test subsets was preserved.

For the experiments with TREPAN, CN2 and C4.5, the validation subset was added to the training subset. Thus, the same number of samples was used for the training of the techniques investigated. For the experiments using ANN, part of the training data was used as a validation subset to support early stop.

Accuracy Evaluation

Table 1 shows the average correct recognition rates obtained from the different techniques investigated, along with their corresponding standard deviation.

Table 1 suggests that the CN2 algorithm with Laplacian method is more efficient to classify new patterns. The worst correct classification rates were achieved by the decision trees produced by C4.5. It is interesting to see that the rules set produced by the same method, C4.5, presents the second highest average of correct classification rates for the test subset. The performance achieved by TREPAN 1-of-n and m-of-n methods are superior to those achieved by TREPAN simple test method. It must also be noticed that the correct classification rates obtained during the training were close to 100% for the CN2 and ANN techniques.

Table 1: Correct classification rates

Method	Training	Test
ANN	96.7 0.4	98.0 1.4
TREPAN (1-of-n)	96.7 0.3	99.0 0.8
TREPAN (m-of-n)	96.3 0.8	99.1 0.7
TREPAN (simp.)	96.4 1.0	99.1 0.7
C4.5 (rules)	97.6 0.8	98.1 0.2
C4.5 (trees)	96.9 0.4	99.2 0.4
CN2	98.6 0.8	98.1 0.2

Since TREPAN's task is to learn the knowledge represented by an ANN model, it is desirable that TREPAN and this model have a high percentage of classification correlation. The results presented in Table 1 do not clearly show this relationship. TREPAN provides a measure, named fidelity, which defines these correlations.

Table 2 presents the correlation of the classification provided by TREPAN and the MLP network used.

As can be seen in Table 2, the classification obtained by TREPAN was very fidel to the classification achieved by the associated MLP network.

Comprehensibility Evaluation

This section presents the rule sets generated by the CN2 and the C4.5 algoritms and the decision trees produced by the C4.5 algorithm and by the TREPAN algorithm.

Rules Set Generated by CN2

For the partition with the average performance, the CN2 algorithm using the Laplacian method produced a set of 11 rules (including the default rule), which is a small number of rules. The classification of a new example is performed by testing each rule until a rule able to classify this example is found. The rules do not have a predefined order. The result of each rule does not depend on the results of previously tested rules. Figure 2 shows some rules produced by the CN2 algorithm.

In order to compare the knowledge extracted by CN2, C4.5 and TREPAN, the same partitions are used in the next sections by C4.5 and TREPAN to generate rule sets and decision trees.

Table 2: Fidelity for the test set

Method	Fidelity
TREPAN (1-of-n)	0.979
TREPAN (m-of-n)	0.972
TREPAN (simp.)	0.989

Figure 2: Some rules generated by CN2

```
if pt_cred > 57.50
    and hon_val < 1000.00
    and ath_sou < 29150.00
    then class = good payer [88 0]

if prop_lim > 2250.00
    and pt_cred < 73.50
    then class = good payer [69 0]

if prop_lim > 750.00
    and card_lim < 2250.00
    and sou_com < 4600.00
    then class = good payer [96 0]

(default) good payer [200 1999]
```

Rules Set and Decision Tree Generated by C4.5

Figure 3 illustrates some of the rules extracted by the C4.5 algorithm. The C4.5 algorithm extracted only 4 rules (including the default rule). Like the rules produced by the CN2 algorithm with the Laplacian method, the rules generated by C4.5 are unordered.

A decision tree was also extracted by C4.5. A branch of the decision tree produced by C4.5 is shown on Figure 4. The tree produced has 10 internal nodes and 14 leaves. Each internal node tests a group of hypotheses related to the customer situation, which can assume values like "single", "own house", etc. Each hypothesis evaluates the values of a subset of the applicant attributes. As all the paths in a tree are mutually exclusive, only one path is satisfied for each question.

Figure 3: Some rules extracted by C4.5

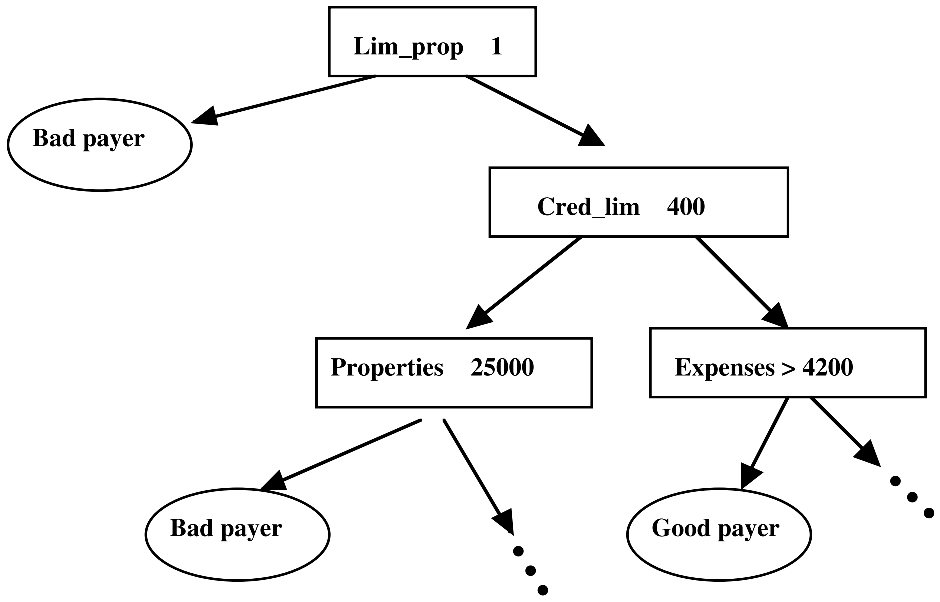
```
Rule 1:
    Lim_prop <= 1
    -> bad payer class [99.3%]

Rule 2:
    Lim_prop > 1
    -> good payer class [93.4%]

Rule 3:
    Properties > 25000
    Card_lim <= 400
    Res_tem <= 72
    -> bad payer class [70.7%]

default class: good payer
```

Decision Trees Created by TREPAN

The MLP networks used in this article have 38 input nodes, 20 nodes in the first hidden layer, 8 nodes in the second and an output layer with 2 nodes. For the output values 0 1, the class is positive (the customer is a good payer). For the values 1 0, the class is negative (the customer is a bad payer).

Figures 5, 6 and 7 show the decision trees obtained by using TREPAN with the simple test, the 1-of-n and m-of-n methods, respectively. For each tree, the internal nodes, shown as rectangles, represent the tests, and the leaves, shown as circles, represent the positive or negative outputs.

Figure 4: Branch of tree generated by C4.5



The assignment of an unknown example to one of the classes, using the m-of-n or 1-of-n methods, obeys a straightforward procedure. The test starts by applying the unknown input to the rule in the root node. According to the result (TRUE or FALSE), the process continues in the top node of either the left tree branch (TRUE) or the right tree branch (FALSE). This step is repeated for each square node until a leaf node is reached. The class associated to this leaf node is then the class of the unknown input. For the simple test method, each test node can produce two different values: good payer or bad payer.

As an example of the classification obtained by using the TREPAN m-of-n method, consider a credit card application represented by the premise: "*pt_cred* > 18.5". A customer with this profile is classified as a good payer by TREPAN, as shown in Figure 8. Thus, given a customer satisfying the rule shown, TREPAN would classify it as a good payer.

Figure 5: Branch of tree generated by TREPAN simple test method

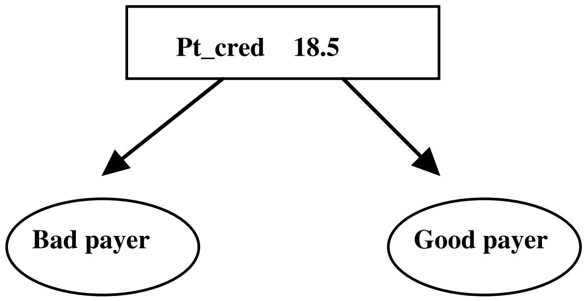


Figure 6: Branch of tree generated by TREPAN 1-of-n method

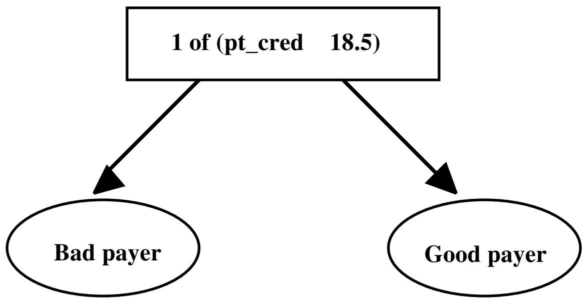
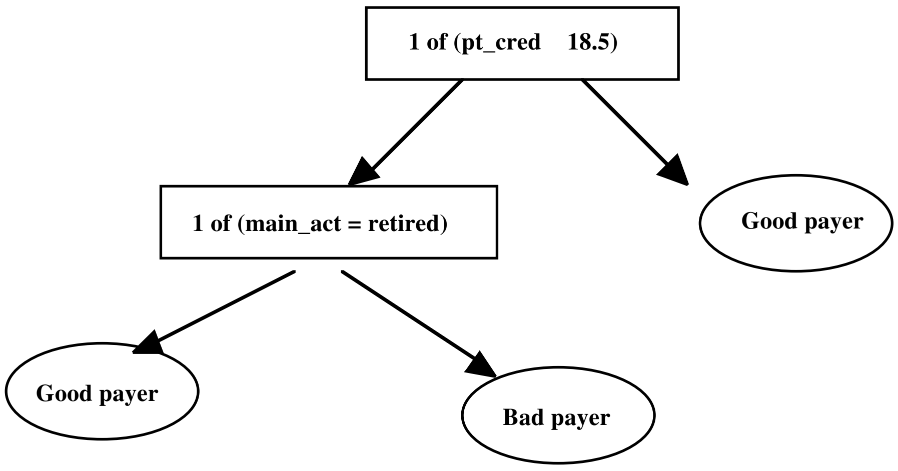


Figure 7: Branch of tree generated by TREPAN m-of-n method



The main features of the trees generated by each method are shown Table 3. According to this table, the shorter trees were produced by the TREPAN algorithms, mainly TREPAN 1-of-*n* and TREPAN simple. There is a large difference in the number of nodes, leaves and height of the trees produced by TREPAN and the tree produced by C4.5. Tall trees are more difficult to understand. Although a small number of nodes makes the tree understanding easier to the user, it must be pointed out that the test used by the tree generated by the TREPAN m-of-n method is more difficult to understand than the test used by the other methods. Moreover, trees usually provide a clearer view of the knowledge extracted than rules.

CONCLUSION

The application of data mining techniques to extract useful knowledge from large databases is gaining a great deal of attention. While there are fundamental

Figure 8: Classification obtained by TREPAN m-of-n

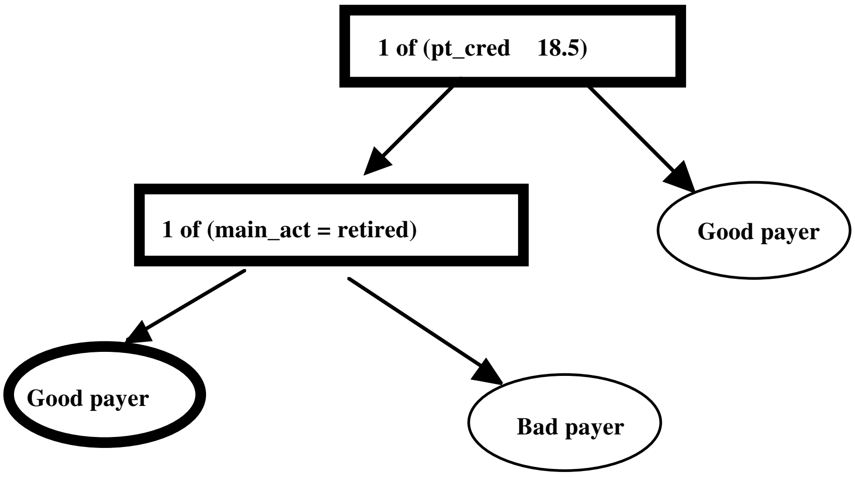


Table 3: Tree generated features

Method	Nodes	Leaves	Height
TREPAN (1-of-n)	1	2	2
TREPAN (m-of-n)	2	3	3
TREPAN (simple)	1	2	2
C4.5	10	14	7

problems that remain to be solved, there have also been numerous significant successful stories reported, and the results and gains are impressive (Simoudis & Emde, 1996). Although the current methods still rely on fairly simple approaches, the benefits from data mining technology have been convincingly demonstrated in the broad range of application domains.

Several techniques have been proposed to use in data mining tasks. This paper investigated the performance of three techniques, two of them symbolic and one connectionist, towards understanding credit card users behavior. Experiments were carried out using MLP neural networks (from which decision trees were extracted using the TREPAN algorithm), the CN2 algorithm, which generates a set of rules from a given dataset and the C4.5 algorithm, which can produce either a set of rules or a decision tree from a given dataset.

In order to evaluate the quality of the knowledge extracted by machine learning methods, three aspects should be considered: accuracy, comprehensibility and new useful knowledge. In this article, the knowledge extracted by these techniques was evaluated in terms of correct classification rates and the comprehensibility of the knowledge representation.

Regarding the classification accuracy, the highest correct classification rates were achieved by the rules generated by the CN2 algorithm (for the training subsets) and C4.5 trees (for the test subsets). The performance achieved by the trees extracted by the three TREPAN methods were similar to those achieved by the C4.5 decision trees and at least 1% superior, for the test subsets, to those obtained by the ANNs.

It must be pointed out that while C4.5 associates to each rule its correct classification rate, CN2 defines for each rule the number of examples covered by the rule. Although CN2 extracted less rules than C4.5, CN2 rules have, in general, a larger number of premises than C4.5. Usually, rules with more premises are less clear. While a set of CN2 rules indicates if a given client C_i will be a good payer, a TREPAN tree indicates if C_i can be a good payer, thus providing higher quality information.

Usually, it is easier to understand the knowledge represented by a decision tree than that described by a set of rules. There is a clear relationship between the complexity of the nodes test and the size of the decision tree. While the trees produced by TREPAN m-of-n and TREPAN 1-of-n methods allow only two options from each node, the trees produced by C4.5 and TREPAN simple test methods have three options for each test node. Besides being shorter, the number of nodes and leaves of the tree produced by the TREPAN simple test method were smaller than those of the tree produced by C4.5.

ACKNOWLEDGMENT

The authors would like to thank the support received from the research agencies CAPES, CNPq, FACEPE, FAPEMIG, FAPESP and FINEP.

REFERENCES

Andrews, R., & Diederich, J. (1995). A Survey and Critique of Techniques for Extracting Rules From Trained Artificial Neural Networks. Technical Report QUTNRC-95-01-02, *Neurocomputing Research Centre, Australia.*

Andrews, R., & Geva, G. (1994). Rule Extraction from a Constrained Error Backpropagation MLP. *Proc. 5th Australian Conference on Neural Networks.* Brisbane, Queensland, Australia, 9-12.

Brachman, R., & Anand, T. (1996). *The Process of Knowledge Discovery in Databases: A Human-centered Approach*, 37-57. AAAI Press/The MIT Press, Cambridge, USA.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, USA.

Clark, P., & Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. *Machine Learning - Proceedings of the Fifth European Conference EWSL-91*, Porto, Portugal, 151-163.

Craven, M. (1996). *Extracting Comprehensible Models from Trained Neural Networks.* PhD Thesis, Computer Science Department, Canrnegie Mellon University.

Craven, M.W., & Shavlik, J. (1994). Using Sampling and Queries to Extract Rules from Trained Neural Networks. *Proceedings of the 11 International Conference*, 152-169.

Fayyad, U. (1996). Data Mining and Knowledge Discovery: Making Sense Out of Data. *IEEE Expert, 11* (5), 20-25.

Fayyad, U., Piatetsky-Shapiro, G., Amith, S., & Smyth, P. (1996). *From Data Mining to Knowledge Discovery: An Overview*, 1-34. AAAI Press/The MIT Press, Cambridge, MA.

Fayyad, U., Piatetsky-Shapiro, G., & Amith, S. (1996). The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Communications of the ACM, 39* (11), 27-34.

Fu, L. (1991). Rule Learning by Searching on Adapted Nets. In *Proceeding of the Ninth National Conference on Artificial Intelligence - AAAI*, Anaheim, USA, 590-595.

Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in Induction*. Academic Press.

Kohonen, T. (1984). *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, Germany.

Prechelt, L. (1994). PROBEN1, A Set of Neural Network Benchmark Problems and Benchmarking Rules. *Technical Report 21, University of Karlsruhe*, Germany.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Rezende, S., Oliveira, R., Félix, L., & Rocha, C. (1998). *Visualization for Knowledge Discovery in Database*, 81-95. WIT Press Computational Mechanics Publication, Ashurst, UK.

Riedmiller, M., & Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *Proceedings of the IEEE International Conference on Neural Networks*, 586-591.

Rivest, R.L. (1987). Learning decision lists. *Machine Learning, 2*, 229-146.

Ross, S., Westerfield, R., & Jaffe, J. (1993). *Corporate Finance*. Richard D. Iwin, Inc., New York, USA.

Rumelhart, D.E., & McClelland, J.L. (1986). *Parallel Distributed Processing*. MIT Press, Cambridge, MA.

Russel, S. & Norvig, P. (1995). *Artificial Intelligence - A Modern Approach*. Prentice Hall, New Jersey.

Setiono, R., & Liu, H. (1995). Understanding Neural Networks via Rule Extraction. In *Proceeding of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 480-487.

Simoudis, E., & Emde, W. (1996). Reality Check for Data Mining. *IEEE Expert, 11* (5), 20-25.

Tan, A.H. (1994). Rule Learning and Extraction with Self-Organizing Neural Networks. In *Proceeding of the 1993 Connectionist Models Summer School*, University of Colorado, Boulder, CO, 192-199.

Thrun, S. B. (1994). Extracting Provably Correct Rules from Artificial Neural Networks. *Technical Report IAI-TR-93-5 Institut fur Informatik III Universitat Bonn*, Germany.

Towell, G.G., & Shavlik, J.W. (1993). Extracting Refined Rules from Knowledge-Based Neural Networks. *Machine Learning, 131*, 71-101.

Chapter XV

Heuristic Knowledge Discovery for Archaeological Data Using Genetic Algorithms and Rough Sets[1]

Alina Lazar
Wayne State University, USA

The goal of this research is to investigate and develop heuristic tools in order to extract meaningful knowledge from archeological large-scale data sets. Database queries help us to answer only simple questions. Intelligent search tools integrate heuristics with knowledge discovery tools and they use data to build models of the real world. We would like to investigate these tools and combine them within the genetic algorithm framework. Some methods, taken from the area of soft computing techniques, use rough sets for data reduction and the synthesis of decision algorithms. However, because the problems are NP-hard, using a heuristic approach by combining Boolean reasoning with genetic algorithms seems to be one of the best approaches in terms of efficiency and flexibility. We will test our tools on several large-scale archeological data sets generated from an intensive archaeological survey of the Valley of Oaxaca in Highland Mesoamerica.

INTRODUCTION

Archaeological Knowledge Discovery Problem

Anthropologists interested in ancient societies of Highland Mesoamerica, Valley of Oaxaca, have used intensive archaeological survey in order to study the state formation. Since these archaeological surveys were begun in the 1960s, the computer was an essential tool because of the large quantity of data resulting from the surveys. After the data was collected, it was placed on punch cards and the additional results were published in several books (Blanton, 1989; Blanton, Kowalewski, Feinman, & Appel, 1982; Kowalewski, Feinman, Finsten, Blanton, & Nicholas, 1989) along with extensive site maps. The reason behind this archaeological survey was to find answers to the following questions: What were the characteristics of Mesoamerican agricultural systems? What role did hydraulic agriculture play in prompting or facilitating the growth of large population centers? When was irrigation first introduced? What was the nature of these population centers? When and where did urbanism first arise? What decision making structures and adaptations were necessary to facilitate these changes? (Blanton et al., 1982).

Our goal for the proposed research is to integrate evolutionary learning tools into the knowledge discovery process and to apply them to the large-scale, archaeological spatial-temporal data produced by the surveys. This heuristic based approach used here will employ rough set concepts in order to represent the domain knowledge and the hypotheses.

While answers to the questions above can possibly be found by investigating the large-scale database resulting from the archaeological survey, this database contains over 6,000 regional sites and over 2,000 residential sites at the Monte Albán urban center. Each site is comprised of one or more components and can be occupied in one or more archaeological periods, spanning a period from approximately 9000 B.C. to 1500 A.D. Thus, the total spatial and temporal scope is so vast as to make manual interpretation a difficult if not impossible task. In addition, each temporal and spatial instance of a site component can be described in terms of several hundred variables of different types. We can clearly see a gap between data generation and data understanding here. Tools and techniques from artificial intelligence can be used to fill this gap and to aid in the extraction of emergent patterns hidden in the data, as is shown by Reynolds (1994, 1999).

Heuristics

Uninformed or blind search, which processes and evaluates all nodes of a search space in the worst case, is not realistic here because of time constraints that are closely related to the dimension of the data. Generally, the search space increases exponentially with problem size, thereby limiting the size of problems which can realistically be solved using exact techniques such as exhaustive search. An alternative solution is represented by heuristic techniques, which can provide much help in areas where classical search methods failed.

The word "heuristic" comes from Greek and means "to know", "to find", "to discover" or "to guide a investigation". Specifically, "Heuristics are techniques which seek good (near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is" (Russell & Norvig, 1995).

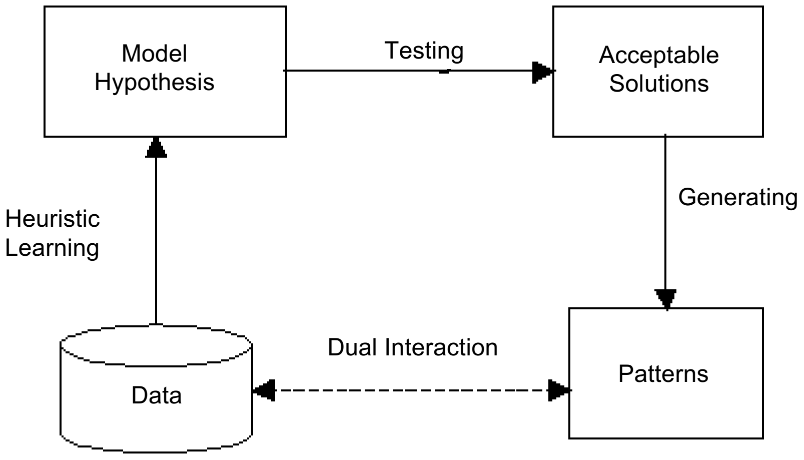
Heuristic refers to any technique that improves the average-case performance on a problem-solving task but does not necessarily improve the worst-case performance. Heuristic techniques search the problem space "intelligently" using knowledge of previously tried solutions to guide the search into fruitful areas of the search space. Often, search spaces are so large that only heuristic search can produce a solution in reasonable time. These techniques improve the efficiency of a search process, sometimes by sacrificing the completeness or the optimality of the solution. Heuristics are estimates of the distance remaining to the goal, estimates computed based on the domain knowledge.

Of special interest is the integration of heuristic search principles with the dynamic processes in which data becomes available in successive stages, or where data and inputs are subject to uncertainties, or with large-scale data sets. The integration is a vehicle to generate data-driven hypotheses. The process is shown in figure 1. Our goal is to generate hypotheses in terms of the archaeological data in order help anthropologists answer their questions.

The kind of knowledge produced and the heuristic search algorithm selected will reflect the nature of the data analysis task. In this chapter, the hypotheses will be represented as sets of decision rules and the extracted rules will be represented in terms of rough sets. Rough sets were selected because of the nature of our data sets as we will discuss later.

From a mathematical point of view the problems that we want to solve can be formulated in terms of the well-known minimal set cover problem, which is a

Figure 1: Knowledge discovery



combinatorial optimization problem.

Traditional methods for combinatorial optimization problems are not appropriate here for several reasons. These methods are NP-hard in the worst case and would be costly to use given the size of the data set. Also, since archaeological efforts in the valley are ongoing, new sites can be added to the database, which would require the traditional combinatorial approach to be restarted from scratch. The techniques used to solve these difficult optimization problems have slowly evolved from constructive methods, like uniformed search, to local search techniques and to population-based algorithms.

Genetic algorithms, as population-based algorithms, are good vehicles in which to build meta-level heuristics to guide the search more efficiently. That knowledge—here we will use rough sets concepts, or rules—can be employed to direct the evolutionary search. The rules can reflect spatial and temporal patterns that will guide the generation of new candidate search objects by the evolutionary engine. The spatial and temporal continuity of the data will facilitate this process. The organization of the chapter is as follows. The first section briefly describes evolutionary learning and queues a short literature review about related approaches. The next section presents the archeological data sets and the data-related problems. The third section is dedicated to the theory of rough sets. Next section presents a genetic algorithms approach for reduct computation using rough sets, and for decision system minimization. The next section describes the experiments, and the last section presents the summary, the discussions and the future work.

EVOLUTIONARY LEARNING UNDER UNCERTAINTY

Population-based heuristic methods are iterative solution techniques that handle a population of individuals which are evolving according to a given search strategy. At each iteration, periods of self-adaptation (mutations) alternate with periods of cooperation (crossover) and periods of competition (selection). The population-based heuristic search (Conrad, 1978) is dependent on the following components: the knowledge representation for the specific problem we want to solve and the search strategy or the evolution process. The adaptability of an individual represents its ability to survive in an uncertain environment (Conrad, 1975). Artificial intelligence researchers have explored different ways to represent uncertainty (Russell & Norvig,1995): belief networks, default reasoning, Dempster-Shafer theory, fuzzy set theory and rough set theory.

For the problems we want to solve, the learning task will require a representation that explicitly deals with uncertainty. The evolutionary learning methods that are employed must be able to work with such a representation. In this chapter we look first at basic ways to represent uncertainty in developing rules. And then we will investigate how that uncertain knowledge can be used to direct evolutionary search and learning.

Uncertainty

Uncertainty, as well as evolution, is a part of nature. When humans describe complex environments, they use linguistic descriptors of cognized real-world circumstances that are often not precise, but rather "fuzzy". The theory of fuzzy sets (Zadeh, 1965) provides an effective method of describing the behavior of a system which is too complex to be handled with the classical precise mathematical analysis. The theory of rough sets (Pawlak, 1991) emerged as another mathematical approach for dealing with uncertainty that arises from inexact, noisy or incomplete information. Fuzzy set theory assumes that the membership of the objects in some set is defined as a degree ranging over the interval [0,1]. Rough set theory focuses on the ambiguity caused by the limited distinction between objects in a given domain.

Rough Sets

A good approach to represent uncertainty is with rough sets. Rough sets are based on equivalence relations and set approximations, and the algorithms for computing rough set properties are combinatorial in nature. Wróblewski (1995) implemented a genetic algorithm for computing reducts, based on permutation code as well as a "greedy" algorithm. Another approach for building reducts is described by Vinterbo (1999) and it is based on the set cover problem, in particular on finding minimal hitting sets using a classical genetic algorithm. Finding a minimal set of decision rules or a satisfactory set is an NP-complete problem. Agotnes (1999) used genetic algorithms to build a optimal set of decision rules, where the fitness function was based on the quality of each rule. In conclusion, there are many hybrid methods that integrate evolutionary algorithms and other methods from soft computing, methods such as rough sets. Our goal is to find a suitable knowledge representation for our data and then to develop a cultural algorithm framework that combines that representation with the appropriate evolution method.

Evolutionary Computation

Evolution can be defined in one word, "adaptation" in an uncertain environment. Nature has a robust way of dealing with the adaptation of organisms to all kind of changes and to evolve successful organisms. According to the principles of natural selection, the organisms that have a good performance in a given environment survive and reproduce, whereas the others die off. After reproduction, a new generation of offspring, derived from the members of the previous generation, is formed. The selection of parents from these offspring is often based upon fitness. Changes in the environment will affect the population of organisms through the random mutations. Evolution is a dynamic two-step process of random variation and selection (Fogel, 1995). Using examples from natural systems and theories of adaptive behavior, researchers have been trying to build heuristic evolutionary learning systems.

Evolutionary algorithms are heuristic optimization methods inspired from natural evolution processes. Currently there are three basic population-only mechanisms that model evolution: genetic algorithms (Goldberg, 1989), evolutionary strategies and evolutionary programming. Each one of the methods models the evolution of a population of individuals at a different scale and applies selection and reproduction operators to find an individual that is fit with regard to the fitness function.

EXTRACTING PATTERNS FROM ARCHAEOLOGICAL DATA

Here we will work with the archeological data from Blanton et al. (1982), a survey of Monte Albán, named the Terrace Data Set. This volume describes and analyses the data collected during the first phase of the Valley of Oaxaca Settlement Project. The project consisted of a detailed mapping and surface collection of the region's major archaeological site, Monte Albán, part of the Oaxaca valley.

Large-Scale Data

One of the most important problems in data analysis relates to the dimensionality of the data because many data analysis techniques involve exhaustive search over the object space. They are very sensitive to the size of the data in terms of time complexity and it is hard to generate compact rules. The solution is to reduce the search space horizontally (in terms of records or objects) and vertically (in terms of fields or attributes or variables) and to use heuristics to guide the search through the large space of possible combinations of attribute values and classes. Our data set, for example, contains 2,073 records and 224 attributes.

Uncertainty in Data

Uncertainty in a data set can appear for different reasons. One reason is noise. Errors, which can occur during data collection or data entry, are referred as noise in the data. It is also possible that the data set can have missing attribute values. In this case, the objects containing missing attribute values can be discarded or the missing values can be replaced with the most common values. Another problem is that the available knowledge in many situations is incomplete and imprecise. This means that sometimes the attribute values for a set of objects are not sufficient and precise enough to differentiate between classes of objects. When we are talking about the Terrace Data Set, errors and noise may have occurred for many reasons. The ancient sites are damaged because of plowing, erosion, pot hunting and grazing. Also, human perception is subjective, and many people worked on the collection of the data. Some errors are possible due to the scanning process since much of the data was available from printed text only. Many different ways of representing and reasoning about uncertainty have been developed in artificial intelligence. These theories include: belief networks, non-monotonic logic, fuzzy sets along with fuzzy logic and rough sets. The well-known fuzzy set theory (Zadeh, 1965) characterizes a concept approximately using a set membership function with a range of values between 0 and 1. Another approach based on the rough set theory (Pawlak, 1991) provides a lower and upper approximation in terms of set belonging to a concept depending on how the relationship between two partitions of a finite universe is defined.

Fuzzy sets are good approaches for problems with multiple membership grade requirements, where judgment on set membership grades is possible and where the ability to deal with vague predicates is required. They are very good for real-valued data. On the other hand, rough sets with the three-valued simplicity, lower, upper, and boundary approximation sets, work well on discrete and categorical data. Rough sets can be useful even with missing data, changes of scale, problems where membership grades are hard to define, and problems requiring changes in the partition. Checking the attributes table for the Terrace Data Set we can see that out of the 92 attributes only 4 attributes are integer in type, with no real-valued data types. All the other attributes are of the categorical data type, nominal, ordinal, or binary. We want to find which sites where occupied in each period of time so we have to deal with data partitions. These facts suggest that rough sets methods are more appropriate here.

ROUGH SETS FORMALIZATION

Pawlak (1991) introduced rough set theory in the early 1980s as a tool for representing imprecise or uncertain information and for reasoning about it. Based on the notion of indiscernability, rough set theory deals with the approximation of sets, using equivalence relations. These approximations can form model hypotheses. Many different applications can be found in the literature, but here we focus on the applications to the classification problem since our goal will be to learn to classify occupied terraces and not occupied terraces in Monte Albán and their characteristics for each archaeological period in order to answer the questions posed in the first section.

Formal Definitions and Properties

An *information system* can be defined as a pair $S = (U, A)$, where U is a finite set of *objects* and A is a finite set of *attributes*. Each attribute $a \in A$ is a function that maps elements of U into a set V_a called the attribute domain, of attribute a, $a:U \rightarrow V_a$. Let $S = (U, A)$ be an information system and let $C,D \subset A$ be two subsets of attributes, called the condition and the decision attributes respectively. A *condition attribute* is an attribute that is thought to influence another attribute, the *decision attribute*. An information system with distinguished conditions and decision attributes it is called *decision table* and it is denoted by $T = (U, A, C, D)$. Because a table with more than one decision attribute can be easily transformed into a similar table with only one decision attribute, usually the set of decision attributes contains only one decision attribute, denoted D = {d}, and T = (U, C, {d}). The decision attribute d determines a partition in the object space U. The partition's elements are named decision classes.

With every $x \in U$ we associate a function d_x, which gives the value from V_c for a given attribute c, $d_x: C \rightarrow V_c$ (Pawlak, 1991), such that $d_x = c(x)$, for every $c \in C \cup \{d\}$. $d_x|C$ and $d_x|d$ are the restrictions of d_x to C, respectively d.

For every $x,y \in U$, we say that the object x is *discernible* if for every y, such that $x \neq y$, $d_x|C = d_y|C$ implies $d_x|d = d_y|d$, otherwise the object is *indiscernible*. A decision table is *consistent* if all of the objects are discernible; otherwise it is inconsistent.

Discernibility Matrix

A decision table T = (U,C,{d}), defines a matrix M_C^d called *the discernibility matrix modulo decision attribute d* (Øhrn, 1999). For $\forall\ x,y \in U$ with the condition that x and y are from different decision classes $d_x|d \neq d_y|d$.

$$M_C^d(x,y) = \{c \in C \mid c(x) \neq c(y) \quad \text{and} \quad d(x) \neq d(y)\} \qquad (1)$$

Since objects are divided into decision classes, we do not have to discern between objects that belong to the same class.

Indiscernibility Relations

A discernibility matrix M_C^d defines a binary relation $R_C^d \subseteq U^2$. The relation R_C^d is called an indiscernibility relation with respect to C and d, and reveals the pairs of objects from different classes that we cannot discern between. For $\forall\ x,y \in U$ under the condition that x and y are from different decision classes, $d_x|d \neq d_y|d$,

$$xR_C^d y \iff M_C^d(x,y) = \phi. \qquad (2)$$

The equivalence relation R_C^d, induces a partition over the universe U, meaning that the resultant equivalence classes are disjoint and the union equals the universe U.

Rough Sets

The idea behind rough sets is to approximate a set of interest in terms of other sets. With each subset $X \subseteq U$ and an equivalence relation R_C^d defined over U we can associate two subsets: $\underline{R_C^d}\, X = \{x \in U \mid \overline{R_C^d}\, (x) \subseteq X$ and $R_C^d\, X = \{x \in U \mid R_C^d\, (x) \cap X \neq \phi\}$ are called the lower and upper approximations of X respectively.

Reducts and the Core of Knowledge

One problem is whether some of the attributes in a decision system are redundant with respect to the object classifications. If an attribute set $B \subset C$ preserves the indiscernibility relation, R_C^d, then the attributes that form the set C-B are said to be dispensable. All minimal subsets, in terms of size, of attributes B that preserve the relation R_C^d are called reducts and we denoted the set by Red(T).

Now, we can define the full set of reducts in terms of the discernibility matrix. The set B, such that $B \subset C$ is the reduct of C if B is a minimal, with respect to inclusion, subset of C such that $B \cap M_C^d \neq \phi$ for any nonempty M_C^d ($M_C^d(x,y) \neq \phi$). Besides the full reducts defined above, we can define reducts that are relative to a particular object in the decision table. We call these reducts *object-related reducts.* If indiscernibility is relative to an object x, two other objects y and z are considered to be indiscernible in comparison with x. Reducts that are related to a particular object x are called *x*-relative reducts, Red(T,x), since they contain the minimum information needed to select that particular object from other objects in the decision table. There are several algorithms for computing reducts or reduct approximations. Some of these algorithms assume that any attributes subset of C can be an approximation to a reduct. The exhaustive reducer algorithm (Øhrn & Komorowski, 1997; Øhrn, Komorowski, Skowron, & Synak, 1998; Øhrn, Ohno-Machado, & Rowland, 1998; Øhrn 2000a, 2000b) computes all the reducts by brute force, by exhaustive search. The algorithm takes exponential time in terms of the number of objects, so it is not suitable for very large decision systems as it may be very time consuming. Another algorithm is the Johnson Reducer (Øhrn & Komorowski, 1997; Øhrn, Komorowski, Skowron & Synak, 1998; Øhrn, Ohno-Machado & Rowland, 1998; Øhrn 2000a, 2000b), which invokes a simple greedy algorithm to compute only a single reduct. Because of the NP-completness of the problem, heuristic methods can be more effective. Wróblewski (1995) proposed a variation of a genetic algorithm to search for reducts, either until the search space is exhausted or until a given maximum number of reducts has been found. Another heuristic approach was proposed by Vinterbo (1999). It is based on minimal hitting sets.

Reducts, Hitting Sets, Approximate Hitting Sets

Multisets are unordered collections of elements where an element can occur as a member more than once. A *hitting set* (Vinterbo & Øhrn, 1999; Vinterbo, 1999) for a given multiset, MS, of elements from P(C) is a set B, $B \subset C$, such that the intersection between B and every set in MS is nonempty.

$$HS(MS) = \{B \subseteq C \mid B \cap MS_i \neq \phi \text{ for all } MS_i \in MS\} \qquad (3)$$

The set $B \in HS(MS)$ is a minimal hitting set of MS, if B is no longer a hitting set, whenever any of its elements are removed. The set of minimal hitting sets is denoted by minHS(MS). An approximation to the hitting set is a set that covers enough elements of the multiset MS as denoted by a constant ε. The set of ε-approximate hitting sets of S is denoted by εHS(MS, e), where the parameter ε controls the degree of approximation,

$$\varepsilon HS(MS,\varepsilon) = \{B \subseteq C \mid \frac{|MS_i \in MS \text{ and } MS_i \cap B \neq \phi|}{|MS|} \geq \varepsilon\} \qquad (4)$$

The set $B \in \varepsilon SH(MS, \varepsilon)$ is a minimal ε-approximation hitting set if it is no longer an ε-approximation hitting set when any of its elements are removed.

The problem of computing the minimal hitting set, like the reducts computation, is an NP-hard problem. Again it is necessary to use heuristics in order to find reducts using hitting sets, but we still cannot guarantee the minimality of the reducts.

Decision System Construction

A *decision rule* is an assertion of the form "if p then s," denoted by p→ s, where p and s are logical formulas in the first order logic. For each object, certain values of the condition attributes determine the value of the decision attribute. We define a *decision system* as a finite collection or set of decision rules. In order to obtain a decision system with a minimum number of rules, superfluous decision rules associated with the same decision class can be eliminated without disturbing the decision-making process.

The problem of decision system construction is to induce a set of rule descriptors of decision classes from the input set of objects. These sets of descriptors, named decision systems, consist of a set of decision rules. We can classify the decision system as follows:

1. Decision systems with a minimum set of rules. They are focused on describing input objects using a minimum number of necessary rules.
2. Decision systems with an exhaustive set of rules. These decision systems contain all possible decision rules.
3. Decision systems with a satisfactory set of rules. This category represents sets of decision rules which satisfy, given a priori user's requirement for an acceptable decision system.

One strategy for finding a simple decision system with good classificatory capabilities is to first induce an exhaustive set of rules and then to prune away those rules that do not lower the decision system's performance significantly. An exhaustive decision system can be generated from the object-related reducts (Øhrn, 1999, 2000).

Pruning can be done by identifying and removing components of the decision system that only explain small parts of the data, thereby preserving general trends in the underlying data material. In order to find a minimal decision system we can use a simple greedy heuristic algorithm described by Lazar and Sethi (1999). This algorithm computes only one decision system. If more than one minimal decision system is required we can use a genetic algorithm, which solves the minimal cover set problem. Agotnes (1999) proposed two algorithms for generating satisfactory decision systems, a quality-based rule filtering algorithm and a genetic rule-filtering algorithm. Rule filtering operates on an existing exhaustive decision system, pruning it while retaining a high performance. Both of the above solutions make no assumptions about the minimal set cover condition. As a result, the decision system may not be minimal. We will propose a new solution based on the genetic algorithm which addresses the minimal set cover problem explicitly.

A FRAMEWORK FOR SOLVING THE PROBLEM USING GENETIC ALGORITHM

In this section we will present an existing genetic algorithms solution for the reduct computation problem and we will propose a new method for finding minimal and satisfactory decision systems using genetic algorithms.

Genetic Algorithm for Reduct Problem

For the reduct problem using a minimal hitting set, the population for the genetic algorithm is a set P of N individuals, each from the space P(C), where C is the condition attributes set. Each individual is encoded as a binary vector, where each bit indicates the presence of an attribute in the set.

For this population, the fitness function rewards individuals hitting more sets in the collection of sets corresponding to the discernibility function.

A possible fitness function proposed by Vinterbo (1999) is the following:

$$f(B)=\frac{|C|-|B|}{|C|}+\frac{|\{MS_i \ in \quad MS|MS_i \cap B \neq \phi\}|}{|MS|} \tag{5}$$

The first term rewards smaller-sized individuals, and the second is used to ensure that we reward sets that are hitting sets.

If we want to find the subsets B in C that are "good enough" hitting sets, i.e., have a fraction hit of at least ε, which tells us how many sets in the multiset, B has to hit. Vinterbo (1999) found that we can additionally control the importance of an individual hitting set by introducing the parameter α that defines a weighting between the two parts of the fitness function.

The resultant fitness function is a discrete, multimodal function. The algorithm used by Vinterbo (1999) is the traditional genetic algorithm implementation.

Genetic Algorithm for Minimal Decision Set of Rules Problem

The problem of producing a minimal decision set of rules can be formulated in set theoretical terms in two ways, because the minimal set cover problem and the minimal hitting set problem are complementary problems.

In the first approach, the minimal set cover problem, let us denote X as the set of objects and R as the set of rules derived from the set X. Then, we can define two functions: one, rX:R→ P(X), which associates with each rule $r_i \in R$ a set of elements $X_i \subseteq X$, and another one, xR:X→ P(R), which associates each element $x_i \in X$ with a set of rules $R_i \subseteq R$. Now, having the function rX defined, and a set X we want to find the subsets of rules $Rm \subseteq R$ that are of minimal cardinality and $\cup_{ri \, \varepsilon \, Rm} X_i = X$.

In the second approach, the complementary minimal hitting set problem, having the function xR defined we want to find subsets of rules $Rm \subseteq R$ that are of minimal cardinality and have a nonempty intersection with all the sets $R_i \subseteq R$. We can see that this is the problem of finding minimal hitting sets.

In both of the cases above each individual is a binary vector, with a length equal to the cardinality of the rule set R. Each bit is associated with one rule, and it tells us whether the rule is included in the set or not.

Following the ideas above we can define two new fitness functions for the two variants. For the minimal set cover problem we propose the following fitness function:

$$f(R') = (1-\alpha)\frac{|R|-|R'|}{|R|} + \alpha \min(\varepsilon, \frac{|\cup_{r_i \in R'} X_i|}{|X|}) \qquad (6)$$

For the minimal hitting set problem we propose the following fitness function:

$$f(R') = (1-\alpha)\frac{|R|-|R'|}{|R|} + \alpha \min(\varepsilon, \frac{|\{x_i \in X \mid R_i \cap R' \neq \phi\}|}{|X|}) \qquad (7)$$

Develop the Model

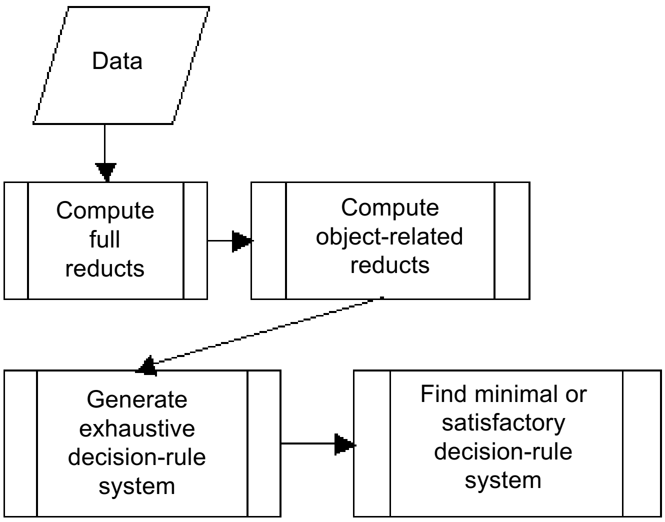
Knowledge discovery techniques are applied to the training data in order to generate a set of hypothesized relations. Following the rough set methodology, the full set of reducts is computed, a set of minimal reducts is chosen, and the data table is vertically pruned. Then the object-related reducts are computed and the exhaustive decision-rule system is generated.

At the end a pruning method for the decision rule set is applied in order to obtain a performant decision system, with a good balance between the number of rules and the accuracy of the classifications. The process is shown in Figure 2. The above procedure was followed exactly and the results are shown in next section.

EXPERIMENTS

The example problem we will investigate here is to discern the differences between the sites occupied in early I and late I in terms of the location and cultural

Figure 2: Model construction phases


Data
Compute full reducts
Compute object-related reducts
Generate exhaustive decision-rule system
Find minimal or satisfactory decision-rule system

attributes. The experts gave us a list with the diagnostic ceramic types for these two periods. The early I period, from 500 B.C. to 300 B.C., is named Ia and the late I period, from 300 B.C. to 200 B.C., combines Ib and Ic since a clear distinction between Ib and Ic cannot be made with the available data. Two binary variables were constructed for early I and late I for each site. A 0 means that the site was not present in the respective period, and 1 means present. Since we do not have sites which appear in period early I and do not appear in period late I we recode the two variables the following way: 0 means the site is present in both early I and late I, 1 means the site is present in late I, but not in early I, 2 means the site is present neither in early I nor in late I, and 3 was designated for site present in early I and not in late I. This will be our decision attribute. Then we selected only the sites with values 0 and 1 for the decision attribute. Out of the 92 attributes, we selected only 74 because some of them were not significant for our problem here such as: terrace number, card number, north grid coordinate, east grid coordinate, recorder, published information, comment written, reference in literature etc. Other attributes were removed because they were duplicates. For this experiment we have 875 sites, 306 sites with a 0 value, and 569 with a 1 value for the decision attribute. Only 2 sites have missing values for the attribute and we did not remove them. The genetic algorithm was used first to compute the full reducts.

Reduct (length=20):
east square #, area designation, elevation of the terrace in meters above the valley floor, silting, damage due to erosion and plowing, barranca or wash adjacent, vegetation, vegetation abundance, special resources, number of unnumbered tombs visible on the surface, estimated area of the terrace in square meters, prevailing wall orientations, presence of well-defined structure or structures less than 1 meter, plaster floor or floors visible on the surface, carved stone, ancient burning visible on the surface, miscellaneous obsidian, miscellaneous other chipped stone pieces (usually nondescript chunks of quartz, chert, or flint), number of whole or fragmentary manos, pottery density .

The reducts give us an idea about the most important variables related to the decision we want to make. They are related primarily with location; i.e., east square # and area designation, with elevation, vegetation, structures present in the site and tools. We picked the 20 attributes in the smallest reduct, and using the object-related reducts, we generated an exhaustive set of decision rules. It contains 16,574 rules. The rules were divided in two subsets, one for decision attribute value 0, and one for the decision attribute 1. After that we performed a quality-looping filter and kept approximately 20 rules for each class. Some of the rules are shown below:

- east square #(10) AND damage due to erosion and plowing(none) AND plaster floor or floors visible on the surface(present) AND pottery density(sparse to light) $\rightarrow$ diffiaibc(0)
- barranca or wash adjacent(absent) AND special resources(none) AND presence of well-defined structure or structures less than 1 meter(present) AND plaster floor or floors visible on the surface(present) $\rightarrow$ diffiaibc(0)

- east square #(10) AND area designation(2) AND damage due to erosion and plowing(none) AND plaster floor or floors visible on the surface(present) → diffiaibc(0)
- elevation of the terrace in meters above the valley floor, to t(375) AND barranca or wash adjacent(absent) AND vegetation(grass and brush) AND prevailing wall orientations(none) AND number of whole or fragmentary manos(1) → diffiaibc(0)

We can briefly give an interpretation for the rules produced for late I. In terms of location the sites expanded into new areas, for example area 15, and they went to east, east-square 4. The wall orientation doesn't have a precise direction anymore since more sites are present and there is little space left on the top hill, so they begin to colonize the sides. Pottery density becomes sparse, from light to sparse in the early I. Elevation is still high, 300, but less than in early I, 375. A lot of rules tell us that sites from late I have moderate to heavy damage due to erosion and plowing. We can see that better filtering is needed since we get multiple rules which fire the same set of objects and don't do a good set covering for the decision table. This is why we strongly need the genetic algorithm approach suggested in order to find a more satisfactory set of rules.

CONCLUSION

We proposed a methodology to develop and model hypotheses and to derive patterns from archaeological data using heuristics. The methodology is based on rough sets as a knowledge representation for uncertainty in data, combined with the evolutionary algorithms. A genetic algorithms, for reduct computation, have been already developed by Vinterbo (1999). A novel representation and performance function were proposed in this chapter for a genetic algorithm in order to solve the minimal and satisfactory decision rule systems problem. Since the search space is big large improvements over the genetic algorithm are needed in order to intensify the search in some regions of the search spaces. Further work will be done to integrate the cultural algorithm framework with the genetic algorithm. Comparisons in terms of time complexity and completeness of the solutions, between runs with the cultural algorithm (Reynolds, 1994) and genetic algorithm will be done. Following the archaeological expert's questions, we will run experiments for other periods of time or for other problems by changing the decision attribute. We will develop good decision rule systems for each of these questions. An integration of the decision rule systems with a GIS (geographical information system) is also possible.

REFERENCES

Ågotnes, T. (1999). *Filtering large propositional rule sets while retaining classifier performance*, Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Ågotnes, T., Komorowski, J. & Øhrn, A. (1999). Finding high performance subsets of induced rule sets: Extended summary. *Proceedings Seventh European Congress on Inteligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany* (H. J. Zimmermann and K. Lieven, eds.).

Blanton, R. E. (1989). *Monte Albán Settlement Patterns at the Ancient Zapotec Capital.* Academic Press, New York, USA.

Blanton, R. E., Kowalewski, S., Feinman G. & Appel, J. (1982). *Monte Albán's Hinterland, Part I, the Prehispanic Settlement Patterns of the Central and Southern Parts of the Valley of Oaxaca, Mexico*. The Regents of the Univerity of Michigan, The Museum of Anthropology.

Conrad, M. (1975). Analyzing ecosystem adaptability. *Mathematical Biosciences*, 27, 213-230.

Conrad, M. (1978). Evolution of adaptive landscape. *Theoretical Approaches to Complex Systems* (R. Heim and G. Palm, eds.), vol. 21 of *Springer Lecture Notes in Biomathematics,* Springer-Verlag, New York, USA,147-169.

Fogel, D.B. (1995). *Evolutionary Computation - Toward a New Philosophy of Machine Learning*. IEEE Press, New York, USA.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass, USA.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Komorowski, J. & Øhrn, A. A. (1999). Modelling prognostic power of cardiac tests using rough sets. *Artificial Intelligence in Medicine*,15(2), 167-191.

Kowalewski, S. A., Feinman, G. M., Finsten, L., Blanton, R. E. & Nicholas, L. M. (1989). *Monte Albán's Hinterland, Part II, Prehispanic Settlement Patterns in Tlacolula, Etla, and Ocotlan, the Valley of Oaxaca, Mexico*. Vol. 1. The Regents of the University of Michigan, The Museum of Anthropology.

Lazar, A. & Sethi I. K. (1999). Decision rule extraction from trained neural networks using rough sets. *Intelligent Engineering Systems Through Artificial Neural Networks* (C. H. Dagli, A. L. Buczak, & J. Ghosh, eds.), Vol. 9, ASME Press, New York, 493-498.

Øhrn, A. (1999). *Discernibility and Rough Sets in Medicine: Tools and Applications*. PhD thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, Trondheim, Norway.

Øhrn, A. (2000). *The Rosetta C++ Library: Overview of files and classes*. Tech. Rep., Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

Øhrn, A. (2000). *Rosetta technical reference manual.* Tech. Rep., Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

Øhrn, A. & Komorowski, J. (1997). Rosetta - a rough set toolkit for analysis of data. *Proceedings of Third International Joint Conference on Information Sciences, Durham, NC,* Vol. 3, 403-407.

Øhrn, A., Komorowski, J., Skowron, A. & Synak, P. (1998). The design and implementation of a knowledge discovery toolkit based on rough sets: The Rosetta System. *Rough Sets in Knowledge Discovery* (L. Polkovski & A. Skowron, eds.), Heidelberg, Germany: Physica Verlag.

Øhrn, A., Ohno-Machado, L. & Rowland T. (1998). Building manageable rough set classifiers *Proceedings of AMIA Annual Fall Symposium, Orlando, FL*, 543-547.

Pawlak, Z. (1991). *Rough Sets - Theoretical Aspects of Reasoning about Data.* Kluwer Academic Publishers, Dordrecht, The Netherlands.

Reynolds, R. G. (1984). A computational model of hierarchical decision systems, *Journal of Anthropological Archaeology*, 3, 159-189.

Reynolds, R. G. (1994). An introduction to cultural algorithms, *Proceedings of the Third Annual Conference on Evolutionary Programming, River Edge, NJ* (A. V. Sebald and L. J. Fogel, eds.), World Scientific Publishing, 131-139.

Reynolds, R. G. (1999). The impact of raiding on settlement patterns in the northern valley of Oaxaca: An approach using decision trees. *Dynamics in Human and Primate Societies* (T. Kohler and G. Gumerman, eds.), Oxford University Press.

Russell, S. J. & Norvig, P. (1995). *Artificial Intelligence a Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

Vinterbo, S. & Øhrn, A. (1999). Approximate minimal hitting sets and rule templates. *International Journal of Approximate Reasoning,* 25(2), 123-143.

Vinterbo, S. (1999). *Predictive Models in Medicine: Some Methods for Construction and Adaptation*. PhD thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, Trondheim, Norway.

Wróblewski, J. (1995). Finding minimal reducts using genetic algorithms. *Proceedings of Second International Joint Conference on Information Science*, Durham, NC, USA,186-189.

Zadeh, L. (1965), Fuzzy sets. *Information and Control*, 8, 338-353.

ENDNOTE

[1] This research was supported by NSF IIS Grant # 9907257

About the Authors

Hussein Abbass gained his PhD in Computer Science from the Queensland University of Technology, Brisbane, Australia. He also holds several degrees including Business, Operational Research, and Optimisation and Constraint Logic Programming, from Cairo University, Egypt, and a degree in Artificial Intelligence, from the University of Edinburgh, UK. He started his career as a Systems Administrator. In 1994, he was appointed associate lecturer at the Department of Computer Science, Institute of Statistical Studies and Research, Cairo University, Egypt. In 2000, he was appointed lecturer at the School of Computer Science, University of New South Wales, ADFA Campus, Australia. His research interests include swarm intelligence, evolutionary algorithms and heuristics where he develops approaches for the satisfiability problem, evolving artificial neural networks, and data mining. He has gained experience in applying artificial intelligence techniques to different areas including budget planning, finance, chemical engineering (heat exchanger networks), blood management, scheduling, and animal breeding and genetics.

Mongi A. Abidi is currently a Professor and Associate Department Head, Electrical and Computer Engineering, University of Tennessee, USA. He obtained his PhD in Electrical Engineering in 1987 from The University of Tennessee. Dr. Abidi published over 130 articles in proceedings, journal, transactions, and technical reports. His major area of interest: imaging processing, three-dimensional processing, data fusion, visualization, and robot vision and sensing. He has been involved with over $15 million of sponsored research project since 1986. He is a senior member of IEEE, Society of Optical Engineers, Pattern Recognition Society, and Association of Computing Machinery.

Ricardo Aler is a lecturer in the Department of Computer Science at Universidad Carlos III, Spain. His research includes automatic control knowledge learning, genetic programming, and machine learning. He has also participated in international projects like automatic machine translation and optimising industry processes. He holds a PhD in Computer Science from Universidad Politécnica de Madrid (Spain) and a MSc in Decision Support Systems for Industry from Sunderland University (UK). He graduated in Computer Science at Universidad Politécnica de Madrid.

Adil Bagirov has a Master's degree in Applied Mathematics from the Azerbaijan State University. He is now a PhD student at University of Ballarat, Australia. His principal research interests lies in optimization and its application to data mining and engineering. He has published in the areas of numerical methods of nonsmooth and global optimization with particular emphasis on applications to data classification.

Andrew Balemi joined Colmar Brunton Research (Auckland, New Zealand) in 1997, having obtained his PhD in Statistics. His job title is Head of Marketing Science and a major component of his job description is the development and implementation of new modelling and statistical techniques.

Daniel Borrajo is a university professor of computer science at Universidad Carlos III de Madrid (UC3M), Spain, since 1998, and previously an associate professor at UC3M and Universidad Politécnica de Madrid (UPM). He received his PhD in Computer Science from UPM in 1990, and a BS in Computer Science (1987) also from UPM. He has been Vice-Dean of the Computer Science degree at UC3M (1996-2000), and, currently, is the Head of the Computer Science Department at UC3M. His main research interest relates to the integration of different Artificial Intelligence techniques, specially concerning machine learning and planning, and their application to organizations and industry. He was head of the Symbolic Artificial Intelligence research lab at UC3M (1995-99), and supervised five PhD theses. He has published over 50 research papers in international refereed journals, conferences, and workshops, as well as three textbooks. He has been a member of the program committee of international conferences, and participated in the review process of R&D projects at the local, national, and European levels. He has been technical manager of a European project, and local and national projects, and participated as researcher in several European projects.

A.P. Braga obtained his first degree in Electrical Engineering and his MSc in Computer Science, both from the Federal University of Minas Gerais, Brazil. The topic of his PhD thesis, which was received from the University of London, England, was Storage Capacity of Boolean Neural Systems. After finishing his PhD, he returned to his position as an adjunct professor in the Department of Electronics Engineering at Federal University of Minas Gerais. His current research interests are neural networks learning, learning in distributed environments, hybrid neural systems and applications of NNs.

Mark R. Brown is a Lecturer in Marketing at Griffith University, Australia. Dr. Brown's primary field of expertise is in the arena of electronic commerce and he consults in this area. His other research interests include advertising strategy, customer relationship management, and sport marketing. He has extensive corporate experience and was most recently employed in the telecommunications and tourism industries. Dr. Brown teaches courses in consumer decision-making, services and relationship marketing, and marketing communications.

A. de Carvalho received his BSc degree in Computer Science and MSc degree in Informatics both at the Federal University of Pernambuco, Brazil. He received his PhD degree in Electronic Engineering from the University of Kent at Canterbury – England. Dr. Carvalho is an Associate Professor at the Departament of Computer Science of the Institute of Mathematics and Computing at the University of São Paulo, Brazil. In 2000, Prof. André de Carvalho was working in the University of Guelph, as Associate Professor. His main interests are artificial neural networks, genetic algorithms, hybrid intelligent systems and bioinformatics.

Peter J. Durrant has been researching the practical implications of the Gamma test for the non-linear data analyst and modeller. He has developed techniques for the analysis of complex and chaotic non-linear systems including embedding dimension search and feature selection routines. In conjunction with Steve Margetts, these routines were developed into a data analysis and modelling application called *winGamma*. Peter now works as a media analyst applying these techniques to a variety of business problems.

Dafydd Evans graduated from University College London in 1993 with a first class degree in Mathematics and obtained an MSc in Mathematics from the University of London the following year. Since 1997 he has been studying for a PhD in the Theory of Data Derived Modelling at the Department of Computer Science, Cardiff University, UK, and is concerned with the development of a rigorous mathematical foundation for the Gamma test. He lives in Penarth with his wife and two children.

Susan E. George received her BSc and PhD in computer science from the Computer Science Department, University of Reading, UK and MSc in Knowledge-Based Systems from the Department of Artificial Intelligence, Edinburgh University, UK. Dr George has undertaken post-doctoral research at both Department of Optometry and Vision Sciences, University of Wales, Cardiff, UK and more recently at the School of Computer and Information Science, University of South Australia where she is currently a senior lecturer. Her research interests include artificial neural networks and pattern recognition with applications in medicine, language learning, music and biometrics.

Craig Howard received his BSc degree (1995) in Computer Science from the University of East Anglia, UK, where he is registered for a PhD. He has undertaken a number of commercial knowledge discovery projects and has examined the problem of handling missing values in databases. Most recently, he has been a TCS associate within a data mining project between the University of East Anglia and the Lanner Group.

Antonia J. Jones graduated from the University of Reading with a first class degree in Mathematics and Physics, and then gained a PhD from Cambridge in Number Theory. During her career, she has published numerous papers across a multifaceted range of topics in number theory, genetic algorithms, neural networks, adaptive prediction, control and synchronisation of chaotic systems, and other works on non-linear modelling and noise estimation. She has worked in mathematics and computer science at many renowned academic institutions in the UK and USA, including the Institute for Advanced Studies, Princeton, NJ, the University of Colorado, Boulder, Royal Holloway College, London, Imperial College, London and a Computing Research Laboratory at NMSU, Las Cruces.

Alina Lazar is a graduate student at the Artificial Intelligence Laboratory at Wayne State University. Her PhD research deals with knowledge acquisition and data mining methods, combining cultural algorithms and rough sets. The goal of her research is to investigate and develop heuristic tools in order to extract meaningful knowledge large-scale data sets.

Agapito Ledezma is a graduate student of Computer Science at Universidad Carlos III de Madrid, Spain. His current research areas include ensemble of classifiers, data mining, knowledge discovery, intelligent agents, machine learning and evolutionary computing. He holds a BS in Computer Science and an Advanced Degree in Computer Science from Universidad Latinoamericana de Ciencia y Tecnología (Panamá), and he is currently working on his PhD in Computer Science at Universidad Carlos III de Madrid. He holds a grant from Agencia Española de Cooperación Internacional (Mutis Program).

T. Ludermir received her BSc degree in Computer Science and MSc degree in Informatics both from the Federal University of Pernambuco, Brazil. She received her PhD degree in Electronic Engineering from the Imperial College, University of London, England. Dr. Ludemir is an adjunct professor at the Informatics Center at Federal University of Pernambuco. Her main research interests are: artificial neural networks, computability and automata theory.

Paula Macrossan was awarded a Master's Degree in Applied Science (Mathematics) through the Queensland University of Technology, Australia in 2000. Her previous qualifications include a Bachelor of Agricultural Science (Animal Production) from the University of Queensland and a Graduate Diploma in Computing Science from the University of New England, Australia. She is currently enrolled in a PhD in Animal Breeding and Genetics at the University of New England.

Steve Margetts graduated from Imperial College in 1996, with a first class degree in Artificial Intelligence and Knowledge Engineering. He then moved to the Department of Computer Science, Cardiff University, UK, to pursue a PhD in adaptive evolutionary algorithms, during which he collaborated with Peter Durrant in the implementation of *winGamma.* He is currently working in the same department, where he continues to investigate evolutionary algorithms and nonlinear modelling techniques.

E. Martineli received his BSc and MSc degrees in Computer Science from the University of São Paulo, Brazil. His main research interests are: Artificial Neural Networks and Knowledge Extraction.

Kerrie Mengersen is a statistician whose research interests include statistical modeling, computational Bayesian methodology and novel methods of analysis. A key passion is the collaborative integration of this research with other disciplines such as information technology, genetics, environment and health. Currently a professor of statistics at the University of Newcastle in Australia and an active commercial statistical consultant, she has held national and international academic positions at four other universities and has co-authored over 50 professional papers on statistical methodology and applications.

Denny Meyer has worked with industry and universities in South Africa and New Zealand. She is currently a senior lecturer on the Auckland campus of Massey University, New Zealand, where her chief interests are multivariate analysis, data mining, time series analysis and the teaching of research methods papers.

Charles Newton is the Head of Computer Science, University of New South Wales (UNSW) at the Australian Defence Force Academy (ADFA) campus, Canberra. Prof. Newton is also the Deputy Rector (Education). He obtained his PhD in Nuclear Physics from the Australian National University, Canberra in 1975. He joined the School of Computer Science in 1987 as a Senior Lecturer in Operations Research. In May 1993, he was appointed Head of School and became Professor of Computer Science in November 1993. Prior to joining at ADFA, Prof. Newton spent nine years in the Analytical Studies Branch of the Department of Defence. In 1989-91, Prof. Newton was the National President of the Australian Society for Operations Research. His research interests encompass group decision support systems, simulation, wargaming, evolutionary computation, data mining and operations research applications. He has published extensively in national and international journals, books and conference proceedings.

Hyeyoung Park received her BS degree (*summa cum laude*) in computer science from Yonsei University, Seoul, Korea, in 1994, and MS and PhD degrees in computer science from the same university in 1996 and 2000, respectively. She was a researcher in the Research Institute of Industrial Technology, Yonsei University and a lecturer in the Graduate School of Yonsei University in 2000. Since September 2000, she has been a member of research staff at Brain Science Institute, RIKEN, Japan, and she is working on neural information processing systems based on mathematical theory. Her current research interests include computational learning theory, and neural network theory and their application to various fields such as pattern recognition, image processing, and data mining.

Nigel Pope is a Senior Lecturer in Marketing at Griffith University, Australia. Dr. Pope came to academia after a varied career in both the public and private sectors and in consulting. He has published extensively in Australia and overseas and has won the Literati award for research articles in 1999 and the Australian Award for Excellence in Academic Publishing in 2000. His research interests include marketing communications, the marketing of science and sport marketing.

S. O. Rezende was awarded her PhD degree from University of São Paulo, Brazil, in 1993. She is currently an Assistant Professor at the Departament of Computer Science of the Institute of Mathematics and Computing at University of São Paulo – USP, Brazil. Her main research interests are on Artificial Intelligence including data mining, machine learning, knowledge-based system and hybrid intelligent systems.

Alex Rubinov has a Master's degree in Pure Mathematics from Leningrad State University (Russia) and a PhD in Applied Mathematics from the Siberian Branch of the USSR Academy of Sciences. He is now a Professor of Mathematics at the University of Ballarat. His principal research interest lies in optimization, nonsmooth analysis, abstract convexity and data mining. He has published in the areas of abstract convexity (in particular, monotonic analysis) and numerical methods of nonsmooth and global optimization and its applications to data classification.

Guillermo Sánchez-Díaz obtained his MS degree in Computer Sciences from the Benemérita Universidad Autónoma de Puebla, Mexico in 1997, and obtained his Ph.D. degree also in Computer Sciences from the Centro de Investigación en Computación, I.P.N., Mexico, in 2001. His main research areas are: pattern recognition, data mining and image processing. He is member of the Pattern Recognition group Cuba-Mexico.

Ruhul Sarker received his PhD in 1991 from DalTech, Dalhousie University, Halifax, Canada, and is currently a Senior Lecturer in Operations Research at the School of Computer Science, University of New South Wales, ADFA Campus, Canberra, Australia. Before joining at UNSW in February 1998, Dr Sarker worked with Monash University, Victoria, and the Bangladesh University of Engineering and Technology, Dhaka. His main research interests are evolutionary optimization, data mining and applied operations research. He is currently involved with four edited books either as editor or co-editor, and has published more than 60 refereed papers in international journals and conference proceedings. He is also the editor of ASOR Bulletin, the national publication of the Australian Society for Operations Research.

Paul D. Scott is currently a Senior Lecturer in the Department of Computer Science at the University of Essex. After reading Physics at Oxford and taking an

MSc in Computer Science at London, he obtained a DPhil at the University of Sussex for research on neural net models of brain function. This was followed by several years working as a neurophysiologist at the Universities of Bristol and Newcastle. He then moved to the University of Michigan, Ann Arbor. Since then his primary research interest has been in various aspects of machine learning, with a secondary interest in applied artificial intelligence. Most of his current research activity is concerned with data mining but he also has active interests in bioinformatics and adaptive agents.

José Ruiz-Shulcloper was born in Cuba, received a BS in Mathematics from the Havana University, and a PhD from the Moscow State University; was Associate Professor at the Havana University; Invited Research Professor in several institutions in Latin America, Europe, and the United States. He has published more than 90 papers in journals and proceedings, several monographs, textbooks, and others publications; was the founder of the Logical Combinatorial Pattern Recognition Groups in several Cuban and Mexican institutions and is Senior Research and Leader of the Pattern Recognition Group at the Institute of Cybernetics, Mathematics and Physics, Cuba. He is also associate editor of the journals *Ciencias Matemáticas*, and *Computación y Sistemas*, received the National Mathematical Prize from the Cuban Mathematical and Computer Society, and the National Prize of Research in 1998 from the Cuban Academy of Sciences.

Kai Ming Ting received degrees in Electrical Engineering from the University of Technology Malaysia and in Computer Science from the University of Malaya and the University of Sydney, Australia. He was a practising electrical engineer from 1986 to 1992. He joined the Department of Computer Science at the University of Waikato between 1995 and 1998, as a post-doctoral fellow/ part-time lecturer. He was a Lecturer at Deakin University from 1998 to 2000. He is now a Senior Lecturer at Monash University. His research interests include machine learning, knowledge discovery and data mining, information retrieval and support vector machines.

Kevin Voges is a Senior Lecturer in Marketing at Griffith University, Australia. He has taught research methods courses in psychology, education, and business. He has consulting experience in education, organizational development, business planning, and market research. His research interests include the application of concepts and techniques from complexity science to marketing theory and practice, research methods based on adaptive systems theory, and sport marketing.

Chris Wearing holds a BSc in pure mathematics and a postgraduate diploma in statistics from Auckland University. Chris has worked within the market research industry since 1992. His job title at Colmar Brunton Research (Auckland) is Senior Market Modeller. He specialises in television polls, the development of interactive software products and providing strategic advice to clients.

John Yearwood has a Master's degree in Pure Mathematics from the University of Sydney and a PhD in Computer Science from RMIT University. He is now a Senior Lecturer in Computing at the University of Ballarat, Australia. His principal research interest lies in the extraction of knowledge from data to support decision-making. He has published in the areas of information retrieval, artificial intelligence, knowledge-based systems and data-mining with particular emphasis on applications to law and health.

Index