Design and Analysis of Scalable Rule Induction Systems

A thesis submitted to the University of Wales, Cardiff

for the degree of

Doctor of Philosophy

by

Ashraf A. Afify: B.Sc., M.Sc.

Intelligent Systems Research Laboratory

Manufacturing Engineering Centre

Systems Engineering Division

School of Engineering

University of Wales, Cardiff

United Kingdom

2004

ABSTRACT

Machine learning has been studied intensively during the past two decades. One motivation has been the desire to automate the process of knowledge acquisition during the construction of expert systems. The recent emergence of data mining as a major application for machine learning algorithms has led to the need for algorithms that can handle very large data sets. In real data mining applications, data sets with millions of training examples, thousands of attributes and hundreds of classes are common. Designing learning algorithms appropriate for such applications has thus become an important research problem.

A great deal of research in machine learning has focused on classification learning. Among the various machine learning approaches developed for classification, rule induction is of particular interest for data mining because it generates models in the form of IF-THEN rules which are more expressive and easier for humans to comprehend. One weakness with rule induction algorithms is that they often scale relatively poorly with large data sets, especially on noisy data. The work reported in this thesis aims to design and develop scalable rule induction algorithms that can process large data sets efficiently while building from them the best possible models.

There are two main approaches for rule induction, represented respectively by CN2 and the AQ family of algorithms. These approaches vary in the search strategy employed for examining the space of possible rules, each of which has its own advantages and disadvantages. The first part of this thesis introduces a new rule induction algorithm for learning classification rules, which broadly follows the approach of algorithms represented by CN2. The algorithm presents a new search method which employs several novel search-space pruning rules and rule-evaluation techniques. This results in a highly efficient algorithm with improved induction performance.

Real-world data do not only contain nominal attributes but also continuous attributes. The ability to handle continuously valued data is thus crucial to the success of any general purpose learning algorithm. Most current discretisation approaches are developed as pre-processes for learning algorithms. The second part of this thesis proposes a new approach which discretises continuous-valued attributes during the learning process. Incorporating discretisation into the learning process has the advantage of taking into account the bias inherent in the learning system as well as the interactions between the different attributes. This in turn leads to improved performance.

Overfitting the training data is a major problem in machine learning, particularly when noise is present. Overfitting increases learning time and reduces both the accuracy and the comprehensibility of the generated rules, making learning from large data sets more difficult. Pruning is a technique widely used for addressing such problems and consequently forms an essential component of practical learning algorithms. The third part of this thesis presents three new pruning techniques for rule induction based on the Minimum Description Length (MDL) principle. The result is an effective learning algorithm that not only produces an accurate and compact rule set, but also significantly accelerates the learning process.

RULES-3 Plus is a simple rule induction algorithm developed at the author's laboratory which follows a similar approach to the AQ family of algorithms. Despite having been successfully applied to many learning problems, it has some drawbacks which adversely affect its performance. The fourth part of this thesis reports on an attempt to overcome these drawbacks by utilising the ideas presented in the first three parts of the thesis. A new version of RULES-3 Plus is reported that is a general and efficient algorithm with a wide range of potential applications.

In The Name of Allah,

The Most Gracious, The Most Merciful

ACKNOWLEDGEMENTS

First of all I thank Allah (My Lord) the all high, the all great who made it possible for me to complete this work.

I wish to express my sincere thanks to the University of Wales Cardiff, especially the Intelligent Systems Laboratory, Systems Engineering Division, Cardiff School of Engineering for the use of the facilities to pursue this research.

I am privileged to have Professor D. T. Pham as my supervisor. The high standard of his research has always been an inspiration and a goal to me. I am deeply grateful to him for his consistent encouragement, invaluable guidance and strong support during the course of this study. His thoughtful advice and constant support extended to me will always be remembered.

Grateful acknowledgement of my funding and support must be made to my home country Egypt and the Egyptian Ministry of Higher Education. Also, my sincere thanks go to their representative in UK, the Egyptian Educational and Cultural Bureau in London and all of its members for their advice, encouragement and support.

Thanks are also due to all the members of staff of the Department of Systems Engineering, Faculty of Engineering, Zagazig University, Zagazig, Egypt, who taught me and gave me the scientific base to continue my postgraduate studies.

I am also very grateful to all the members of the Manufacturing Engineering Centre for their friendship and help. Special thanks go to my fellow Ph.D. students Mr. Z. Cai and Mr. S. Bigot for their useful technical discussions.

My most sincere gratitude and appreciation go to my dear wife Z. Said for her patience, continuous encouragement and support over the past difficult years. Thanks as well to Allah for his gifts; my beloved daughter "Rofaida" is truly one of them.

I am deeply indebted to my parents and all the members of my family who gave me continuous support and encouragement throughout my life.

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed.. (Candidate)
Date...................................

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed .... (Candidate)
Date.................................

Statement 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed .... (Candidate)
Date.................................

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

DM	Data Mining
ECOC	Error Correcting Output Coding
IREP	Incremental Reduced Error Pruning
JIT	just-in-time
KDD	Knowledge Discovery from Databases
MDL	Minimum Description Length
MDL_PP	Minimum Description Length based Post Pruning
MDL_HP	Minimum Description Length based Hybrid Pruning
MDL_IP	Minimum Description Length based Incremental Pruning
MML	Minimum Message Length
REP	Reduced Error Pruning
RULES	RULe Extraction System
RULES-3 Plus	RULe Extraction System – Version 3 Plus
RULES-6	RULe Extraction System – Version 6
SRI	Scalable Rule Induction
STM	Short Term Memory
TDP	Top-Down Pruning
VL1	Variable-valued Logic System 1

SYMBOLS

A_i — the i^{th} attribute in an example.

b — the number of partitions resulting from the test T.

C — the number of instances covered by the rule set R.

$\overline{C}$ — the number of instances not covered by the rule set R.

C_j — the j^{th} class.

C_t — the target class (the class to be learned).

$Cond_i$ — the i^{th} condition in a given rule.

d — the number of distinct values for a continuous attribute A_i.

D — a training data set.

e — the total number of errors ($f_p + f_n$).

E — the expected frequency distribution of instances.

f_n — the number of false negative instances.

f_p — the number of false positive instances.

F — the observed frequency distribution of instances among classes satisfying a given complex.

GT — a continuous attribute split of type "greater-than (>)".

GT_Conds — a sequence of conditions where each condition is an interval of values (greater than some threshold value) of the continuous attribute.

k — the number of classes in a data set.

LTE — a continuous attribute split of type "less-than-or-equal (≤)".

LTE_Conds — a sequence of conditions where each condition is an interval of values (less-than-or-equal-to some threshold value) of the continuous attribute.

m — a domain dependent parameter.

m_1 — the number of nominal attributes out of N_n attributes in the antecedent of the rule.

m_2 — the number of continuous attributes out of N_c attributes used to create conditions of the form $A_i > t_{ij}$.

m_3 — the number of continuous attributes out of N_c attributes used to create conditions of the form $A_i \leq t_{ij.}$

M — a possible model that might explain the training data set D.

n_c — the number of conditions in a rule.

n_{class} — the number of positive instances covered by a given rule.

$n_{covered}$ — the total number of instances covered by a given rule.

n_{A_i} — the number of possible values for a nominal attribute A_i.

n_{C_j} — the number of instances in class C_j.

N — the total number of instances in the training data set D.

N_a — the total number of attributes.

N_c — the total number of continuous attributes.

N_n — the total number of nominal attributes.

Nominal_Conds — a sequence of conditions where each condition is a value of the nominal attribute.

p — the probability of a message selection.

P — the number of instances belonging to the target class C_t.

P(C$_j$, S) — the proportion of instances in *S* which are in class C_j.

$P_0(C_t)$ — the *a priori* probability of the target class C_t.

r — a rule in the rule set *R*.

r' — any specialisation of rule *r*.

R — a set of rules.

s — a seed example.

S — a set of instances.

|*S*| — the number of instances in *S*.

S_i — the i^{th} partitions of the data set *S*.

t — the number of exceptional instances that are erroneously classified by a model *M*.

t_{ij} — the j^{th} threshold value (cutting point) in the domain of a continuous attribute A_i.

T — a test for partitioning *S* at a decision-tree node.

v_{ij} — the j^{th} value for a nominal attribute A_i.

v_{is} — the value of attribute A_i in the seed example *s*.

Value (C$_j$, a) — The proportions of correctly classified instances of all the rules that cover an instance *a* and belong to the same class C_j.

Value (r, a) — The proportion of correctly classified instances of rule *r* that covers an instance *a*.

w — the beam width.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Recent developments in information technology have facilitated the collection and storage of massive amounts of data. It is no longer practical to rely on traditional manual data analysis due to the large amount of data involved. To utilise this abundant data resource effectively, a way of distilling information and knowledge from the data has to be found. There is a need for effective techniques to refine such data.

Statistics is a powerful tool in data analysis. From a modelling perspective, it mainly focuses on finding a model to fit the available data. However, this model is usually determined *a priori* and comes from a restricted set, e.g. a linear model, or additive Gaussian model. There are also computational and theoretical difficulties in applying statistical methods to data having high dimensions and large volumes. Furthermore, it is not easy for a user to employ statistical modelling techniques without a deep knowledge of statistics and the underlying domain.

Machine learning helps this work by finding patterns, trends and dependencies hidden in the data and inducing models that have predictive power. Machine learning techniques have greatly extended and enhanced traditional statistical data analysis. However, they also have limitations. First, the volume of data these techniques can handle is usually small. Second, they concentrate on simulating the reasoning intelligence of human beings while ignoring important practical issues such as how to prepare the data. Consequently, the applications of these techniques are generally limited to particular areas, e.g. medical diagnosis.

The new field of data mining (DM) has attracted research efforts from the domains of databases, enterprise information systems, statistics, machine learning, artificial intelligence and pattern recognition with the aim of transferring the rich data possessed by enterprises into rich knowledge for better decision making. Data mining includes all the activities involved in finding interesting patterns in data. A clear definition of data mining is given in (Fayyad et al., 1996a).

> *"Data mining, which is also referred to as knowledge discovery in databases (KDD), means a process of nontrivial extraction of implicit, previously unknown and potentially useful information, such as rules, constraints, regularities from data in databases".*

The most important step in data mining concerns applying appropriate data mining algorithms to the prepared data. There are many different kinds of algorithms, such as those for association rule discovery (Agrawal et al., 1993; Megiddo and Srikant, 1998), classification learning (Quinlan, 1993; Cohen, 1995; Mehta et al., 1996; Shafer et al., 1996; Rastogi and Shim, 1998), and clustering (Zhang et al., 1997; Guha et al., 1998).

Classification learning is the most common data mining technique. It employs a set of pre-categorised examples to develop a model that can classify new examples from the same population. Classification learning has a wide range of applications, including scientific experimentation, manufacturing, telecommunications, medical diagnosis, fraud detection, credit approval and target marketing (Braha, 2001; Monostori, 2002; Pham et al., 2002; Lavrač et al., 2004; Pham and Afify, 2004). Among the techniques developed for classification learning, popular ones include inductive learning algorithms such as decision tree induction and rule induction, instance-based learning, neural networks, genetic algorithms and Bayesian learning algorithms (Han and kamber, 2001; Witten and Frank, 2000; Giudici, 2003). Among these techniques, inductive learning techniques are particularly suited to data mining (Apté and Weiss, 1997; Pham and Afify, 2002). They are simple and fast. Another advantage is that they generate models that are easy to understand. Finally, inductive learning classifiers are more accurate compared with other classification techniques.

Inductive learning algorithms have proven to be valuable, practical tools for classification, but run into difficulties in their application to large, complex problems. Most existing algorithms are prohibitively inefficient when it comes to dealing with large data sets (Aronis and Provost, 1997). One of the defining challenges for the knowledge discovery and data mining community is to develop inductive learning algorithms that can scale up to large data sets (Fayyad et al., 1996; Fayyad et al., 1996b; Piatetsky-Shapiro et al., 1996; Mitchell, 1999a; Provost and Kolluri, 1999; Klösgen and Żytkow, 2002). "Scalability" means the ability of an algorithm to process large data sets efficiently, while building from them the best possible models. However, the existence of very large data sets is not the only reason for scalability. The most cited reason for scaling up is that increasing the size of the training set often improves the accuracy of learned classification models (Catlett, 1991a). Another reason is that scaling up to very large data sets implies, in part, that fast learning algorithms must be developed. There are, however, other motivations for fast learning. For example, interactive induction, in which an inductive learner and a human analyst interact in real time, requires very fast learning algorithms in order to be practical. Wrapper approaches, which for a particular problem and algorithm iteratively search for feature subsets or good parameter settings (Provost, 1992; Kohavi, 1995a; Provost and Buchanan, 1995; Kohavi and John, 1997), also require very fast learning because such systems run the learning algorithms multiple times, evaluating them under different conditions. Furthermore, each evaluation may involve multiple runs to produce performance statistics (e.g., using cross-validation). As a final example, the popular practice of learning multiple models and combining their predictions also multiplies the execution time.

Different techniques have been proposed and implemented for scaling up inductive learning algorithms. Several scalable decision tree learning algorithms have been developed, which are considerably faster than their predecessors (Mehta et al., 1996; Shafer et al., 1996; Rastogi and Shim, 1998). However, due to its representation of rules and its strategy for induction, decision tree learning has a number of problems. The first problem is called the *replication* problem (Pagallo and Hausseler, 1990). It often happens that identical subtrees have to be learned at various places in a decision tree. Another problem is known as the *redundancy* problem (Cendrowska, 1987). By minimising the average entropy of a set of instances, a decision tree algorithm, such as ID3, disregards the fact that some attributes or attribute values may be irrelevant to a particular classification. Rule induction algorithms, on the other hand, do not suffer from these problems. They have the advantage that the knowledge of domain experts can be incorporated into the rule learning process. Also, rule induction algorithms can be extended naturally to the first-order inductive logic programming framework (Fürnkranz, 1999). One weakness with rule induction algorithms, however, is that they often scale relatively poorly with the sample size, particularly on noisy data. Given the prevalence of large noisy data sets in real-world applications, this problem is of critical importance.

1.2 Research Objectives

The overall aim of this research was to design and develop scalable rule induction algorithms suitable for data mining applications. These algorithms should be able to handle large noisy data sets in an efficient and effective way. Moreover, they should be able to deal properly with both continuous and nominal attributes. Finally, their generated models should be comprehensible to users without machine learning expertise. Accordingly, they would be able to achieve good accuracy, compact rule sets and fast execution times. To achieve the overall aim of the research, the following objectives were set:

♦ To perform a detailed analysis of existing machine learning techniques for classification learning, with particular emphasis on inductive learning, and to assess their appropriateness for data mining applications.

- To develop computationally efficient rule induction algorithms that can scale up well to larger and more complex problems.

- To design a fast and effective on-line discretisation method for use in rule induction algorithms.

- To develop new pruning techniques for rule induction algorithms that can significantly reduce rule-set sizes and execution times, and also improve accuracy.

1.3 Thesis Organisation

The remainder of the thesis is organised as follows:

Chapter 2 defines the classification learning problem, presents a framework for viewing approaches to it, discussing in some detail inductive learning algorithms and briefly reviews other machine learning approaches. Current trends and recent developments in machine learning research are also presented.

Chapter 3 presents a new rule induction algorithm which broadly follows the approach of CN2-like learning algorithms. The proposed algorithm uses advanced search techniques and rule-space pruning strategies to efficiently explore the exponential rule spaces involved in many learning problems. These techniques and strategies are detailed and analysed. A comprehensive empirical evaluation of the algorithm is also reported and discussed.

Chapter 4 proposes a new method for discretising continuous-valued attributes during the learning process. The chapter starts with a review of current discretisation approaches in classification learning and is followed by a detailed description of the new discretisation method. Finally, the chapter gives the results of experiments carried out to demonstrate the performance of the proposed method.

Chapter 5 addresses the problem of handling noisy data by developing three novel pruning techniques that can be used with rule induction systems. These techniques are built on the theoretically sound Minimum Description Length (MDL) principle. The chapter first reviews previous work on pruning in the context of inductive learning. The principles of MDL as used in pruning and a modified coding scheme are then presented. This is followed by a description of the complete pruning techniques. Finally, the performance results are discussed.

Chapter 6 focuses on the improvement of a simple rule induction algorithm, RULES-3 Plus, based on the results of the last three chapters. RULES-3 Plus, which follows the approach of AQ-like learning family of algorithms, is extended so that it works faster and can effectively handle continuous attributes and noisy data. The chapter first gives a brief description of the RULES-3 Plus algorithm. Then, extensions to the algorithm are discussed. Finally, details of the various conducted experiments are provided.

Finally, chapter 7 summarises the contributions and conclusions of the thesis and proposes directions for further research.

Appendix A describes all the data sets used in this work.

Appendix B contains a pseudo-code of the AQ15 algorithm.

Appendix C shows the control procedure of the CN2 algorithm for both ordered and unordered rules as well as the beam search procedure.

CHAPTER 2

APPROACHES TO CLASSIFICATION LEARNING

2.1 Preliminaries

Artificial intelligence is a subfield of computer science, which is concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics associated with intelligence in human behaviour – understanding language, learning, reasoning, solving problems, and so on. Learning is clearly one of the hallmarks of intelligence and the subfield of artificial intelligence concerned with it is called machine learning. The field of machine learning is concerned with enabling computer programs automatically to improve their performance at some tasks through experience.

A great deal of research in machine learning has focused on concept learning or classification learning, that is, the task of inducing the definition of a general category from specific positive and negative examples of that category. Among the various machine learning approaches developed for classification, inductive learning from instances is perhaps the most commonly adopted in real-world application domains. Inductive learning is the inference of general patterns from data. The study of inductive learning is mainly motivated by the desire to automate the process of knowledge acquisition during the construction of expert systems. Inductive learning has gained attention recently in the context of data mining (DM) and knowledge discovery in databases (KDD).

This chapter gives an overview of machine learning approaches to classification learning. The chapter is organised as follows. Section 2 formally defines the classification learning problem and presents a framework for viewing approaches to it. Section 3 describes in some detail different techniques for inductive learning. Section 4 briefly reviews other major machine learning approaches. Current trends and recent developments in machine learning research are presented in Section 5. Section 6 concludes the chapter with a summary of some of the key research issues in machine learning.

2.2 The Supervised Classification Learning Problem

In classification learning, a learning algorithm is given a sample of pre-classified examples from the problem domain called the training set. Each example is described by a vector of attributes. An attribute is either nominal or continuous. The algorithm learns a model that is used to predict the class of future examples.

Learning methods can be divided into supervised and unsupervised schemes based on whether or not a dedicated target function for prediction has been defined. In unsupervised methods, such a function is not available and the goal is grouping or clustering instances based on some similarity or distance measure. In supervised learning, there is either a nominal or continuous-valued target function to be predicted. The former case is referred to as classification and the latter as regression or continuous prediction. In this thesis, only methods for supervised classification learning will be addressed.

If the examples in the training set are presented and used all at once, learning is said to be batch or off-line. If the examples are presented one at a time, and the concept definition evolves over time as successive examples are incorporated, learning is said to be incremental or on-line. This thesis concentrates on batch learning.

The main goal of a classification learning system is to produce a classifier that will assign previously-unseen examples (i.e., examples not in the training set) to the corresponding classes with high accuracy. The accuracy of a classifier is defined as the probability that it will correctly classify a new, unlabelled example. This accuracy can be estimated by presenting the classifier with unlabelled examples from a test set.

Ideally, given a complete description of an example (i.e., the values of all its attributes), its class should be unambiguously determined. In practical tasks, however, the available attributes will often not contain all the information necessary to do this. The training set may contain examples with the same attribute values but in different classes. Also, examples may appear with erroneous class values, or with erroneous attribute values, or both. These errors may stem from a diversity of sources, including limitations of measuring instruments, and human error while typing examples into a computer. All these phenomena, referred to collectively as noise, limit the achievable accuracy in an induction problem. The degree of robustness of a learning system with respect to noise is one of its most important characteristics. It also occurs often in practice that the values of certain attributes for certain examples are simply not available. These are called missing values, and again a practical induction system must be able to handle them.

2.3 Description of Inductive Learning Algorithms

A classification learning algorithm can be viewed as having three components: representation, search, and evaluation (Fayyad et al., 1996a). The representation component is the formal language in which concepts are described; the output of the learning algorithm is a statement in this language. The search procedure is the process by which the learning algorithm finds the concept description in the space of possible descriptions defined by the representation language. The evaluation component takes a candidate concept description and returns a measure of its quality. This is used to guide the search, and possibly to decide when to terminate it. Often, different evaluation procedures are used for these two purposes.

Inductive learning algorithms can be divided into two main categories, namely, decision tree induction and rule induction. Each of these categories will be analysed in view of the above three components.

2.3.1 Decision Tree Induction

There are a variety of algorithms for building decision trees. The most popular are: CART (Breiman et al., 1984), ID3 and its descendants C4.5 and C5.0 (Quinlan, 1983; 1986; 1993; ISL, 1998; RuleQuest, 2001). These learning systems are categorised as "divide-and-conquer" inductive systems. The knowledge induced by these systems is represented as decision trees. A decision tree consists of internal nodes and leaf nodes. Each internal node represents a test on an attribute and each outgoing branch corresponds to a possible result of this test. For a nominal attribute A_i with n_{A_i} possible values

v_{i1}, v_{i2},..., v_{ij},...., $v_{in_{A_i}}$ there are n_{A_i} different branches originating from an internal node. For a continuous attribute A_i, a binary test is carried out, and a corresponding branch $A_i \leq t_{ij}$ is created, with a second branch corresponding to $A_i > t_{ij}$, where t_{ij} is a threshold in the domain of A_i. Each leaf node represents a classification to be assigned to an example. Table 2.1 shows an example data set and Figure 2.1 displays a decision tree constructed from this data.

To classify a new example, a path from the root of the decision tree to a leaf node is identified based on values of the attributes of the example. The class at the leaf node represents the predicted class for that example.

Decision trees are generated from training data in a top-down, general-to-specific direction. The initial state of a decision tree is the root node that is assigned all the examples from the training set. If it is the case that all examples belong to the same class, then no further decisions need to be made to partition the examples, and the solution is complete. If examples at this node belong to two or more classes, then a test is made at the node, which will result in a split. The process is recursively repeated for each of the new intermediate nodes until a completely discriminating tree is obtained.

CART is a binary decision tree algorithm that is extensively used. The evaluation function used for splitting in CART is the *Gini index*. Given a labelled data set S with k classes, let k classes be C_1, C_2,...., C_k and let $P(C_j, S)$ be the proportion of instances in S which are in class C_j. Then the index is defined as:

Vibration	Pressure	Temperature	Fault Type
Present	30	65	A
Absent	23	15	B
Absent	40	75	B
Present	55	40	A
Absent	55	100	B
Present	45	60	A
Present	25	55	A
Absent	24	20	B

Table 2.1 An example of a data set.

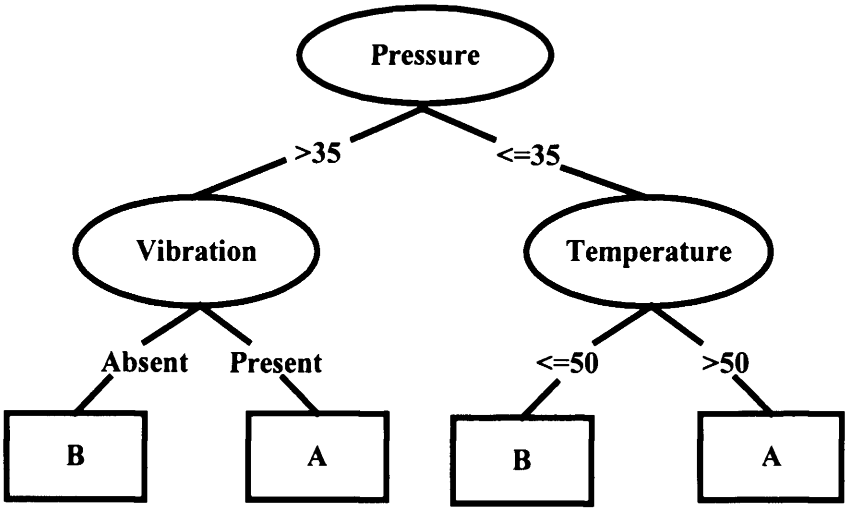


Figure 2.1 A decision tree constructed from the data in Table 2.1.

$$Gini\ (S) = 1 - \sum_{j=1}^{k} P(C_j, S)^2 \qquad (2.1)$$

For each candidate split, the "impurity" (as defined by the Gini index) of all the sub-partitions is summed and the split that causes the maximum reduction in impurity is chosen.

ID3 is a well-known decision tree system. It utilises the *information gain* criterion for splitting nodes. The information gain is computed from the entropy measure that characterises the impurity in a collection of training instances as explained below. For a given data set *S*, the entropy is defined as:

$$Entropy\ (S) = -\sum_{j=1}^{k} P(C_j, S) \log_2 P(C_j, S) \qquad (2.2)$$

Let a test *T* with *b* outcomes partition the data set *S* into $S_1, S_2, \ldots\ldots, S_b$. Then, the total entropy of the partitioned data set is defined as the weighted sum of the entropy of the subsets as described below:

$$Entropy\,(S,T) = \sum_{i=1}^{b} \frac{|S_i|}{|S|} Entropy\,(S_i) \qquad (2.3)$$

where $|S_i|$ and $|S|$ are the numbers of instances in S_i and *S* respectively.

The information gained by partitioning in accordance with the test *T* is measured by:

$$Gain(S,T) = Entropy(S) - Entropy(S,T) \tag{2.4}$$

Gain (S,T) is therefore the expected reduction in entropy as a result of partitioning the data set into mutually exclusive subsets based on test *T*. The gain criterion selects a test to maximise this information.

C4.5, a variant and extension of ID3, is another popular decision tree algorithm. It employs the *gain ratio* criterion, because the information gain criterion has a strong bias in favour of attribute tests with many values. To reduce the bias of the gain criterion, the *split information* measure as defined by the following equation is employed:

$$SplitInformation(S,T) = -\sum_{i=1}^{b} \frac{|S_i|}{|S|} \cdot \log_2\left(\frac{|S_i|}{|S|}\right) \tag{2.5}$$

The split information measure can be regarded as the cost of selecting a given attribute as a test. Notice that it discourages the selection of attributes with many values.

The gain ratio is then given by:

$$GainRatio(T) = \frac{Gain(S,T)}{SplitInformation(S,T)} \tag{2.6}$$

The gain ratio computation for a nominal attribute test is relatively straightforward. For continuous attributes, the *d* possible values appearing in the subset associated with an internal node are sorted. Then, all d-1 possible splits on this continuous attribute are examined. The one that maximises the gain ratio criterion is selected as a threshold.

A decision tree generated as described above is potentially an over-fitted solution, i.e., it may have components that are too specific to noise and outliers that may be present in the training data. To relax this overfitting, C4.5 uses a tree pruning method that tries to simplify the tree by eliminating subtrees that seem too specific. Pruning is done by examining each subtree and replacing it with one of its branches or leaf nodes if such a replacement does not degrade the accuracy of the subtree.

The C4.5 inductive learning system can also transform the generated decision tree to a set of IF-THEN rules. For the transformation to a rule set, every path from the root of the unpruned tree to a leaf gives one initial rule, in which the left-hand side is the conjunction of all attribute-based tests established by the path, and the right-hand side specifies the class predicted at the leaf. If the path to each leaf node is transformed into a production rule, the resulting collection of rules would classify examples exactly as the tree and, as a consequence of their tree origin, the rules would be mutually exclusive and hence their order would not matter. After producing a rule set from an unpruned tree, C4.5 implements a very complicated multiphase rule pruning procedure. First, each rule is simplified by deleting some conditions based on the *pessimistic-error* estimate as adopted in tree pruning. Second, the set of rules is partitioned into several groups according to the rule consequent, with one group corresponding to one class. All possible subsets of rules from each group are then examined and the best subset based upon the Minimum

Description Length (MDL) principle is selected. In the third stage, all the rule subsets are ordered, based on their classification error on the training data set. A default rule is then chosen whose consequent is the class that contains the largest number of training instances not covered by any rule. The pruning procedure then attempts to reduce the size of the rule set further by eliminating rules, the removal of which does not cause a deterioration in the accuracy of training data classification.

2.3.2 Rule Induction

As with decision tree learning, there are many rule induction algorithms. Among them are AQ (Michalski, 1969; Michalski et al., 1986; Cervone et al., 2001; Michalski and Kaufman, 2001), CN2 (Clark and Niblett, 1989; Clark and Boswell, 1991) and RIPPER (Cohen, 1995) which can all be categorised as "separate-and-conquer" inductive systems.

In contrast to decision tree learning, rule induction directly generates IF-THEN rules. Each rule can be represented in the following form: $Cond_1 \wedge ... \wedge Cond_i \wedge ... \wedge Cond_{n_c} \rightarrow C_j$, where the antecedent consists of a conjunction of conditions $Cond_i$. Each condition takes the form $[A_i = v_{ij}]$ or $[t_{i1} < A_i \leq t_{i2}]$ depending on the property of the attribute A_i. If A_i is a nominal attribute, v_{ij} is a valid nominal value that A_i can take. If A_i is a continuous attribute, t_{i1} and t_{i2} are two thresholds in the domain of attribute A_i. The consequent is the class to which instances satisfying the antecedent can be assigned. Figure 2.2 displays a rule set generated from the data set given in Table 2.1.

If [Pressure > 35] [Vibration = Absent] → B

If [Pressure > 35] [Vibration = Present] → A

If [Pressure ≤ 35] [Temperature ≤ 50] → B

If [Pressure ≤ 35] [Temperature > 50] → A

Figure 2.2 A set of rules derived from the data in Table 2.1.

Rule induction systems produce either an unordered set of IF-THEN rules or an ordered set of IF-THEN rules, also known as decision lists (Rivest, 1987), both including a default rule. To classify an instance in the case of ordered rules, the ordered list of rules is examined to find the first whose antecedent is satisfied by the instance. The predicted class is then the one nominated by this rule. If no rule antecedent is satisfied, the instance is predicted to belong to the default class. In the case of unordered rules, it is possible for some instances to be covered by more than one rule. To classify a new instance in this case, some conflict resolution approach must be employed.

The general operation of separate-and-conquer rule induction algorithms is the same. They create the rule set one rule at a time. After a rule is generated, the instances covered by it are removed from the training data set and the same induction procedure is applied to the remaining data set until all the instances are covered by at least one rule in the rule set.

AQ15 (Michalski et al., 1986) is a well-known inductive learning system. It is based on the AQ algorithm as originally described in (Michalski, 1969) and implements the STAR method of inductive learning (Michalski and Larson, 1983). A pseudo-code listing of the AQ15 algorithm is given in appendix B.

In AQ15, decision rules are represented as expressions in the *Variable-valued Logic System 1 (VL1)*. VL1 is a multiple-valued extension to propositional logic. In VL1, a *selector* relates an attribute to an attribute value or disjunct of values using one of the relational operators <, ≤, =, !=, ≥, or >. A selector or a conjunction of selectors forms a *complex*. A *cover* is a disjunction of complexes describing all positive instances and none of the negative instances of the concept. A cover defines the condition part of a corresponding decision rule. AQ15 is able to implement a form of constructive induction as well. An example of a decision rule with an internal disjunct is:

[Outlook = sunny $\vee$ *cloudy]* $\wedge$ *[Temperature > 60]* $\vee$ *[Wind = true]* $\wedge$ *[Temperature >70]* → *class [Nice]*

When building a complex, AQ15 performs the general-to-specific beam search technique to find the best complex. The algorithm considers specialisations that exclude some particular covered negative instances from the complex, while ensuring some particular "seed" positive instances remain covered, iterating until all negative instances are excluded. As a result, AQ15 searches only the space of complexes that are completely consistent with the data. Seeds are selected at random and negative examples are chosen according to their distance from the seed (the nearest ones are picked first, where distance is the number of attributes with different values in the seed and negative instances).

The AQ15 system can generate unordered and ordered rules. In the case of unordered rules, a new instance is classified by finding which of the induced rules have their complexes satisfied by the instance. If the instance satisfies only one rule, then the class predicted by that rule is assigned to the instance. If the instance satisfies more than one rule, a heuristic called *Estimate of Probability* (EP) is used to predict its class. With this method, each rule is weighted by the proportion of learning instances covered by it. The weights of rules of the same class are probabilistically combined to form a weight for the entire class and the class with the highest weight is taken as the predicted class of the test example. If the instance is not covered by any rule, a heuristic called *Measure of Fit* (MF) is used. In this case the instance belongs to a part of the decision space that is not covered by any decision rule. The measure of best fit of a class can be interpreted as a combination of "closeness" of the instances to a class and an estimate of the prior probability of the class.

The AQ15 algorithm uses a post-pruning technique to remove redundant conditions from the body of a rule and to remove unnecessary rules from the rule set. Simplification generally leads to smaller, more accurate rule sets. This framework was later generalised in the POSEIDON system (Bergadano et al., 1992). POSEIDON can simplify a complete and consistent concept description, which has been induced by the AQ15 algorithm, by removing conditions and rules, and by contracting and extending intervals and internal disjunctions. POSEIDON successively applies the operator that results in the highest coverage gain as long as the resulting rule set increases some quality criterion.

CN2 is a rule induction algorithm that incorporates ideas from both ID3 and AQ. The representation of decision rules in CN2 is very similar to that of AQ15 and can be viewed as a subset of VL1. The inductive learning system CN2 was developed by Clark and Niblett (1987; 1989) and later modified by Clark and Boswell (1991). The objective behind the design of CN2 was to modify the AQ algorithm by retaining its beam search through the space of complexes, but removing its dependency on specific training instances during search. While the AQ algorithm searches only the space of complexes that are completely consistent with the training data, CN2 extends its search space to rules that do not perform perfectly on the training data by broadening the specialisation process to examine all specialisations of a complex, in much the same way as ID3 considers all attribute tests when growing a node in a tree. A cut-off method similar to decision tree pruning is applied to halt specialisation when no further specialisations are statistically significant. The modified version of CN2 produces either an ordered set of IF-THEN rules like the original CN2 version or an unordered set of IF-THEN rules. The control procedure of the CN2 algorithm for both ordered and unordered rules as well as the beam search procedure are given in appendix C.

The CN2 algorithm consists of two main procedures: a search algorithm performing a beam search for a good rule and a control algorithm for repeatedly executing the search. The control procedure of the CN2 algorithm for ordered rules iteratively calls the beam search procedure to find the best complex, until no better complexes are found. It then appends a rule to the rule set with this best complex as the condition and the most common class among the instances covered by this complex as the prediction. The instances covered by a rule are removed from the training set. The last rule in the rule list is a default rule predicting the most common class among the training examples not covered. The beam search procedure to find the best complex corresponds to the STAR procedure of the AQ algorithm. The pruned general-to-specific search retains a size-limited set or star of "best complexes found so far". The system examines only specialisations of this set, carrying out a beam search for the space of complexes. A complex is specialised by either adding a new selector to the conjunction or by removing a disjunctive element in one of its selectors.

The CN2 algorithm can be easily modified to generate an unordered rule set by changing only the control procedure, leaving the beam search procedure unaltered (apart from the evaluation function, described below). The main modification to the algorithm is to iterate the search for each class in turn, removing only covered instances of the current class where a rule has been found. Unlike the case for ordered rules, the negative instances remain because now each rule must independently stand against all negatives. The covered positives must be removed to stop CN2 from repeatedly finding the same rule.

The CN2 algorithm employs two types of heuristics in the search for the best complexes, goodness and significance. Goodness is a measure of the quality of the complex that is used for ordering complexes that are candidates for inclusion in the final cover. Like ID3, the original CN2 version used the information-theoretic *entropy* measure to evaluate the quality of the complex (the lower the entropy, the better the complex). This function prefers complexes covering a large number of instances of a single class and few examples of other classes, but it tends to select very specific rules covering only a few training instances. The modified version of CN2 employs the *Laplacian error estimate* instead. The expected accuracy, one minus the expected Laplacian error estimate, is given by:

$$LaplaceAccuracy(n_{class}, n_{covered}, k) = \frac{n_{class} + 1}{n_{covered} + k} \qquad (2.7)$$

where k is the number of classes, n_{class} is the number of positive instances covered by the rule and $n_{covered}$ is the number of instances covered by the rule. This formula is a special case of the *m-probability-estimate* developed in (Cestnik, 1990). This estimate avoids the downward bias of the entropy measure of favouring very specific complexes in the general-to-specific search operation.

The second evaluation function tests whether a complex is statistically significant, i.e. whether it locates a regularity that is unlikely to have occurred by chance and thus reflects a genuine correlation between attribute values and classes in the training data. To test significance, CN2 uses the *likelihood ratio statistic* (Kalbfleish, 1979). This is given by:

$$LikelihoodRatio(F, E) = 2 \cdot \sum_{i=1}^{n} f_i \cdot \log \frac{f_i}{e_i} \qquad (2.8)$$

where the distribution F = $(f_1, f_2, \ldots\ldots, f_n)$ is the observed frequency distribution of instances among classes satisfying a given complex and E = $(e_1, e_2, \ldots\ldots, e_n)$ is the expected frequency distribution of the same number of instances under the assumption that the complex selects instances randomly from the training set. Thus the two functions, the Laplacian error estimate and statistical significance serve to determine whether complexes found during the search are both "good" (have high accuracy when predicting the majority class covered) and "reliable" (the high accuracy on the training data is not just due to chance).

CN2 performs another check that can be viewed as a form of pre-pruning. It checks whether the Laplace estimate of the best complex is greater than that of the default rule predicting the class with the largest number of training instances. If this is not the case, then the new complex does not contribute any new information and the generation of complexes for the current class is terminated.

To apply unordered rules to classify a new instance, all rules are tried and those whose conditions are all satisfied are collected. If a clash occurs, i.e., more than one class is predicted by the collected rules, a probabilistic method is employed to resolve the clash. Each rule is tagged with the distribution of covered instances among classes and these distributions are summed to find the most probable class.

RULES (RULe Extraction System) is a set of inductive learning algorithms that follow a similar approach to the AQ family. The first three algorithms in the RULES family of algorithms (RULES-1, 2 and 3) were developed by Pham and Aksoy (1993; 1995a; 1995b). Later, Pham and Dimov (1997a) introduced a new algorithm called RULES-3 Plus. Compared to its immediate predecessor RULES-3, RULES-3 Plus has two new strong features. First, it employs a more efficient search procedure instead of the exhaustive search conducted in RULES-3. Second, it incorporates a metric for selecting and sorting candidate rules according to their generality and accuracy. RULES-3 does not employ any measure for assessing the information content of rules. The first incremental learning algorithm in the RULES family was RULES-4 (Pham and Dimov, 1997b). It allows the stored knowledge to be updated and refined rapidly when new examples are available. RULES-4 employs a *Short Term Memory* (STM) to store training examples when they become available. The STM has a user-specified size. When the STM is full, the RULES-3 Plus algorithm is used to generate rules. In order to increase the efficiency of the RULES family of algorithms, Pham et al. (2000) used a simple clustering technique to select a good set of training examples that were representative of the overall data set. The method was tested on different problems. The results showed that when the algorithm was applied to clustered data sets, the execution time was reduced, as well as the size of the generated rule sets. Pham et al. (2003) described a new algorithm, called RULES-5, which overcomes some of the deficiencies of the RULES family. In particular, RULES-5 employs a new method for handling continuous attributes and uses a simple and more efficient search method. The test results obtained with RULES-5 showed that the rule sets extracted were more accurate and compact than those obtained using its immediate predecessor RULES-3 Plus. One of the main weaknesses of the RULES-5 algorithm is its inability to handle noisy data. Pham et al. (2004) proposed a new pruning technique that improved significantly the performance of the RULES-5 algorithm on data sets containing noisy examples.

2.4 Other Machine Learning Approaches to Classification Learning

Besides decision trees and rule induction, several other approaches to classification learning exist. This section will briefly review some of the main alternatives: instance-based learning, neural networks, genetic algorithms and Bayesian methods.

2.4.1 Instance-based Learning

Instance-based learning is based upon the idea of letting the examples themselves form an implicit representation of the target concept (Aha et al., 1991; Aha, 1997). In contrast to learning methods that construct a general, explicit description of the target concept when training instances are provided, instance-based learning methods, such as those using nearest-neighbour methods, simply store the training instances. Generalising beyond these instances is postponed until a new instance must be classified. Because of this, instance-based methods are sometimes referred to as "lazy" learning methods. A test instance is classified by finding the nearest stored instance according to some similarity function, and assigning the class of the latter to the former. Advantages of instance-based methods include the ability to model complex target concepts and the fact that information present in the training instances is never lost (because the instances themselves are stored explicitly). One disadvantage of instance-based approaches is that the cost of classifying new instances can be high. This is because nearly all the computation takes place at classification time rather than when the training instances are first encountered. Therefore, techniques for efficiently indexing training instances are a significant practical issue in reducing the computation required at classification time. A second disadvantage of many instance-based approaches, especially nearest-neighbour methods, is that they typically consider all attributes of the instances when attempting to retrieve similar training instances from the memory. If the target concept depends on only a few of the many available attributes, then the instances that are really most "similar" may be a long distance apart.

2.4.2 Neural Networks

Neural networks provide a general practical method for learning real-valued and discrete-valued target concepts in a way that is robust to noise in the training data (Haykin, 1994; Michie et al., 1994; Chauvin and Rumelhart, 1995; Hassoun, 1995; Mitchell, 1997; Pham and Liu, 1999). Neural network learning is well-suited to problems in which the training data corresponds to noisy and complex sensor data, such as inputs from cameras and microphones. The backpropagation algorithm is a common learning method adopted for multi-layer perceptrons, the most popular type of neural networks. Neural networks have been successfully applied to a variety of learning tasks, such as setting the number of *kanbans* in a dynamic just-in-time (JIT) factory (Wray et al., 1997; Markham et al., 2000), modelling and controlling dynamic systems including robot arms (Pham and Liu, 1999), identifying arbitrary geometric and manufacturing categories in CAD databases (Ip et al., 2003) and minimising the makespan in a flow shop scheduling problem (Akyol, 2004). One of the chief advantages of neural networks is their wide applicability, however, they also have two particular drawbacks. The first is the difficulty in understanding the models they produce. The second is the often time-consuming training. Recent years have seen much research in developing new neural network methods that effectively address these comprehensibility and speed issues (Towell and Shavlik, 1993; Craven and Shavlik, 1997; Zhou et al., 2000; Jiang et al., 2002; Duch et al., 2004).

2.4.3 Genetic Algorithms

Genetic algorithms provide a learning method motivated by analogy with biological evolution (Holland, 1975; Goldberg, 1989; Davis, 1991; Michalewicz, 1996; Mitchell, 1996; Liu and Kwok, 2000; Pham and Karaboga, 2000; Freitas, 2002). Rules may be represented by bit strings whose particular interpretation depends on the application. The search for an appropriate rule begins with a population, or collection, of initial rules. Members of the current population give rise to the next-generation population by means of operations such as random mutation and crossover. At each step, the rules in the current population are evaluated relative to a given measure of fitness, with the fittest rules selected probabilistically as seeds for producing the next generation. The process performs generate-and-test beam-search of the rules, in which variants of the best current rules are most likely to be considered next. Genetic algorithms have been applied successfully to a variety of learning tasks and to other optimisation problems. For example, they have been used to form manufacturing cells and to determine machine layout information for cellular manufacturing (Wu et al., 2002), to optimise the topology and learning parameters for neural networks (Öztürk and Öztürk, 2004) and to solve job-shop scheduling problems (Chryssolouris and Subramaniam, 2001; Pérez et al., 2003). Genetic algorithms have a potentially greater ability to avoid local minima than is possible with the simple greedy search employed by most learning techniques. On the other hand, they have a high computational cost.

2.4.4 Bayesian Approaches

Bayesian approaches employ probabilistic concept representations, and range from a simple Bayesian classifier (Domingos and Pazzani, 1996) to Bayesian networks, which learn the full joint probability distributions of the attributes and class, as opposed to just the class description (Heckerman, 1996). Bayesian networks provide a natural platform for combining domain knowledge and empirical learning. However, inference in Bayesian networks can have a high time complexity, and as tools for classification learning, they are not yet as mature or well-tested as other approaches. More generally, as Buntine (1991) notes, the Bayesian paradigm extends beyond any single representation and forms a framework in which many learning tasks can be usefully studied.

2.5 Current Trends in Machine Learning Research

Machine learning research has been making significant progress in many directions. This section examines two of the most important directions and discusses some current problems. The two directions are scaling up of machine learning algorithms and learning multiple models.

2.5.1 Scaling up Machine Learning Algorithms

The first major research area concerns techniques for scaling up machine learning algorithms so that they can process very large data sets efficiently, while building from them the best possible models. The recent emergence of data mining as a major application of machine learning algorithms has underlined the need for algorithms to be able to handle large data sets that are currently beyond their scope. In data mining applications, data sets with millions of training examples, thousands of attributes and hundreds of classes are common. Fayyad et al. (1996a) cited several representative examples of databases containing many gigabytes (even terabytes) of data. Designing learning algorithms appropriate for such applications has thus become an important research problem.

Many approaches have been proposed and implemented for scaling up machine learning algorithms (Dash and Liu, 1997; Fürnkranz, 1998; Liu and Setiono, 1998; Moore and Lee, 1998; Zaki, 1998; Opitz, 1999; Ye and Li, 2002; Blockeel and Sebag, 2003). The most straightforward approach is to produce more efficient algorithms or increase the efficiency of the existing algorithms. This approach includes a wide variety of algorithm design techniques for optimising the search and representation, for finding approximate instead of exact solutions, or for taking advantage of the inherent parallelism of the task. A second approach is to partition the data, avoiding the need to run algorithms on very large data sets. This approach involves breaking the data set up into subsets, learning from one or more of the subsets, and possibly combining the results. Data partitioning is useful to avoid memory management problems that occur when algorithms try to process huge data sets in main memory. An approach orthogonal to the selection of example subsets is to select subsets of relevant features upon which to focus attention.

In order to provide focus and specific details, the application of inductive learning techniques to very large data sets is now reviewed; the issues and techniques discussed generalise to other types of machine learning.

Decision tree algorithms have been improved to handle large data sets efficiently and several new algorithms have been proposed. Catlett (1991a; 1991b) proposed two methods for improving the time taken to develop a classifier. The first method used data sampling at each node of the decision tree, and the second method discretised continuous attributes. These methods decrease classification time significantly but also reduce the classification accuracy. Moreover, Catlett only considered data sets that could fit in the main computer memory. Methods for partitioning the data set such that each subset fits in main memory were considered in (Chan and Stolfo, 1993; 1997; Zhang and Wu, 2001). Although these methods enable classification of large data sets, studies show that the quality of the resulting decision tree is worse than that of a classifier that was constructed by using the complete data set at once. Incremental learning methods, where the data are classified in batches, have also been studied (Wu and Lo, 1998). However, the cumulative cost of classifying data incrementally can sometimes exceed the cost of classifying the entire training set once. The decision tree classifier in (Mehta et al., 1996), called *SLIQ*, utilised the novel techniques of pre-sorting, breadth-first growth, and MDL-based pruning to improve learning time for the classifier without loss of accuracy. At the same time, these techniques allowed classification to be performed on large amounts of disk-resident training data. However, due to the use of a memory-resident data structure that scales with the size of the training set, SLIQ has an upper limit on the number of examples it can process. Shafer et al. (1996) presented a classification algorithm called *SPRINT* that removes all memory restrictions that limit existing decision tree algorithms, and yet exhibits the same excellent behaviour as SLIQ. SPRINT efficiently allows classification of virtually any sized data set. Also, the algorithm can be easily and efficiently parallelised. However, SPRINT has been criticised for several reasons. For example, it utilises data structures called attribute lists that can be costly to maintain, including a potential tripling of the size of the data set and an associated significant increase in scan cost (Graefe et al., 1998). Like C4.5, both SLIQ and SPRINT are two-stage algorithms which include building and pruning phases. Generating the decision tree in two distinct phases could result in a substantial amount of wasted effort since an entire subtree constructed in the first phase may later be pruned in the next phase. *PUBLIC* (Rastogi and Shim, 1998) is a decision tree classifier that tightly integrates the pruning phase into the building phase instead of performing them one after the other. Its tree-growing phase is the same as that of SPRINT except that it uses entropy instead of the Gini index. However, when a leaf node is generated, PUBLIC can immediately decide whether there is a need to split it further by estimating a lower bound on the cost of coding the subtree rooted at this leaf node. The integrated approach of PUBLIC can result in substantial performance improvements compared to traditional classifiers such as SPRINT. In recent work, Gehrke et al. (1998) proposed Rainforest, a framework for developing fast and effective algorithms for constructing decision trees that gracefully adapt to the amount of main memory available. Finally, Morimoto et al. (1998) developed algorithms for decision tree construction for categorical attributes with large domains. The emphasis of this work was to improve the quality of the resulting tree.

As with decision tree learning, there are a number of rule induction algorithms that can scale up to large data sets. *IREP* (Furnkranz and Widmer, 1994) is a rule learning algorithm that can efficiently handle large sets of noisy data. The main reason for its efficiency is the use of a technique called *incremental reduced error pruning*, which prunes each rule immediately after it has been induced, rather than after all rules have been generated. This speeds up the induction process as the pruned rules allow larger subsets of the remaining positive instances to be removed at each iteration compared to the non-pruned rules. Unfortunately, the accuracy of the class descriptions learned by IREP is often lower than the accuracy of those learned with the *C4.5rules* algorithm (Quinlan, 1993), a rule-inducing variant of C4.5. Cohen (1995) detailed several modifications to improve the accuracy of IREP, including a different rule-evaluation criterion, a different stopping criterion and a post-processing optimisation operation, producing an algorithm called *RIPPER*. He showed that RIPPER is competitive with C4.5rules in terms of error rates and that it maintains the efficiency of IREP. RIPPER supports missing attributes, continuous variables and multiple classes. This makes it applicable to a wider range of benchmark problems.

2.5.2 Learning Multiple Models

The second active research area concerns a particular method for improving accuracy in supervised learning. The term multiple models or ensemble of classifiers is used to identify a set of classifiers whose individual decisions are combined in some way (typically by voting) to classify new examples (Dietterich, 1997). Ensembles have been found to be more accurate than the individual classifiers that make them up (Pham and Oztemel, 1996; Bauer and Kohavi, 1999; Dietterich, 2000; Fern and Givan, 2003; Kuncheva and Whitaker, 2003), and also have substantial theoretical foundations (Friedman, 1996; Madigan et al., 1996; Schapire et al., 1997; Schapire, 1999). An ensemble can be more accurate than any of its component classifiers only if the individual classifiers are "accurate" and "diverse" (Hansen and Salamon, 1990). An accurate classifier is one that performs at least better than random guessing. Two classifiers are diverse if they make different errors on new data points. The reason why ensembles improve performance can be intuitively explained in that taking a majority vote of several hypotheses reduces the random variability of individual hypotheses.

Several methods have been proposed for generating multiple classifiers using the same learning algorithm. Most of them manipulate the training set to generate multiple hypotheses. The learning algorithm runs several times, each time using a different distribution of the training instances. This technique works especially well for unstable learning algorithms – algorithms whose output classifier undergoes major changes in response to small changes in the training data.

Breiman (1996a) described a technique called *bagging* that manipulates the training data to generate different classifiers. Bagging produces a replication of the training set by sampling with replacement from the training instances. Each replication of the training set has the same size as the original data. Some examples do not appear at all while others may appear more than once. Such a training set is called a *bootstrap replicate* of the original training set, and the technique is called *bootstrap aggregating* (from which the term bagging is derived). From each replication of the training set a classifier is generated. All classifiers are used to classify each instance in the test set, usually using a uniform voting scheme where each component classifier has the same vote. Bagging methods require that the learning system should be unstable, so that small changes to the training set should lead to different classifiers. Although Breiman also notes that poor predictors can be transformed into worse ones by bagging, it is a simple and easy way to improve an existing learning method. All that is required is the addition of a pre-processor that selects the bootstrap sample and sends it to the learning algorithm and a post-processor that does the aggregation of votes. What one loses, in comparison with decision trees and rule sets, is a simple and interpretable structure. What one gains is increased accuracy.

Freund and Schapire (1996; 1997) presented another method for manipulating the training set called *boosting*. Instead of drawing a succession of independent bootstrap samples from the original instances, boosting maintains a weight for each instance in the training set that reflects its importance – the higher the weight the more the instance influences the learned classifier. During each iteration, the weights are adjusted in accordance with the performance of the corresponding classifier, with the result that the weight of misclassified instances is increased. Adjusting the weights causes the learner to focus on different instances, leading to different classifiers. The final classifier is constructed from the learned classifiers by a weighted voting scheme where each component classifier contributes to the final classification with a different strength based on its accuracy on the training instances that it was trained with. Like bagging, boosting depends on instability of the boosted learning system. However, it does not preclude poor predictors, provided that their error on the given distribution of instances can be kept below 50%.

A third technique for constructing a good ensemble of classifiers is to manipulate the set of classes that are given to the learning algorithm. Dietterich and Bakiri (1995) described a technique called *error-correcting output coding (ECOC)*. This method was originally designed to handle multi-class problems by solving multiple two-class problems. ECOC represents classes with a set of output bits, where each bit encodes a binary classification function corresponding to a unique partition of the classes. Algorithms that use ECOC learn a function corresponding to each bit. All functions are then combined to generate class predictions.

Bagging, boosting and ECOC are general combining algorithms that significantly improve classifiers such as decision trees, rule learners, or neural networks. Quinlan (1996a) conducted experiments with boosting and bagging over a diverse collection of data sets. His experiments confirmed that boosted and bagged versions of C4.5 produced noticeably more accurate classifiers than the standard version. The results also showed that boosting seemed to be more effective than bagging when applied to C4.5, although the performance of the bagged C4.5 was less variable than that of its boosted counterpart. Freund and Schapire (1996) also applied boosting and bagging to C4.5 on 27 data sets. Their results confirmed that the error rates of boosted and bagged classifiers were significantly lower than those of single classifiers. However, they found bagging much more competitive than boosting. Bauer and Kohavi (1999) presented an empirical comparison between boosting and bagging, and argued that both techniques, when applied to decision trees, were able to reduce the error rate at the cost of increased tree size. They also observed that boosting was not robust when dealing with noise. This is expected, because noisy examples tend to be misclassified, and their weight will consequently increase. Dietterich and Bakiri (1995) reported that ECOC improved the performance of both the C4.5 and backpropagation algorithms on a variety of different classification problems. Schapire (1997) showed how boosting can be combined with ECOC to yield an excellent ensemble classification method that was superior to the ECOC method.

In addition to these methods for generating ensembles using a single learning algorithm, there are other methods that produce an ensemble by combining classifiers constructed with different learning algorithms. When classifiers from different learning algorithms are combined, as in stacked generalisation (Wolpert, 1992), diversity is implied. Therefore, they only need to be checked (e.g., by cross-validation) for accuracy, with some form of weighted combination employed. This approach of generating ensembles has been shown to be effective in some applications (Zhang et al., 1992; Breiman, 1996b; Ting and Witten, 1997).

2.6 Summary

This chapter has given background information on different machine learning algorithms with attention focused on inductive learning. The basic concepts of inductive learning algorithms have been described and the two main types of these algorithms currently available presented. The chapter has also outlined a number of algorithms of each type and discussed their suitability for handling very large data sets. Finally, recent directions in the machine learning research have been presented.

CHAPTER 3

SRI: A SCALABLE RULE INDUCTION ALGORITHM

3.1 Motivation

Classification learning can be viewed as conducting a search over the space of possible rules for the rules that best fit the training data. The large number of potential rules has prevented most induction algorithms from evaluating every rule. Most algorithms use greedy search to find a good rule by evaluating only a small fraction of all rules. Greedy search, also known as hill-climbing search, tries to find a rule with an optimal evaluation by repeatedly choosing the best partial rule at each specialisation step and halting when no further improvement is possible. While greedy search performs well on many problems, it is not guaranteed to find the best rule. Exhaustive search, on the other hand, explores all the rule space to find the best rule. This simple method will find a complete and consistent model if there is one in the search space. In theory, exhaustive search does not miss the best rule. However, this approach can have high computational costs and often decreases generalisation accuracy instead of improving it. For example, Webb (1993) used the efficient best-first search algorithm OPUS for inducing decision lists in a covering framework and found, surprisingly, that the generalisations discovered by the limited search method of CN2 were often superior to those found by an exhaustive best-first search. This is essentially because when a very large space of possible rules is exhaustively searched, there is a high probability of finding a rule set that is highly accurate on the training data purely by chance. This rule set will be chosen over others that are in fact more accurate outside the training set, leading to poorer results (Quinlan and Cameron-

This chapter is organised as follows. In section 3.2, a detailed description of the new rule induction algorithm is given. The description includes the representation scheme, the basic search method, the various forms of inductive biases, the search-space pruning rules and other efficiency considerations. The different possibilities that might result when using the learned rule set to classify unseen instances are discussed and analysed in section 3.3. A new method to solve the overlapping problem (where more than one rule match the same instances) is also presented. Section 3.4 describes the experimental method and the data sets used in empirical evaluations in this thesis. Section 3.5 gives empirical evidence that the new search-space pruning rules significantly reduce the learning time while increasing the accuracy and comprehensibility of the learned rules. This section also reports on experiments performed on real data sets to demonstrate the effectiveness of the algorithm. Section 3.6 summarises the chapter.

3.2 The SRI Algorithm

As pointed out in chapter 2, algorithms such as those in the AQ and RULES families search only the space of rules that are completely consistent with the training data. Although this reduces the search space significantly, it prevents the algorithm from exploring the complete training data space. Therefore, specific rules based on a few training examples tend to be selected. Although these rules perform perfectly on the training examples, their predictive accuracy on future test examples is often lower because rules formed on the basis of small numbers of examples are susceptible to noise. Consequently, the rule set is both large and not of the highest accuracy. For an induction system to induce accurate rules in domains containing noise, which is the norm in real-world data mining applications, the search space must be extended to include rules for which counter-examples exist and the evaluation of rules appropriately modified to enable the most accurate rules to be located. CN2 is an algorithm that functions in this way and that has significant advantages. Because statistical measures rather than individual examples are used to evaluate induction steps, good noise immunity can be achieved. Also, when the induction process stops early, as it often does, learning can be fast and the resulting rule set concise. For example, in spite of its greater computational complexity, CN2 is faster overall than AQ (Clark and Niblett, 1989). This is because the number of iterations in producing a new rule is lower in CN2 than in the AQ algorithm, as CN2 may halt specialisation of a rule before it performs perfectly on the training examples. Also, CN2 may halt the entire search for rules before all the training examples are covered if no further statistically significant rules can be found. In this section, a new rule induction algorithm, called *SRI* (for Scalable Rule Induction), is presented. The proposed SRI algorithm broadly follows the approach of CN2 and similar algorithms.

3.2.1 Representation and Basic Concepts

SRI extracts IF-THEN rules directly from a set of instances called the training set. Each instance is described by a vector of attribute-value pairs, together with a specification of the class to which it belongs. An attribute is either nominal or continuous. In supervised learning, the class to be learned is called the target class. Instances of the target class in the training set are called positive instances. Instances in the training set that do not belong to the target class are called negative instances. In SRI, an attribute-value pair constitutes a condition. Each rule, or concept description, consists of a conjunction of antecedents and a predicted class. Each antecedent is a condition on a single attribute and there is at most one antecedent per attribute. Conditions on nominal attributes are equality tests of the form [$A_i = v_{ij}$], where A_i is the attribute and v_{ij} is one of its valid values. Conditions on continuous attributes are inequalities of the form [$A_i > t_{i1}$] or [$A_i \leq t_{i2}$], where t_{i1} and t_{i2} are two thresholds in the domain of attribute A_i. A rule is said to cover an instance if the instance satisfies all of the rule conditions. A rule is said to be consistent if it covers none of the negative instances in the training set, and it is complete if it covers all the positive instances in the training set. A rule set is the disjunction of a number of rules.

3.2.2 The Search Method

SRI follows the general one-rule-at-a-time procedure of separate-and-conquer rule induction algorithms. It searches the rule space in a top-down fashion. A pseudo-code description of SRI is given in Figure 3.1. The procedure *Induce_Rules ()* starts with an empty rule set. It generates rules for each class in turn. Having chosen a class on which to focus, it calls the procedure *Induce_One_Rule ()* to extract a rule that will cover a subset of the positive instances. The *Induce_One_Rule ()* procedure is outlined in Figure 3.2. All positive instances covered are then temporarily removed from the training set, the learned rule is added to the rule set and another rule is learned from the remaining instances. Rules are learned in this way until no positive instances are left or until the rule stopping criterion is satisfied. The test in the *Rule_ Generation_Stopping_Criterion ()* procedure is employed to decide when to stop adding rules for a given class. This will be further detailed in section 3.2.3.3. Once all the rules for one class are produced, all removed instances are put back into the training set before the induction of rules for the next class. In this way, the algorithm is always based on all available training instances to form rules for a specific class. This whole process is repeated for each class to produce an unordered set of rules.

```
Procedure Induce_Rules (TrainingSet, BeamWidth)
RuleSet = ∅
For each class in the TrainingSet Do
  Instances = TrainingSet
  While Positives (Instances) ≠ ∅ Do
    Rule = Induce_One_Rule (Instances, CurrentClass, BeamWidth)
    If Rule_Generation_Stopping_Criterion (Rule, Instances) is True Then
      Exit While
    Instances = Instances – Covered_Positives (Rule, Instances)
    RuleSet = RuleSet ∪ {Rule}
  End While
End For
Return RuleSet
End
```

Figure 3.1 A pseudo-code description of SRI.

```
Procedure Induce_One_Rule (Instances, ClassLabel, $w$)
PartialRules = NewPartialRules = $\varnothing$
BestRule = most general rule (the rule with no conditions)                    (step 1)
PartialRules = PartialRules $\cup$ {BestRule}
While PartialRules $\neq \varnothing$ Do                                      (step 2)
  For each Rule $\in$ PartialRules Do
  {First, generate all specialisations of the current rule, save useful ones and determine all the
   InvalidValues according to one of the conditional tests in steps (6), (7) or (8).}
    For each nominal attribute $A_i$ that does not appear in Rule Do
      For each valid value $v_{ij}$ of $A_i \in$ Rule.ValidValues Do
        NewRule = Rule $\wedge$ $[A_i = v_{ij}]$                              (step 3)
        NewRule.Instances = Covered_Instances (Rule.Instances, $v_{ij}$)       (step 4)
        If NewRule.Score > BestRule.Score Then                                (step 5)
          BestRule = NewRule
        If Covered_Positives (NewRule) $\leq$ MinPositives OR                 (step 6)
          Covered_Negatives (Rule) – Covered_Negatives (NewRule) $\leq$ MinNegatives OR (step 7)
          Consistency (NewRule) = 100% Then                                   (step 8)
            Parent (NewRule).InvalidValues = Parent (NewRule).InvalidValues + {$v_{ij}$}   (step 9)
        Else
          NewPartialRules = NewPartialRules $\cup$ {NewRule}                  (step 10)
      End For
    End For
  End For
  Empty PartialRules
```

Figure 3.2 A pseudo-code description of the *Induce_One_Rule ()* procedure of SRI.

PartialRules: a list of rules to be specialised and ***NewPartialRules***: a new list of rules to be used for further specialisations.

```
  For each Rule ∈ NewPartialRules Do
  {Next, delete partial rules that cannot lead to an improved rules and determine all the
   InvalidValues according to the conditional test in step (11).}
    If Rule.OptimisticScore ≤ BestRule.Score Then                                    (step 11)
      NewPartialRules = NewPartialRules – {Rule}                                     (step 12)
      Parent (Rule).InvalidValues = Parent (Rule).InvalidValues + Last_Value_Added (Rule)
                                                                                     (step 13)
  End For
  For each Rule ∈ NewPartialRules Do
  {Finally, remove from the ValidValues set of each rule all the values that will lead to
   unnecessary construction of useless specialisations at subsequent specialisation steps.}
   Rule.ValidValues = Rule.ValidValues – Parent (Rule).InvalidValues                 (step 14)
  End For
  If w > 1 Then
    Remove from NewPartialRules all duplicate rules.
  Select w best rules from NewPartialRules and insert into PartialRules.             (step 15)
  Remove all rules from NewPartialRules.
End While
Return BestRule
End
```

Figure 3.2 A pseudo-code description of the *Induce_One_Rule ()* procedure of SRI (continued).

Given a training instance list, a class label and the beam width, the procedure *Induce_One_Rule ()* searches for a rule that optimises a given quality criterion by performing a pruned general-to-specific beam search. The search for rules is aimed at covering as many positive instances and as few negative instances as possible. It starts with the most general rule that has no conditions on the left hand side of the rule (step (1) in Figure 3.2) and gradually specialises it by considering all possible specialisations. For each nominal attribute, conditions of the form [$A_i = v_{ij}$] are created for each one of its values. One new rule is then created for each such condition by appending the condition to the current rule (step (3) in Figure 3.2), provided that the attribute does not already appear in it. This prevents the construction of a rule that contains a pair of incompatible conditions such as [Attribute A_1 = yes] $\wedge$ [Attribute A_1 = no]. The version of SRI described here deals only with nominal attributes; a method for handling continuous attributes is given in chapter 4.

Each new rule is evaluated and if the evaluation is better than that of the best rule found previously, the best rule is set to the new rule (step (5) in Figure 3.2). The new rule is then inserted into the *NewPartialRules* list (step (10) in Figure 3.2) unless one of the conditional tests (step (6), (7) or (8) in Figure 3.2) prevents this because it is deemed that no improved rule will be obtained from the new rule. In the latter case, the new rule is regarded as ineffective and additional specialisations will not improve the values for the quality measure. If the new rule is discarded, the last attribute value used to form it is added to the set of invalid attribute values *(InvalidValues)* of its immediate parent, the current rule, so as to ensure that it will be removed from the other specialisations of the same parent rule (step (9) in Figure 3.2). Thus, the *NewPartialRules* list only contains useful rules that can be employed for further specialisation. This process is repeated until there are no remaining rules to be specialised in the *PartialRules* list.

Another test that allows sections of the search space to be pruned away is now applied to each rule in the *NewPartialRules* list after the best rule overall in the current specialisation step is identified. Rules that satisfy the conditional test at step (11) are removed from the *NewPartialRules* list (step (12) in Figure 3.2), again, because they will not lead to improved rules. The last attribute values used to generate these rules are added to the *InvalidValues* set of their parents (step (13) in Figure 3.2). All *InvalidValues* are then deleted from the set of *ValidValues* for each rule in the *NewPartialRules* list (step (14) in Figure 3.2). Invalid values cannot lead to a viable specialisation from any point in the search space that can be reached via identical sets of specialisations and thus excluding them will prevent the unnecessary construction of ineffective specialisations at subsequent specialisation steps.

After eliminating all duplicate rules, the best *w* rules from the *NewPartialRules* list are chosen to replace all rules in the *PartialRules* list (step (15) in Figure 3.2). The comparison between rules is based on the quality measure as defined in the next section. If two rules have an equal value for the quality measure, the simplest rule is favoured. In other words, one with fewer conditions is selected. If both the value of the quality measure and the simplicity of the rules are the same, the most general rule that covers more instances is preferred.

The specialisation process is then repeated until the *PartialRules* list becomes empty (step (2) in Figure 3.2) due to the tests in steps (6), (7), (8) and (11). It should be noted that the *PartialRules* and *NewPartialRules* lists are reused after each iteration. During specialisation, the best rule obtained is stored and returned at the end of the procedure.

3.2.3 Learning System Biases

A system is considered a learning system if it makes an "inductive leap". A system that performs inductive leaps is one that is able to produce knowledge that was not previously known either explicitly or implicitly. In principle, this is impossible. The goal of inductive learning is to infer a general model M from a set of training instances. It is impossible to make such inference on the basis of the training data because nothing in the data can help in deciding whether the inferred model will be true when applied to fresh data. To circumvent such a problem, some assumptions concerning M must be imposed. These assumptions are called the "bias" of the learning system and they provide it with some means for making a guess concerning M. This section discusses various forms of biases that are implemented in the new rule induction algorithm.

3.2.3.1 Employing a general-to-specific beam search

Like most widely-used rule learners, SRI performs a top-down general-to-specific search, thus intentionally being biased towards generality. In this approach, the learning algorithm starts with a general concept description (step (1) in Figure 3.2) and gradually specialises it (step (3) in Figure 3.2) until some stopping growth criterion is met. Because of its bias towards generality, this approach is less susceptible to overfitting of the training data which occurs when a complex rule is learned that too closely mimics the training data. As a result, accurate and simple concept descriptions can be induced in an efficient way. Other methods for searching the hypothesis space are bottom-up (specific-to-general) and bi-directional (Fürnkranz, 1999). These methods have not come into widespread use because they cope poorly with noise.

Since exhaustive search is impractical for large data sets and greedy search is less likely to find the best rule, as with most rule induction algorithms, beam search is also implemented in SRI. This is done by using two rule lists named *PartialRules* and *NewPartialRules*. *PartialRules*, which is the same size as the beam width *w*, stores the *w* best partial rules during the specialisation process. Only the rules in *PartialRules* are considered for further specialisation since they are regarded as the rules that will most likely lead to truly optimal rules. *NewPartialRules* is used to save valid partial rules obtained by specialising the rules in *PartialRules*. Only the *w* best rules from *NewPartialRules* are selected to replace the rules in *PartialRules*. In this way, the search space is considerably reduced.

A beam search of width *w* improves on greedy search by giving the learning system *w* more chances to find the best rule. Beam search is practical because the added complexity grows linearly with *w*. Although increasing the beam width will lead the algorithm to search larger portions of the solution space, it does not mean that the algorithm will generate a better rule set. In fact, the larger the beam width, the more likely it is that the algorithm will find rules that overfit the training data (Quinlan and Cameron-Jones, 1995). In SRI, *w* is a parameter of the algorithm that can be specified by the user. An appropriate empirical range for this parameter is between 1 and 32.

3.2.3.2 Assessing rule quality

Given that the rule induction process could be conceived as a search process, a metric is needed to estimate the quality of rules found in the search space and to direct the search towards the best rule. The rule quality measure is the most influential bias in rule induction. In real-world applications, a typical objective of a learning system is to find rules that optimise a rule quality criterion that takes both training accuracy and rule coverage into account so that the rules learned are both accurate and reliable.

A quality measure must be estimated from the available data. All common measures are based on the number of positive and negative instances covered by a rule. Several different metrics are used in existing algorithms. These include *purity* (utilised in GREEDY (Pagallo and Haussler, 1990) and SWAP-1 (Weiss and Indurkhya, 1991)), *information content* (employed in PRISM (Cendrowska, 1987)), *entropy* (adopted in the original version of the CN2 algorithm (Clark and Niblett, 1989)), the metric applied in RIPPER (Cohen, 1995) and *accuracy* (used in I-REP (Fürnkranz and Widmer, 1994) and PROLOG (Muggleton, 1995)). The problem of the first four measures is that they obtain their optimal values when no negative instances are covered. For example, a rule r_1 that only covers one positive instance scores more highly than a rule r_2 covering 999 positive instances and one negative instance. Also, they do not aim to cover many positive instances. For example, a rule r_3 that covers 100 positive and 10 negative examples is deemed of identical value to another rule r_4 that covers 10,000 positive and 1000 negative examples. As a result, these metrics tend to select very specific rules covering only a small number of instances. This is undesirable since rules covering few instances are unreliable, especially where there is noise in the data. The accuracy of these rules on the training data does not adequately reflect their true predictive accuracy on new test data. The problem of the accuracy measure, as pointed out by Cohen (1995), is that this measure sometimes does not lead to a satisfactory behaviour. For example, it favours a rule r_5 that covers 2000 positive and 1000 negative examples over a rule r_6 that covers 1000 positive and only 1 negative example.

One of the popular metrics that penalises rules with low coverage is the *Laplace accuracy estimate* (used in CN2 (Clark and Boswell, 1991), COVER (Webb, 1993; 1995) and several other algorithms). The Laplace formula is given in chapter 2 (Equation 2.7). The Laplace function trades-off accuracy against generality. In general, it favours rules that cover more positive instances over rules that cover fewer instances and prefers rules with a lower proportion of the cover that is negative over those for which that proportion is higher. Segal (1997) showed that the Laplace estimate has the desirable property of taking into account both accuracy and coverage when estimating rule accuracy. However, it has a problem when learning rules with less than 50% training accuracy. The Laplace estimate does not satisfy the requirement that the rule quality value should rise with increased coverage. Cestnik (1990) also conducted experiments in four medical domains and his results indicated that the Laplace accuracy estimate was often unrealistic, especially in multi-class decision problems. This occurred because of the assumption that underlies Laplace accuracy estimate, namely, that the *a priori* distribution is uniform.

A more general version of the Laplace measure, called the *m-probability-estimate*, has been developed by Cestnik (1990) and is defined as follows:

$$mAccuracy(n_{class}, n_{covered}, k) = \frac{n_{class} + mP_0(C_t)}{n_{covered} + m} \tag{3.1}$$

where $P_0(C_t)$ is the *a priori* probability of the target class and *m* is a domain dependent parameter. The value of *m* is related to the amount of noise in the domain. *m* can be small if little noise is expected and should increase if the amount of noise is substantial. The Laplace estimate can be obtained from the m-probability-estimate when *m* is set to *k*, the total number of classes, and $P_0(C_t)$ is assumed to be uniform. It should be noted that the m-probability-estimate generalises the Laplace estimate so that rules that cover no instances will be evaluated with the *a priori* probability instead of the value 1/*k*, which is more flexible and convenient.

The performance of the seven quality measures mentioned above when used in the SRI algorithm was evaluated empirically. The evaluation was carried out on a large number of data sets and the results showed that the m-probability-estimate outperformed the other measures. Therefore, SRI employs the m-probability-estimate (Equation 3.1) to select the best rule (step (5) in Figure 3.2) and to decide on the best specialisations to retain (step (15) in Figure 3.2) after each specialisation step. In SRI, the *m* value is set to *k* and the *a priori* probability $P_0(C_t)$ is assumed to be equal to the training accuracy of the empty rule that predicts the target class, namely:

$$P_0(C_t) = \frac{P}{N} \tag{3.2}$$

where *P* is the number of instances in the target class C_t and *N* is the total number of instances in the training data set. This version of the Laplace accuracy estimate is a good choice because it has a strong theoretical background (Good, 1965) and it meets the requirements of a good estimation function.

For the examples mentioned above, if it is assumed that *k* equals 2 and that both the total numbers of positive and negative instances are equal, the rule r_1 that only covers one positive instance scores 0.667 and the rule r_2 that covers 999 positive instances and one negative instance scores 0.998. Scores of the rules r_3, r_4, r_5 and r_6 with positive and negative coverages of (100, 10), (10,000, 1000), (2000, 1000) and (1000, 1) are 0.902, 0.909, 0.667 and 0.998 respectively. Therefore, rules r_2, r_4 and r_6 are considered better than rules r_1, r_3 and r_5 respectively, which seems intuitively correct. This indicates that the m-probability-estimate prefers rules that cover many positive instances and few negative instances, thus being biased towards finding general rather than more specific rules.

3.2.3.3 Stopping rule generation

In learning tasks known to involve no noise, a complete and consistent rule set that covers all of the positive and none of the negative training instances is usually preferred. Where there is noise, absolute completeness and consistency becomes unrealistic as this will result in the generation of over-specific rules that overfit the training data. Various studies have indicated that if the training data is noisy, some degree of inconsistency and incompleteness of the rule set is not only acceptable, but also desirable (e.g., Bergadano et al., 1988; Michalski and Kaufman, 1999).

Short rules are often preferred to avoid the problem of overfitting. Such a bias towards short rules is known as overfitting avoidance bias (Schaffer, 1993).

Preference for short rules can be implemented using stopping criteria, which can be viewed as a form of pre-pruning. Pre-pruning and other forms of pruning are discussed in detail in chapter 5. A simple criterion called *minimum purity criterion* is used in FOIL (Quinlan, 1990) to stop generating new rules when the percentage of positive instances covered by the current rule is below a certain purity threshold (usually 80%). A more flexible criterion is employed in SRI (*Rule_Generation_Stopping_Criterion ()* procedure in Figure 3.1). This criterion terminates the induction process for the current class when the accuracy of the current rule is not greater than the accuracy of the empty rule. The rationale for this is that the proportion of positive instances covered by the induced rule should be greater than the proportion of instances of its class with regard to the training data. Another stopping criterion based on the Minimum Description Length (MDL) principle is introduced in chapter 5.

3.2.4 Search-space Pruning Rules

As pointed out earlier, the size of the search space for inducing one rule grows exponentially with both the number of attributes used to describe each instance and the number of values allowed for each attribute. The search space can be efficiently organised by taking advantage of a naturally occurring structure over the hypothesis space that exists for any classification learning problem – a general-to-specific partial ordering of hypotheses (Mitchell, 1997). This structure implies that all specialisations of a rule cover a monotonically decreasing number of positive and negative instances. This organisation property provides a powerful source of constraints on the search performed by the SRI algorithm.

SRI constrains the search space by employing the four pruning rules listed in Table 3.1. These pruning rules remove portions of the search space that do not maximise the quality measure. The effectiveness of these pruning rules depends upon how efficiently they can be implemented and upon the regularity of the data to which the search is applied. The remainder of this section describes the pruning rules in detail.

The pruning rules in Table 3.1 are derived from the following ideas. As the aim of specialisation is to find a rule that maximises the quality measure, further specialisation of a rule can be stopped the moment it becomes clear that additional specialisation will not improve the quality measure for the rule. Furthermore, in order to reduce the number of specialisation steps and thus speed up the learning process, a rule ought to be an improvement over its parent. If this is not the case, the rule should not be further specialised. Finally, as only one solution is sought, further specialisation of a rule can be terminated when it cannot improve on the current best rule.

The first pruning rule (step (6) in Figure 3.2) is used to stop further specialisation when the number of positive instances covered by a rule is below a threshold (*MinPositives*) and thus can be viewed as implementing a form of pre-pruning. Such specialisations are deemed ineffective since the goal is to find rules that cover as many positive instances as possible. In SRI, *MinPositives* is a user specified parameter. The value of this parameter should be kept low, especially in domains that are free of noise, to avoid generating over-simplified rule sets. An appropriate empirical range for this parameter is between 1 and 5. This pruning rule requires almost no additional overhead to employ since the number of positive instances

(1) **If** Covered_Positives (r) $\leq$ MinPositives **Then** Prune (r)

(2) **If** Covered_Negatives (r) – Covered_Negatives (r′) $\leq$ MinNegatives **Then** Prune (r′)

(3) **If** Consistency (r) = 100% **Then** Prune (r)

(4) **If** OptimisticScore (r) $\leq$ Score (BestRule) **Then** Prune (r)

Table 3.1 Search-space pruning rules employed by SRI.

r′ is any specialisation of rule *r* and *Prune (r)* indicates that the children of *r* should not be searched.

covered by a rule must in any case be determined to calculate its accuracy. Section 3.5.1 gives empirical evidence that this pruning rule reduces the learning time of SRI without decreasing the accuracy of its rule sets.

The second pruning rule (step (7) in Figure 3.2) discards descendants of a rule that does not exclude at least some new negative instances. A rule that does not remove any new negative instances is deemed ineffective since either it excludes positive instances only, or it keeps the covered instances unchanged. With greater values of the minimum number of removed negative instances (*MinNegatives*), this pruning rule ensures that each specialisation step changes a rule significantly. As a result, part of the search space can be eliminated in the early stages of the rule specialisation process, which speeds up the execution of the algorithm. In SRI, *MinNegatives* is a parameter of the algorithm that can be specified by the user. An appropriate empirical range for this parameter is between 1 and 5. As is the case for the first pruning rule, no additional overhead is required to employ this pruning rule since the number of negative instances covered by a rule must be determined for the quality measure. Section 3.5.1 shows that this pruning rule improves the quality of the generated rules and speeds up the execution of the algorithm.

The third pruning rule (step (8) in Figure 3.2) avoids expanding rules that have become consistent. The reason is that any further specialisation will only decrease the number of positive instances covered by these rules and therefore yield lower values for the quality measure. It should be noted that the best overall rule will still be returned because the best rule is retained after each specialisation step. Again, no additional overhead is required to use this pruning rule. Section 3.5.1 demonstrates the effectiveness of this pruning rule.

The fourth pruning rule (step (11) in Figure 3.2) removes all specialisations of a rule if its optimistic value of the quality measure cannot improve on the current best rule. The optimistic value can be determined by observing that the specialisation of a rule can only make it become more specific and thereby decrease the number of instances that it covers. As the quality measure is highest when positive cover is maximised and negative cover is minimised, a simple optimistic value is obtained by determining the quality measure of a rule with the same positive cover as the current rule but with a negative cover of zero. If this value is lower than that for the current best rule, specialisation is terminated because none of the rule specialisations can improve on the current best value. Section 3.5.1 confirms that this pruning rule improves the quality of the rule sets of SRI substantially and reduces its learning time.

Clearly, the effectiveness of this pruning rule depends on the value of the current best rule. In general, the larger the best rule value, the greater the search space that can be removed. As a result, the implementation of the fourth pruning rule is delayed until the best overall rule in the current specialisation step is determined. In this way, the performance of this rule is maximised. It should be noted that when all rules at a certain specialisation level satisfy any of the above-mentioned pruning rules, the *PartialRules* set becomes empty. This terminates the search for the best rule (step (2) in Figure 3.2).

The pruning rules discussed above only remove specialisations of rules that are guaranteed not to be a solution. The effect of these rules can be maximised based on the following ideas. If it can be determined that an attribute value used to specialise a certain rule in the search space cannot lead to a solution, then it follows that no solution can also result from the application of such an attribute value to the other specialisations of this rule. This can be justified as follows. Consider the rule $r_1 = A \wedge G \rightarrow$ *target class* where condition G is used to specialise a conjunct A. Let $r_2 = A \wedge B \wedge G \rightarrow$ *target class* be another rule where conditions B and G are successively used to specialise the same conjunct A. If rule r_1 resulting from the application of condition G to conjunct A does not cover any positive instances, then it follows that rule r_1 will not be considered for further specialisation according to the first pruning rule. Furthermore, as conjunct $A \wedge B$ is a specialisation of conjunct A, it must cover fewer instances than are covered by A. As a result, rule r_2 resulting from the application of condition G to conjunct $A \wedge B$ will also not cover any positive instances and consequently rule r_2 should also be excluded from further specialisation.

A similar argument demonstrates that if the application of condition G to conjunct A causes rule r_1 to be discarded according to any of the other pruning rules in Table 3.1, then it follows that rule r_2 should also be discarded as it covers a subset of instances covered by rule r_1.

The above idea can be implemented by maintaining and manipulating for each rule, r, a separate list containing all possible attribute values, *r.ValidValues*, that can be applied in the search space below r. Initially, the list contains all the nominal attribute values. A rule is only specialised by appending to it values, provided that the attributes of such values do not already appear in the rule. Each value that is appended to a rule is removed from its *r.ValidValues* list. When the rule is specialised, the values are examined to determine if any can be pruned away. Any values that can be pruned away are deleted from *r.ValidValues* to prevent the unnecessary construction of ineffective specialisations at subsequent specialisation steps. New rules are then created for each of the attribute values remaining in *r.ValidValues*.

3.2.5 Other Efficiency Improvements

Given the exponential growth in the size of the search space and the iterative nature of rule induction algorithms, it is important that SRI be as efficient as possible in order to be a practical algorithm. The efficiency of SRI is determined by two factors, namely, the number of rules that must be examined and the cost of evaluating each rule. Each of these factors presents an opportunity for improving efficiency. Section 3.2.4 has discussed increasing efficiency by developing pruning rules that can significantly reduce the number of rules the algorithm has to process by pruning away portions of the search space that do not contain the best rule. This section considers techniques for reducing the cost of rule evaluation.

As mentioned previously, SRI induces a rule through the iterative specialisation of the most general rule. The fundamental operation is evaluation of the accuracy of each specialised rule. A straightforward approach to rule evaluation is to scan all the instances in the training set and count the number of positive and negative instances covered by the current rule. These counts are used as input to the quality measure function to compute an estimate of the rule quality. This approach would entail a high computational cost if it was applied repeatedly as outlined.

The efficiency of rule evaluation can be improved by keeping a list of pointers to the instances covered by each rule. Initially, the list contains a pointer to every instance in the training data set, as rule specialisation starts with the most general rule which covers all instances. As the rule is specialised, only the instances covered by the rule are retained and those not covered are removed (step (4) in Figure 3.2). As a result, in subsequent specialisation steps, it is only necessary to check instances in the list, consequently eliminating the need to scan all the training data set. In the later stages of the specialisation process, the size of the instance pointer list is generally small and this significantly reduces the cost of rule evaluation.

It is possible to improve the rule evaluation process further as follows. A rule is said to cover an example if all its conditions hold for that example. Storing the examples covered by each rule makes it possible to evaluate the set of its specialisations by only examining the newly added conditions against the corresponding attribute-values of the instance. This reduces the number of comparisons required for processing each example and thus speeds up the rule evaluation.

3.3 Classification of New Instances

When the rule set generated by SRI is used to classify a new instance, three outcomes are possible:

- Only one rule covers the new instance,
- More than one rule covers the new instance, or
- No rules cover the new instance.

Each case requires a different classification procedure to predict a label for the new instance. In the first case, the class predicted by that rule is simply assigned to the new instance. The conflict between rules in the second case may be due to one or more of the following reasons. First, during the learning process, different forms of biases towards general rules are adopted in SRI to avoid overfitting of the training data. Rules that are too general will be in conflict with others when classifying new instances. It was assumed that this conflict would be dealt with in the classification phase by a resolution method based on a more suitable preference criterion. Second, the generated rule set is usually simplified in a post-processing phase by removing specific rules and deleting superfluous conditions. This is to increase both the predictive accuracy on unseen instances and the comprehensibility of the resulting rule set. Again this will result in an overlapping rule set where two or more rules match the same instances. Finally, as mentioned previously, SRI generates a rule set separately for each class. This has the disadvantage that rule sets for different classes can overlap in the instance space. As a result, a test instance can be assigned to more than one class and some conflict resolution scheme must be applied to determine a prediction.

One possible solution to this problem is to select the rule with the highest value for the quality measure to classify the new instance. This solution overlooks the effect of other rules that might help in determining the best or the most probable decision. Another method that takes into consideration the effect of all matching rules is as follows. When classifying a new instance, each rule is examined and rules that cover the instance and belong to the same class are collected. The proportions of correctly classified instances of such rules are probabilistically summed to form a value for the entire class. For example, if there are two rules r_1 and r_2 that cover an instance a and belong to the same class C_j, then the entire class value for that instance can be determined as follows:

$$Value\ (C_j,\ a) = Value\ (r_1,\ a) + Value\ (r_2,\ a) - Value\ (r_1,\ a) * Value\ (r_2,\ a) \qquad (3.3)$$

where $Value\ (r_1,\ a)$ and $Value\ (r_2,\ a)$ are the proportions of correctly classified instances of rules r_1 and r_2 that cover an instance a respectively.

When all the rules have been scanned, the class with the largest value is taken as the class of the new instance.

These two methods together with the classification methods of AQ15 and CN2 discussed in chapter 2 were applied in SRI. Experimental results on a large range of data sets showed that the second method of the two proposed outperformed the other three methods and was thus adopted in SRI.

In the final case, the new instance belongs to a part of the instance space that is not covered by any rule in the rule set. The rule set generated by SRI is incomplete, that is, it does not cover all the positive instances in the training set, due to the use of the rule stopping criterion discussed in section 3.2.3.3. As a result, it is possible to find instances that do not satisfy any rule in this incomplete rule set and a method for specifying how these instances are to be classified is required. The generally adopted solution is to use a default rule which simply assigns the most frequent class in the entire training set to the new instance to be classified, independent of its attributes.

Another method is to classify the new instance by assigning it to the class of the nearest rule (according to some distance measure) in the rule set. However, because separate metrics are used for nominal and continuous attributes, poor results may arise from combining such different metrics. Consequently, the default rule approach is implemented in SRI.

3.4 Data Sets and Experimental Methodology

All the data sets employed in this research were obtained from the University of California at Irvine (UCI) repository of machine learning databases (Blake and Merz, 1998). These data sets are representative of many different types of classification learning problems and are commonly used to evaluate machine learning algorithms. They differ regarding the number of learning instances that are available, the degree of noise in these instances, the number of classes and the proportion of instances belonging to each class, the number of nominal and continuous-valued attributes used to describe the instances, and the application area from which the data was obtained. A detailed description of these data sets is given in appendix A.

The most widely used schemes for evaluating the performance of a classifier are the "hold-out" scheme and the "cross-validation" scheme (Devijver and Kittler, 1982; Langley and Kibler, 1988; Efron and Tibshirani, 1993). The hold-out scheme randomly partitions a data set into two mutually exclusive subsets, of which one is the training data set, and the other is the test data set. The training data is used for inducing a classifier and the test data is then used for accuracy estimation. The hold-out accuracy estimate is usually taken *n* times for different partitions where *n* ranges from 10 to 50. The accuracy of the classifier is then computed as the average of *n* estimated accuracies and its standard deviation could also be calculated. For large data set, the hold-out method is still preferred due to its efficiency (Brieman et al., 1984). In n-fold cross-validation, the whole data set is randomly divided into *n* approximately equal-sized disjoint subsets (folds). *n* classifiers are constructed and tested, each classifier is built using data from *(n-1)* folds, and tested on the remaining one fold. The accuracy of the classifier estimated by cross-validation is defined as the average of *n* estimated accuracies. The advantage of n-fold cross-validation is that it makes use of all the available data. Usually the parameter *n* is set to ten. It has been found empirically that this choice produces the most reliable estimate of the classifier's true performance on average (Kohavi, 1995a). To achieve a more reliable estimate, n-fold cross validation is usually executed for many times. However, this method is prohibitively expensive on large data sets and is only preferred when the number of instances in the data set is a few hundreds or less in total (Breiman et al., 1984; Kohavi, 1995b).

In the experiments conducted in this thesis, the hold-out approach was used. For large data sets with more than 1000 instances, each set was randomly divided once into a training set with two-thirds of the data and a test set with the remaining one-third. For small data sets with fewer than 1000 instances, the above procedure was repeated ten times, and the results were averaged.

3.5 Empirical Evaluation of SRI

This section presents an empirical evaluation of the pruning rules of the SRI algorithm. Experiments were conducted to explore the relative contribution of each of these rules to the performance of the algorithm. SRI was also compared to the well-known inductive learner C5.0 which is probably the best performing commercially available induction algorithm.

Three criteria were used to evaluate the performance of the tested algorithms, namely, classification accuracy, rule set complexity and execution time. Classification accuracy is generally the most important criterion in induction tasks. It is defined as the percentage of instances from the test set that were correctly classified when the rules developed from the corresponding training set were applied. The complexity of a rule set is measured by the total number of rules or total number of conditions in that rule set. The execution time measures were taken as the total CPU time in seconds and the number of rules evaluated during the search process.

In order to draw reliable conclusions about the behaviour of the learning algorithms, 12 data sets shown in Table 3.2 were considered. As the current implementation of the SRI algorithm is not capable of handling continuous attributes, the chosen data sets were limited to those that had only nominal attributes.

3.5.1 Evaluation of the Search-space Pruning Rules

To evaluate the relative effectiveness of each of the search-space pruning rules given in Table 3.1, the SRI algorithm was employed to find rule sets using first none of the pruning rules and then each individual rule. Results are also reported where all pruning rules are employed. SRI was used with a beam width of 8 and no pre-pruning (the *Rule_Generation_Stopping_Criterion ()* procedure of Figure 3.1 was de-activated). Recall from section 3.2.4 that SRI can employ the two parameters

Data Set Name	No. of Instances	No. of Nominal Attributes	No. of Classes
Breast-cancer	286	9	2
Car	1728	6	4
Chess	3196	36	2
Monk1	556	6	2
Monk2	601	6	2
Monk3	554	6	2
Mushroom	8124	22	2
Promoter	106	57	2
Soybean-large	683	35	19
Splice	3190	61	3
Tic-tac-toe	958	9	2
Vote	435	16	2

Table 3.2 Summary of the data sets used in the experiments (Nominal data).

MinNegatives and *MinPositives* as pre-pruning tests. In the experiments reported here, these two parameters were set to 1 and 2 respectively.

Table 3.3 presents the number of rules explored for each search method. Also given is the percentage by which the number of rules examined is reduced by the addition of each pruning rule. This equals *(x-y)/y*100*, where *x* is the number of rules explored when no pruning rules were applied and *y* is the number of rules considered using the added pruning rule(s). A rule is deemed to have been examined if it is generated at step (3) of the SRI algorithm (Figure 3.2).

As can be seen, the addition of each pruning rule reduced the number of rules that SRI had to process for all the data sets. Further, in many cases the magnitude of this reduction was very large. For example, for the *Promoter* data set, the number of rules to explore dropped by 99.3 % from 513000 to 3750 when all the four rules were applied. It is also notable that the second pruning rule had the largest impact on the number of rules examined. The order of importance of the remaining pruning rules appeared to be as follows: fourth pruning rule (most important), first pruning rule and third pruning rule (least important).

Table 3.4 shows the execution time in CPU seconds for each additional pruning rule. The percentage reduction in the execution time for each search method is also indicated. All execution times were obtained on a Pentium IV computer with a 2.4 GHz processor, 512 MB of memory and the Windows NT 4.0 operating system. For all the data sets, the addition of the pruning rules resulted in a decrease in the computation time.

	SRI with no pruning rules	SRI with rule (1) added		SRI with rule (2) added		SRI with rule (3) added		SRI with rule (4) added		SRI with all pruning rules	
Data Set Name	Number	Number	% Red.	Number	% Red.	Number	% Red.	Number	% Red.	Number	% Red.
Breast-cancer	72900	8615	88.2	16430	77.5	62574	14.2	23392	67.9	5802	92.0
Car	43917	18939	56.9	37243	15.2	42281	3.7	29735	32.3	17022	61.2
Chess	353607	228716	35.3	14864	95.8	222849	37.0	96357	72.8	19402	94.5
Monk1	7405	4727	36.2	4686	36.7	6695	9.6	3974	46.3	3310	55.3
Monk2	31216	10217	67.3	21531	31.0	27958	10.4	23544	24.6	9314	70.2
Monk3	4163	2832	32.0	1937	53.5	3387	18.6	1217	70.8	1102	73.5
Mushroom	277821	64402	76.8	4473	98.4	160204	42.3	6537	97.6	2765	99.0
Promoter	513000	18317	96.4	11733	97.7	246155	52.0	12467	97.6	3750	99.3
Soybean-large	690783	169881	75.4	89001	87.1	582159	15.7	46438	93.3	11463	98.3
Splice	5557095	784013	85.9	335862	94.0	3201674	42.4	441421	92.1	273236	95.1
Tic-tac-toe	21384	13126	38.6	10967	48.7	16484	22.9	9714	54.6	7468	65.1
Vote	64416	11022	82.9	8688	86.5	45048	30.1	11792	81.7	2592	96.0

Table 3.3 Total number of rules explored for each search method.

	SRI with no pruning rules	SRI with rule (1) added		SRI with rule (2) added		SRI with rule (3) added		SRI with rule (4) added		SRI with all pruning rules	
Data Set Name	Time (s)	Time (s)	% Red.	Time (s)	% Red.	Time (s)	% Red.	Time (s)	% Red.	Time (s)	% Red.
Breast-cancer	22	5	77.3	9	59.1	17	22.7	10	54.5	3	86.4
Car	40	17	57.5	29	27.5	29	27.5	24	40.0	16	60.0
Chess	302	235	22.2	37	87.7	208	31.1	115	61.9	46	84.8
Monk1	3	2	33.3	2	33.3	3	0.0	2	33.3	2	33.3
Monk2	10	4	60.0	8	20.0	8	20.0	8	20.0	4	60.0
Monk3	1	1	0.0	1	0.0	1	0.0	1	0.0	1	0.0
Mushroom	431	185	57.1	56	87.0	216	49.9	85	80.3	47	89.1
Promoter	402	32	92.0	30	92.5	286	28.9	15	96.3	7	98.3
Soybean-large	474	160	66.2	120	74.7	414	12.7	54	88.6	27	94.3
Splice	6210	1575	74.6	1009	83.8	4174	32.8	1199	80.7	989	84.1
Tic-tac-toe	10	8	20.0	8	20.0	9	10.0	6	40.0	5	50.0
Vote	19	6	68.4	5	73.7	14	26.3	6	68.4	2	89.5

Table 3.4 Execution times taken for each search method.

It is worth noting the computation times for the *Mushroom, Promoter* and *Soybean-large* data sets.

Table 3.5 shows the complexity of the rule sets generated with each of the pruning rules. The percentage reduction in the number of conditions obtained with each experiment is also given. Table 3.6 gives the classification accuracies obtained with each of the search-space pruning rules. The percentage increase in the classification accuracy achieved with each condition is also given. A number of results are notable. First, the application of the fourth pruning rule resulted in a minor increase in the complexity of the rule sets. Second, the other pruning rules caused a large reduction in the complexity of the rule sets and an improved classification accuracy in most cases. For example, for the *Car* data set the number of conditions dropped from 423 to 311, while the accuracy increased from 93.2 % to 95.3 % when all the pruning rules were employed.

3.5.2 Comparison with C5.0

SRI was compared to C5.0 on the twelve data sets listed in Table 3.2. C5.0 has a facility to generate a set of pruned production rules from a decision tree. SRI and C5.0 each has a number of parameters whose values determine the quality of their induced rule sets. For SRI, the beam width was set to 4 and the *Rule_Generation_Stopping_Criterion ()* procedure of Figure 3.1 was activated. The two parameters MinNegatives and MinPositives were set to 1 and 2 respectively. For C5.0, the default settings were used.

	SRI with no pruning rules	SRI with rule (1) added		SRI with rule (2) added		SRI with rule (3) added		SRI with rule (4) added		SRI with all pruning rules	
Data Set Name	Number	Number	% Red.	Number	% Red.	Number	% Red.	Number	% Red.	Number	% Red.
Breast-cancer	108	111	-2.8	103	4.6	111	-2.8	121	-12.0	109	-0.9
Car	423	311	26.5	423	0.0	423	0.0	431	-1.9	311	26.5
Chess	62	83	-33.9	49	21.0	50	19.4	88	-41.9	79	-27.4
Monk1	61	61	0.0	61	0.0	61	0.0	61	0.0	61	0.0
Monk2	375	204	45.6	335	10.7	349	6.9	351	6.4	201	46.4
Monk3	23	23	0.0	23	0.0	23	0.0	23	0.0	23	0.0
Mushroom	26	26	0.0	26	0.0	27	-3.8	27	-3.8	26	0.0
Promoter	19	14	26.3	18	5.3	18	5.3	18	5.3	14	26.3
Soybean-large	113	104	8.0	113	0.0	113	0.0	118	-4.4	105	7.1
Splice	218	300	-37.6	221	-1.4	220	-0.9	305	-39.9	249	-14.2
Tic-tac-toe	82	77	6.1	89	-8.5	84	-2.4	84	-2.4	78	4.9
Vote	56	39	30.4	57	-1.8	54	3.6	54	3.6	43	23.2

Table 3.5 Total number of conditions generated for each search method.

	SRI with no pruning rules	SRI with rule (1) added		SRI with rule (2) added		SRI with rule (3) added		SRI with rule (4) added		SRI with all pruning rules	
Data Set Name	Acc. (%)	Acc. (%)	% Incr.	Acc. (%)	% Incr.	Acc. (%)	% Incr.	Acc. (%)	% Incr.	Acc. (%)	% Incr.
Breast-cancer	75.8	70.5	-6.9	71.6	-5.6	77.9	2.8	77.9	2.8	72.6	-4.2
Car	93.2	95.3	2.2	93.2	0.0	93.2	0.0	93.6	0.4	95.3	2.2
Chess	95.7	99.1	3.5	96.2	0.6	96.3	0.7	99.2	3.6	98.7	3.1
Monk1	100.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
Monk2	67.5	63.8	-5.5	65.7	-2.6	66.9	-0.9	66.9	-0.9	65.4	-3.1
Monk3	100.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
Mushroom	99.7	100.0	0.3	99.7	0.0	100.0	0.3	100.0	0.3	100.0	0.3
Promoter	71.4	74.3	4.0	71.4	0.0	71.4	0.0	71.4	0.0	74.3	4.0
Soybean-large	93.0	92.1	-0.9	91.7	-1.4	93.0	0.0	93.0	0.0	91.2	-1.9
Splice	88.0	87.7	-0.3	89.5	1.7	88.0	0.0	86.7	-1.5	89.0	1.2
Tic-tac-toe	97.8	97.2	-0.6	98.1	0.3	98.8	1.0	98.8	1.0	98.1	0.3
Vote	97.8	96.3	-1.5	95.6	-2.3	96.3	-1.5	96.3	-1.5	95.6	-2.3

Table 3.6 Classification accuracies obtained with each search method.

Table 3.7 presents the results for each algorithm on each data set. In each case, the accuracy on the test data and the complexity of the resulting rule sets are given. The number of rules was taken as a measure of the complexity of the rule set. A complexity of one was assigned to the default rule.

It is clear from Table 3.7 that the accuracy obtained by SRI was in total higher than that of C5.0. In addition, on 6 of the 12 data sets, SRI achieved higher accuracy than C5.0. On three data sets, *Monk1*, *Monk2*, *Monk3*, both algorithms had the same accuracy. On the remaining three data sets, SRI was inferior to C5.0. However, with SRI, the number of rules was higher in 9 data sets. The smaller number of rules produced by C5.0 can be attributed to the rule set (decision tree) pruning techniques employed. Much simpler and also more accurate rules are obtained when the pruning techniques of chapter 5 are incorporated into the SRI algorithm.

3.6 Summary

This chapter has presented a new rule induction algorithm for classification learning which includes a new heuristic search technique and search-space pruning rules. The search technique performs a pruned general-to-specific beam search and employs other search biases to traverse the large search spaces involved in many machine learning problems. These strategies enable efficient processing of such search spaces and induction of accurate and simple rules from a set of data. The search technique also uses several novel pruning rules that take advantage of the special structure of the search space to eliminate portions that do not contain a solution. Experimental results have demonstrated that the new pruning rules can significantly reduce the number of

	C5.0		SRI	
Data Set Name	Acc. (%)	No. of Rules	Acc. (%)	No. of Rules
Breast-cancer	**75.8**	17	73.7	**15**
Car	91.8	**58**	**94.3**	78
Chess	97.2	**21**	**98.8**	36
Monk1	100.0	**17**	100.0	23
Monk2	65.7	**1**	65.7	19
Monk3	100.0	**6**	100.0	13
Mushroom	99.8	**10**	**100.0**	15
Promoter	74.3	**7**	**76.3**	9
Soybean-large	**93.4**	32	90.4	**27**
Splice	**92.7**	**60**	91.1	74
Tic-tac-toe	92.2	34	**97.8**	**21**
Vote	97.0	**5**	**97.8**	10
Total	1079.9	**268**	**1085.7**	340

Table 3.7 Results for C5.0 and SRI.

rules considered during the rule learning process and greatly increase the efficiency of the algorithm.

The algorithm presented in this chapter is limited to nominal attributes. A method for learning directly from continuously valued data is considered in the next chapter.

CHAPTER 4

DISCRETISATION OF CONTINUOUS-VALUED ATTRIBUTES FOR LEARNING CLASSIFICATION RULES

4.1 Motivation

Since most real-world applications of classification learning involve continuous-valued attributes, properly addressing the discretisation process is an important problem to be solved in developing generally applicable methods for data mining. Discretisation of a continuous attribute involves finding an appropriate set of cutting points for that continuous attribute. "Appropriate" means that the information in the data for discriminating the classes is not lost. Discretisation can improve the accuracy of the learned model. The reason is that discretisation produces a concise representation of continuous attributes and this helps learning algorithms to capture the relationship between different attributes.

The usual approach to discretisation of continuous-valued attributes is to perform this discretisation off-line, prior to the learning process (Ting, 1994; Wu, 1996; Ho and Scott, 1997; Jun et al., 1997; Kontkanen et al., 1997; Liu and Setiono, 1997; Zighed et al., 1997; Frank and Witten, 1998; Perner and Trautzsch, 1998; Wang and Liu, 1998; An and Cercone, 1999; Peng, 2004). First, all continuous attributes in the data are discretised to obtain a discrete data set. Then learning algorithms are applied to this discretised data set. Off-line discretisation is useful for several reasons. First, it enables learning algorithms that can inherently only handle nominal attributes to process continuous-valued attributes in a consistent manner. Furthermore, it can effectively speed up inductive learning. It has been shown that, for some learning algorithms, efficient discretisation as a pre-processing operation resulted in significant speed increases (Catlett, 1991b). Finally, off-line discretisation can produce simpler classifiers than those learned from the raw continuous data. Discretisation produces inherent generalisation by grouping data into several ranges, representing it in a more general way. Also, by restricting the search space that the learning algorithm can explore the likelihood of overfitting the training data is reduced and hence the chance of finding a less complex classifier is increased.

Off-line discretisation, however, suffers from at least two problems. First, the discretisation process is independent of the learning process and therefore does not comply with the demands of the learning algorithm. Second, independent discretisation of attributes may destroy the higher-order correlation between them (Ventura, 1995). Higher-order correlation between attributes means that an attribute by itself may not directly correlate with the output class, but in combination with one or more of the other attributes, it may have a very high correlation with the output class.

As a result, a different approach is proposed and a method of learning directly from continuous-valued attributes is developed. By handling continuous-valued attributes during induction, the bias of the induction system can be taken into account and interactions among different attributes considered. Therefore, this approach should yield even greater improvements in the performance of the learning algorithm than achieved with off-line discretisation. However, the execution speed of an induction algorithm incorporating on-line discretisation would increase since the discretisation process may need to be repeated many times within the inductive process. The main purpose of this chapter is to develop an effective and a computationally efficient on-line discretisation method for use in rule induction algorithms.

This chapter is organised as follows. First, a review of existing discretisation approaches, their defining characteristics and their strengths and limitations is given in the context of inductive learning. This is followed by a detailed description of the new discretisation method. Then, an empirical evaluation of the method is presented. Finally, a summary of the findings of the chapter is given.

4.2 Survey of Methods for Discretisation of Continuous-valued Attributes

4.2.1 Overview

Current discretisation methods can be divided in four ways, namely, supervised vs. unsupervised, multivariate vs. univariate, off-line vs. on-line and parametric vs. non-parametric.

Supervised vs. Unsupervised

Supervised discretisation techniques take the relationship between the class label and the continuous attribute to be discretised into account during the discretisation process. On the other hand, unsupervised methods do not consider the class label, and hence are "class-blind". Class-blind methods are typically used in unsupervised learning where there are no assigned class labels, whereas supervised discretisation methods are naturally adopted when supervised learning are used. Comparative studies have shown that supervised discretisation generally performs better than unsupervised discretisation (Ching et al., 1995; Dougherty, et al., 1995; Liu et al., 2002).

Multivariate vs. Univariate

Many data sets have more than one continuous attribute. Multivariate discretisation refers to the discretisation of several continuous attributes simultaneously. This approach can take interactions between different attributes into account. The univariate method discretises one attribute at a time, with the discretisation of the next continuous attribute beginning after the discretisation of the current attribute has finished. The drawback of this method is that once a continuous attribute has been discretised, this discretisation cannot be revoked. Most of the current discretisation methods are univariate.

Off-line vs. On-line

As previously mentioned, if discretisation is performed while the data is pre-processed and before learning has begun, it is regarded as off-line. The off-line method discretises the data only once and therefore it is efficient. With an on-line discretisation approach, discretisation is an integral component of the learning algorithm itself. For instance, the discretisation approach of the decision-tree induction algorithm C4.5 (Quinlan, 1993) falls into this category. At each node of the tree, the continuous attribute is binarised based only on instances associated with this node and competes with other attributes during the attribute selection process. Consequently, one continuous attribute can be discretised several times at different levels of the tree.

Parametric vs. Non-parametric

Parametric discretisation methods require the user to specify some parameters. The most important parameter is the number of intervals into which a continuous attribute can be partitioned. If the method can be executed without this kind of user intervention, it is non-parametric. The number of intervals has an important effect on learning performance and classification accuracy (Ching el al., 1995). For inductive learning, a large number of intervals is not always desired because the performance of many inductive learners deteriorates dramatically with large numbers of discrete intervals. After all, the reason for discretisation is to reduce the number of possible values an attribute can take. Therefore, for the purpose of supervised learning, the optimal number of intervals can be regarded as the smallest number that does not significantly weaken the interdependency between attribute values and classes (Kurgan and Cios, 2001).

4.2.2 Discretisation Methods

A typical unsupervised discretisation method is *equal-width interval* discretisation (Wong and Chiu, 1987). This is perhaps the simplest discretisation procedure. It simply involves dividing the range of a continuous variable into l equal intervals, where l is a user-defined parameter. Since the equal-width approach considers neither the distribution of the values of the continuous attribute, nor the dependency between the class label and the continuous attribute, it is likely that classification information will be lost as a result of combining values that are closely associated with different classes into the same interval. Furthermore, the number of intervals has a strong impact on performance. If too many intervals are specified, the learned model will be complex. If too few intervals are specified, information that can be used to distinguish instances will be lost.

A related method, *equal-frequency intervals*, divides the range into l intervals each of which contains the same number of instances (Wong and Chiu, 1987). A variation of the equal-frequency approach, called *maximum marginal entropy*, adjusts the interval boundaries using an entropy measure so as to reduce the amount of information lost due to discretisation (Wong and Chiu, 1987; Chmielewski and Grzymala-Busse, 1994).

ChiMerge (Kerber, 1992) and *StatDisc* (Richeldi and Rossotto, 1995) are two supervised parametric discretisation methods. Both approaches employ a bottom-up merging process, where intervals are repeatedly merged until a termination condition is met. However, StatDisc is more general than ChiMerge in that it considers merging up to a user-defined number of intervals at a time, rather than just two adjacent intervals as in ChiMerge. They are, also, different in the interval initialisation scheme and in the statistical measure which they employ. ChiMerge is initialised by putting each instance into its own interval, and uses the χ^2 statistic to decide whether two adjacent intervals should be merged. StatDisc is initialised by grouping adjacent instances labelled with the same class into the same interval, and uses the Φ statistic to measure the association of adjacent intervals. Compared with class-blind techniques, these two methods are more robust. The main drawback is that both methods require users to specify the significance level which is employed to control the merge granularity.

A number of information-theory-based discretisation methods have been developed. Among these are methods belonging to the entropy-based discretisation approach (Catlett, 1991b; Fayyad and Irani, 1993; Phahringer, 1995a) and the distance-based approach (Cerquides and Lopez de Mantaras, 1997). They are mostly inspired by Quinlan's decision tree induction algorithms ID3 and C4.5 (1986; 1993).

Catlett (1991b) proposed a supervised discretisation method called *D2* as a means of reducing the learning time of the ID3 algorithm when continuous attributes are encountered. In D2 the discretisation is carried out off-line at the pre-processing stage. D2 adopts a greedy top-down approach. To find the set of intervals, the training instances are first sorted on the values of the continuous attribute in question. The method then evaluates all candidate cut points and selects the one that maximises the information gain. The training set is then split into two subsets by the cut point value. Subsequent cut points are selected by recursively applying the same binary discretisation method to each of the newly generated subsets until one of four stopping conditions is satisfied: the number of instances in an interval is sufficiently small, the number of cut points produced for any attribute reaches a maximum limit, the gain for all intervals is equal or all instances in an interval are in the same class. Since the instances are not reordered, they need not be re-sorted, and this is the reason for reduced learning times. If u cut points are found, the continuous attribute is mapped to a discrete attribute with u+1 values, one for each interval. One of the main problems with this discretisation method is that it is rather computationally expensive. It must be evaluated N-1 times for each attribute (assuming that the N examples have distinct values). Typically N is very large.

Fayyad and Irani (1993) introduced a similar method to that of Catlett, but developed an elegant test based on the Minimum Description Length (MDL) principle to determine a stopping criterion for the recursive discretisation strategy. Moreover, the method takes advantage of the fact (Fayyad, 1992) that the optimal cutting points when discretising a continuous attribute using an average class entropy evaluation function can only be selected from a set called the boundary points. This can be used to improve the efficiency of the discretisation method, as the latter needs only to examine the boundary points of each continuous attribute rather than all its distinct values.

Phahringer (1995a) proposed a two-step discretisation method called *MDL-DISC*. First, a simplified version of Catlett's D2 method is used to select a set of promising split points. Second, this set is searched thoroughly by a best-first search to determine a good discretisation according to the Minimum Description Length (MDL) estimate. Special provisions (for example, escape to a class-blind method) are made for the degenerate case where just a single interval results from the discretisation.

Holte (1993) described a simple example of a supervised discretisation method called *1R Discretizer*. This method first sorts the values of a continuous attribute in ascending order and then puts instances having equal values or having the same class label into one interval. Adjacent intervals can then be merged if they share the same majority class label. To avoid too many intervals being generated, each interval must include at least a pre-specified number of instances.

The above-mentioned methods are heuristic and, therefore, they cannot guarantee finding the optimal discretisation. However, their efficiency makes them attractive choices in practical applications. In recent years, several optimisation techniques for discretisation of continuous-valued attributes have been developed. Maass (1994) was the first to suggest a dynamic programming method which finds the minimum partitioning of a continuous attribute with respect to the training set error evaluation function in polynomial time. This method was implemented as part of the *T2* induction algorithm (Auer et al., 1995) which induces one- or two-level decision trees. Fulton et al. (1995) followed by introducing a quadratic-time general method which works for a class of evaluation functions in two-class learning tasks. They also proposed a linear-time method, which works for a narrower range of evaluation measures than the quadratic-time method. Later, Birkendorf (1997) devised linear-time methods for the multi-class case. Elomaa and Rousu (1999a) and Rousu (2001) extended the work of Fulton et al. by introducing a pruning technique to improve dynamic programming search and by operating on example intervals instead of individual examples. These enhancements resulted in a general and efficient method. Following a similar approach, Cai (2001) proposed an efficient discretisation method using an evaluation function based on the Minimum Description Length (MDL) principle. The optimal number of intervals is obtained by examining only the search space of boundary points of each continuous attribute and selecting those points that optimise the evaluation metric.

Empirical evaluations of the different methods of discretising continuous attributes have been conducted. Dougherty et al. (1995) compared the equal-width approach, the 1R Discretizer proposed by Holte, the entropy-based discretisation method by Fayyad and Irani and the binary discretisation method of C4.5 using two induction algorithms, C4.5 (Quinlan, 1993) and a Naïve-Bayesian classifier (Good, 1965). They reported in their study that the entropy-based discretisation method was the best. This method was compared to the "optimal" discretisation method of Maass in a study presented by Kohavi and Sahami (1996). These discretisation methods were also evaluated on C4.5 and Naïve-Bayesian classifiers on data sets from the UCI repository. Results showed that the entropy-based method generally outperformed the "optimal" discretisation method.

Quinlan (1996b) conducted experiments similar to those described by Dougherty et al. on twenty databases from the UCI repository that involve continuous attributes, either alone or in combination with nominal attributes. In this study, a new version of C4.5, which modifies the formation and evaluation of tests on continuous attributes, is employed. The results showed that the binary discretisation method of the new C4.5 algorithm was superior to the entropy-based method of Fayyad and Irani. Comparisons with the T2 induction system, which employs the dynamic discretisation method of Maass (1994), again confirmed the superiority of C4.5. However, T2 trees were much smaller than those found by C4.5 – less than half the size on average.

Trautzsch and Perner (1996) evaluated three methods of discretisation, namely, the binary discretisation method of C4.5, the entropy-based method of Fayyad and Irani and the ChiMerge method. The results showed that neither of the latter two discretisation methods outperformed the method used in C4.5 significantly.

Elomaa and Rousu (1996b) compared three different discretisation strategies and examined their impact on decision tree learning. The contrasted strategies were the C4.5 binarisation strategy, an implementation of the entropy-based strategy and the "optimal" splitting strategy. Experiments on a large number of commonly used data sets showed that the entropy-based and "optimal" strategies did not give any higher prediction accuracy than the binarisation strategy. This confirms the results of Quinlan (1996b). Also, the entropy-based and "optimal" strategies were slower than binarisation; the entropy-based method took on average twice the time of the binarisation method in decision tree learning and the "optimal" splitting method further doubled the average time of the entropy-based method.

4.3 Proposed Discretisation Method

The simplest method for discretising continuous attributes during learning is to determine for each attribute A_i all the distinct values that occur in the instances covered by the current partial rule, create conditions of the form ($A_i \leq v_{ij}$) and ($Ai > v_{ij}$) for each value v_{ij} and evaluate all the rules that can be formed by adding such conditions to the current rule. All such rules and those resulting from specialising the current rule with nominal conditions can then be compared according to the evaluation metric to select a size-limited set of best rules at each specialisation step. One problem with this method is that a continuous attribute with numerous distinct values will have an advantage over a nominal attribute and also over other continuous attributes that have fewer distinct values as a large number of rules generated with such an attribute might be selected within the size-limited set of rules considered for further specialisations. These rules typically have very small differences and thus the search can be concentrated in a limited region of the rule space with a possible degradation of performance. Another problem with this simplistic method is that it is very expensive computationally, especially for data sets involving continuous attributes with a large number of values.

The efficiency of this method can be improved by only examining the boundary points of each continuous attribute rather than all of its individual values. A boundary point occurs if, in the sorted list of values for attribute A_i, v_{i1} and v_{i2} are adjacent and they are associated with different classes. However, boundary points may change from point to point in the rule space as their determination depends on the frequencies of classes as well as the distribution of values of each continuous attribute in the set of instances covered by each rule. Recalculating the boundary points for each rule is computationally expensive. Moreover, the change of boundary points makes it difficult to apply the pruning rules discussed in chapter 3 (section 3.2.4), which have proved useful for discarding large portions of the search space.

As mentioned earlier, a practical approach effective in the context of decision trees is the binary discretisation method. This involves partitioning the range of values for a continuous attribute into only two intervals at any node in the search space. This section proposes a new discretisation method also based on binary discretisation but suitable for use in rule learning systems. The implementation of this method in the SRI rule induction algorithm presented in chapter 3 and the possibility of applying the search-space pruning rules of SRI are also discussed.

4.3.1 The Basic Method

As mentioned in chapter 3, the SRI rule induction algorithm can only handle nominal attributes. The search procedure of SRI can be easily modified to deal with continuous attributes by changing only the *Induce–One–Rule ()* procedure (Figure 3.2), leaving the *Induce-Rules ()* procedure (Figure 3.1) unchanged. A pseudo-code of the modified *Induce–One–Rule ()* procedure is shown in Figure 4.1.

This procedure starts with a check to see if there are continuous attributes in the given training instance list (step (1) in Figure 4.1). If this is the case, a separate list called *AttributeAndLabel* list is created for each continuous attribute in the current instance list of the rule to be specialised. An entry in an *AttributeAndLabel* list consists of an attribute value and a class label. A pass is made over the current instance list, distributing values of the continuous attributes for each instance across all the lists. Each attribute value is also tagged with the corresponding class label. This allows for independent processing of each continuous attribute and thus improves efficiency.

A continuous-valued attribute is typically discretised during rule generation by partitioning its range into two intervals. A threshold value, t_{ij}, for the continuous-valued attribute A_i is determined and two conditions of the form ($A_i \leq t_{ij}$) and ($A_i > t_{ij}$) are created. The next section discusses how the threshold value can be calculated for each continuous attribute. A rule is formed by adding the condition ($A_i \leq t_{ij}$) to the current rule. The accuracy of the new rule is computed and compared with that of the best rule found so far and the one with the largest value is remembered. The new rule is then added to the *NewPartialRules* list if it satisfies the conditional test at step (5) in Figure 4.1. The same is done for the condition ($A_i > t_{ij}$). SRI determines at most one

```
Procedure Induce_One_Rule (Instances, ClassLabel, $w$)
PartialRules = NewPartialRules = $\emptyset$
BestRule = most general rule (the rule with conditions)
PartialRules = PartialRules $\cup$ {BestRule}
While PartialRules $\neq \emptyset$ Do
  For each Rule $\in$ PartialRules Do
  {First, generate all specialisations of the current rule and save useful ones according to the
   conditional tests in steps (2) and (5). Determine all the InvalidValues for nominal attributes
   only (step 3).}
    If there are continuous attributes Then                                          (step 1)
      Create a separate AttributeAndLabel list for every continuous attribute in the instance list
      of the current rule.
    For each attribute $A_i$ Do
      If $A_i$ is a nominal attribute Then
        If $A_i$ does not exist in Rule Then
          For each valid value $v_{ij}$ of $A_i \in$ Rule.ValidValues Do
            NewRule = Rule $\wedge$ [$A_i = v_{ij}$]
            NewRule.Instances = Covered_Instances (Rule.Instances, $v_{ij}$)
            If NewRule.Score > BestRule.Score Then
              BestRule = NewRule
            If (Covered_Positives (NewRule) $\leq$ MinPositives OR
              Covered_Negatives (Rule) – Covered_Negatives (NewRule) $\leq$ MinNegatives OR
              Consistency (NewRule) = 100%) Then                                     (step 2)
                Parent (NewRule).InvalidValues = Parent (NewRule).InvalidValues + {$v_{ij}$}  (step 3)
            Else
              NewPartialRules = NewPartialRules $\cup$ {NewRule}
          End For
```

Figure 4.1 A pseudo-code description of the modified *Induce_One_Rule ()* procedure of SRI.

PartialRules: a list of rules to be specialised and ***NewPartialRules***: a new list of rules to be used for further specialisations.

```
        Else If $A_i$ is a continuous attribute Then
          If $A_i$ does not have both bounds set in Rule Then                              (step 4)
            ♦ Sort the attribute list for $A_i$ from small to large to get an ordered list of attribute
              values, $\{v_{i1}, v_{i2}, ..., v_{ij}, ..., v_{id}\}$.
            ♦ Compute the Information Gain for each midpoint value, $(v_{ij} + v_{i(j+1)})/2$, where $v_{ij}$, $v_{i(j+1)}$
              are two distinct values of the attribute $A_i$.
            ♦ Find the best threshold value, $t_{ij}$, which has the highest Information Gain.
            If the upper bound does not exist in Rule Then
              NewRule = Rule ∧ [$A_i \le t_{ij}$]
              NewRule.Instances = Covered_Instances (Rule.Instances, $t_{ij}$)
              If NewRule.Score > BestRule.Score Then
                BestRule = NewRule
              If (Covered_Positives (NewRule) > MinPositives AND
                Covered_Negatives (Rule) – Covered_Negatives (NewRule) > MinNegatives AND
                Consistency (NewRule) < 100%) Then                                          (step 5)
                  NewPartialRules = NewPartialRules ∪ {NewRule}
            If the lower bound does not exist in Rule Then
              Do the same for [$A_i > t_{ij}$] interval.
      End For
    End For
    Empty PartialRules.
    For each Rule ∈ NewPartialRules Do
    {Next, delete partial rules that cannot lead to an improved rules according to the conditional test
      in step (6). Determine all the InvalidValues for nominal attributes only (step 8).}
      If Rule.OptimisticScore ≤ BestRule.Score Then                                         (step 6)
        NewPartialRules = NewPartialRules – {Rule}                                          (step 7)
        If Last_Value_Added (Rule) is a nominal attribute value Then
          Parent (Rule).InvalidValues = Parent (Rule).InvalidValues + Last_Value_Added (Rule)
                                                                                            (step 8)
    End For
```

Figure 4.1 A pseudo-code description of the modified *Induce_One_Rule ()* procedure of SRI (continued).

```
  For each Rule ∈ NewPartialRules Do
  {Finally, remove from the ValidValues set of each rule all the values that will lead to
   unnecessary construction of useless specialisations at subsequent specialisation steps.}
     Rule.ValidValues = Rule.ValidValues – Parent (Rule).InvalidValues          (step 9)
  End For
  If $w > 1$ Then
    Remove from NewPartialRules all duplicate rules.
  Select $w$ best rules from NewPartialRules and insert into PartialRules.
  Remove all rules from PartialRules.
End While
Return BestRule
End
```

Figure 4.1 A pseudo-code description of the modified *Induce_One_Rule ()* procedure of SRI (continued).

lower and one upper bound for a continuous attribute (step (4) in Figure 4.1). As soon as both bounds have been determined, no further specialisations are considered that involve that particular attribute. This preference for simple rules was empirically found more effective than continuing the specialisation process and making the intervals more specific. This speeds up the learning process and increases the comprehensibility of the learned rule set.

Note that rules that do not satisfy the conditional test in step (5) are only prevented from being added to the *NewPartialRules* list. Unlike for nominal attributes, the last intervals used to specialise these rules are not added to the set of invalid attribute values (*InvalidValues*). Therefore, the full benefits of the search-space pruning rules as explained in chapter 3 (section 3.2.4) is not realised. The reason for this is that intervals (boundary points) for continuous attributes may change at any location in the rule space as mentioned before. Consequently, if the pruning rules determined that such intervals when used to specialise a certain rule in the search space cannot lead to a solution, then it cannot be concluded that rules resulting from the application of these same intervals to the other specialisations of the current rule are also not solutions. Consideration of the attributes of such intervals for further specialisations in this context may result in better rules.

Rules that satisfy the conditional test in step (6) are removed from the *NewPartialRules* list (step (7) in Figure 4.1). Again, only the invalid values of nominal attributes are appended to the *InvalidValues* set (step (8) in Figure 4.1). All *InvalidValues* of nominal attributes are then removed from the set of *ValidValues* for each rule in the *NewPartialRules* list (step (9) in Figure 4.1).

4.3.2 Best Threshold Determination

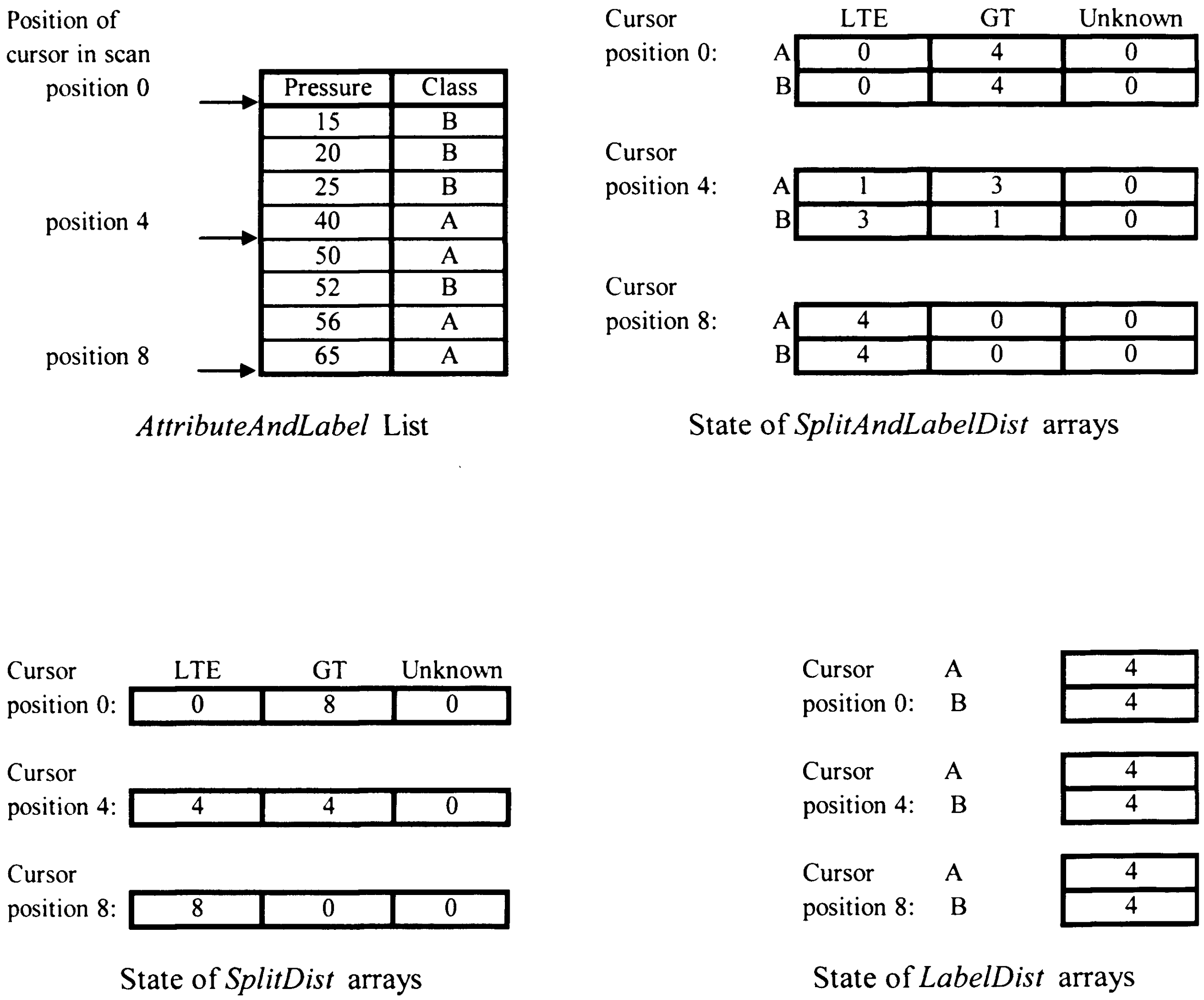
For each continuous-valued attribute A_i, the best threshold is selected from its range of values by evaluating every candidate threshold in the range of values. The *AttributeAndLabel* list for attribute A_i is first sorted from small to large based on the attribute values. Then, the midpoint between each successive pair of distinct values in the sorted list is taken as a potential threshold. Midpoints are chosen because any value between every two consecutive attribute values will divide the set of instances into the same two subsets. While splitting attribute A_i, the goal is to determine the threshold that best divides the training instances belonging to that attribute. The value of a threshold depends upon how well it separates the classes. The information gain criterion (Equation (2.4) in chapter 2) is used to evaluate the appropriateness of each threshold. An efficient way of performing this evaluation is discussed in the next section. Finally, the threshold with the highest information gain is used to split attribute A_i.

4.3.3 Data Structures

To compute the best threshold for each continuous-valued attribute efficiently, three array structures are attached to each rule that is under consideration for specialising. These arrays are used to record the class distribution of the attribute's instances for a given rule. The first array, called *SplitAndLabelDist*, is a two-dimension array in which the rows correspond to labels and the columns correspond to split type.

There are three types of splits denoted as *LessThanOrEqual (LTE)*, *GreaterThan (GT)* and *Unknown*. The *LTE* and *GT* columns record the class distribution for instances that satisfy tests of the form ($A_i \leq t_{ij}$) and ($A_i > t_{ij}$) respectively, whereas the *Unknown* column stores the distribution for those that contain missing values. The other two arrays, denoted as *SplitDist* and *LabelDist*, are derived from the first array to facilitate the computation of the best split. Each element in the *SplitDist* array corresponds to the total number of instances for each split type and is obtained by summing up a certain split column. Also, each element in the *LabelDist* array corresponds to the total number of instances for each label and is obtained by summing up a certain labelled row.

To determine the best threshold for a continuous-valued attribute at a given rule, the *LTE* column of the *SplitAndLabelDist* array is initialised to zero whereas the *GT* and *Unknown* columns are initialised with the class distribution for all the instances at that rule. This distribution is obtained when the *AttributeAndLabel* list is created (step (1) in Figure 4.1). The sorted *AttributeAndLabel* list is scanned from the beginning and for each threshold value, the class distributions in the *LTE* and *GT* columns of the *SplitAndLabelDist* array are updated by shifting one instance from right to left according to the label associated with the threshold value. Figure 4.2 shows the schematic for this update. It should be noted that the *SplitAndLabelDist* array has all the necessary information to compute the information gain. Since the lists for continuous attributes are kept in a sorted order, the information gain for each threshold can thus be efficiently computed. If a winning threshold was found during the scan, it is saved and the *AttributeAndLabel* list and all the distribution arrays are de-allocated before processing the next attribute.



Position of cursor in scan	Pressure	Class
position 0		
	15	B
	20	B
	25	B
position 4	40	A
	50	A
	52	B
	56	A
position 8	65	A

AttributeAndLabel List

Cursor		LTE	GT	Unknown
position 0:	A	0	4	0
	B	0	4	0
position 4:	A	1	3	0
	B	3	1	0
position 8:	A	4	0	0
	B	4	0	0

State of *SplitAndLabelDist* arrays

Cursor	LTE	GT	Unknown
position 0:	0	8	0
position 4:	4	4	0
position 8:	8	0	0

State of *SplitDist* arrays

Cursor		
position 0:	A	4
	B	4
position 4:	A	4
	B	4
position 8:	A	4
	B	4

State of *LabelDist* arrays

Figure 4.2 Evaluating thresholds for continuous attributes.

4.4 Experimental Results

A series of tests was conducted to assess the performance of the proposed on-line discretisation method. The proposed method was compared with four state-of-the-art off-line discretisation procedures. These are the *equal-width* method, the *1R Discretizer* proposed by Holte, the *entropy-based discretisation* method introduced by Fayyad and Irani and the *"optimal"* discretisation method of Cai. These methods are representative of the different discretisation techniques described in section 4.2.2 and widely used in other comparative studies. Each of the off-line procedures was first employed to discretise all the data sets. The discretised data sets were then used to generate classification rules by the SRI algorithm and the results obtained were compared with those produced with the built-in discretisation procedure of SRI. The quality of the discretisation was evaluated based on the accuracy and complexity of the generated rules, as well as the time of execution.

The data sets used in this experiment were obtained from the UCI repository. The selected data sets either have only continuous attributes or a mixture of nominal and continuous attributes. They are summarised in Table 4.1. For details of these data sets, see appendix A. It is important to note that in performing the hold-out test, the training instances for each data set are separately discretised. Discretising all the data once before creating the partitions allows the discretisation method to have access to the test data, which is known to result in optimistic accuracy estimates. For SRI, the same parameters setting as given in chapter 3 (section 3.5.2) was followed. For the equal-width discretisation method, the number of intervals was set to 6. For the 1R Discretizer, the number of instances in one interval was set to 6 for large data sets, while the number was set to 3 for small data sets as recommended in (Holte, 1993).

Data Set Name	No. of Instances	No. of Nominal Attributes	No. of Continuous Attributes	No. of Classes
Abalone	4177	1	7	29
Anneal	898	32	6	6
Australian	690	8	6	2
Auto	205	10	15	6
Balance-scale	625	0	4	3
Breast	699	0	10	2
Cleve	303	7	6	2
Crx	690	9	6	2
Diabetes	768	0	8	2
German	1000	13	7	2
German-organisation	1000	12	12	2
Glass2	163	0	9	2
Heart-disease	270	0	13	2
Heart-Hungarian	294	5	8	2
Hepatitis	155	13	6	2
Horse-colic	368	15	7	2
Hypothyroid	3163	18	7	2
Ionosphere	351	0	34	2
Iris	150	0	4	3
Lymphography	148	15	3	4
Segment	2310	0	19	7
Shuttle	58000	0	9	7
Sick-euthyroid	3163	18	7	2
Sonar	208	0	60	2
Tokyo	961	0	46	2
Vehicle	699	0	18	4

Table 4.1 Summary of the data sets used in the experiments (Continuous and mixed-type data).

Table 4.2 shows the results of employing the SRI learning system using all considered discretisation schemes. As shown in the table, the accuracy obtained with the on-line discretisation method over all the data sets was in total higher than that produced with all the other off-line methods. Moreover, it produced significantly fewer rules in total than the other discretisation methods. The table also shows that the on-line discretisation method achieved results as good as the best off-line method in terms of the total CPU time and the number of rules evaluated during the search process. It should be noted that the execution time required by each of the four off-line methods to discretise the data sets was not included in the figures reported in the table. It could therefore be concluded that the proposed on-line discretisation method gives the best performance among the five methods tested.

4.5 Summary

Discretisation of continuous-valued attributes can be performed both as a pre-processing step preceding the learning phase and as a step integrated into the induction algorithm. In most classification rule learning systems, continuous-valued attributes are discretised prior to the learning process as a pre-processing step. This chapter has addressed the problem of inducing classification rules from data having both nominal and continuous-valued attributes by proposing a new method that discretises the continuous-valued attributes during the learning process. Incorporating discretisation into the learning process has the advantage of taking into account the bias inherent in the learning system as well as the different relationships among continuous and nominal attributes, leading to improved performance. The careful implementation of the discretisation method and the SRI pruning rules which discard portions of the search space without losing the best solution enables an elegant and

	On-line				Optimal				Entropy				1RD				Equal-width			
Data Set Name	Acc. (%)	# Rules	# Rules explored	Time (s)	Acc. (%)	# Rules	# Rules explored	Time (s)	Acc. (%)	# Rules	# Rules explored	Time (s)	Acc. (%)	# Rules	# Rules explored	Time (s)	Acc. (%)	# Rules	# Rules explored	Time (s)
Abalone	**25.4**	66	21766	262	24.4	35	10045	100	25.0	**30**	**6218**	**59**	22.0	40	12379	125	24.1	36	8209	64
Anneal	97.3	17	1291	11	**97.7**	18	1639	9	97.0	**14**	**1076**	**7**	95.7	19	1294	10	95.3	22	2149	14
Australian	**83.7**	35	**4139**	7	82.6	35	4937	7	83.0	**34**	4851	**6**	78.3	46	5920	20	82.6	47	6017	13
Auto	**65.2**	15	2865	5	62.3	15	**1792**	**4**	**65.2**	**14**	2111	6	60.9	19	1923	5	60.9	18	2053	7
Balance-scale	**83.7**	19	954	1	79.4	28	1407	2	71.8	**10**	**511**	**0**	71.8	12	573	1	73.7	36	2495	3
Breast	93.1	**9**	742	**1**	95.7	13	747	2	**96.6**	11	721	**1**	95.7	**9**	742	**1**	93.1	11	**649**	**1**
Cleve	81.3	22	2033	**2**	**82.2**	**19**	**1950**	**2**	74.3	21	2230	**2**	77.2	24	2557	3	74.3	24	2437	3
Crx	**82.5**	**28**	**3671**	**6**	77.5	34	4916	7	**82.5**	**28**	4013	7	78.0	43	5686	18	77.0	31	4232	9
Diabetes	67.2	23	2435	6	**68.8**	20	2045	**2**	67.6	**17**	**1096**	**2**	64.5	26	3451	7	67.2	32	4308	9
German	70.9	**33**	**6972**	**13**	**73.1**	40	8385	15	**73.1**	40	8385	15	72.5	49	7749	15	71.2	53	13336	19
German-org.	73.6	**32**	7991	17	**75.1**	33	**7289**	**10**	74.0	33	7298	11	71.0	40	8301	18	73.3	45	11721	27
Glass2	74.5	10	531	1	76.4	9	215	**0**	70.9	**5**	**126**	**0**	70.9	13	507	1	**81.8**	15	965	1
Heart-disease	80.0	**12**	**1262**	**1**	86.7	17	1446	**1**	**87.8**	15	1462	**1**	81.1	23	2134	2	76.7	23	2218	3
Heart-Hungarian	79.5	11	815	1	**80.6**	8	497	1	79.6	**7**	**380**	**0**	74.5	14	1040	1	76.5	14	1491	1
Hepatitis	**82.7**	7	583	1	80.8	**5**	**314**	1	80.8	**5**	316	1	80.8	6	379	1	80.8	7	455	1
Horse-colic	**86.8**	21	**3623**	**4**	73.5	22	3769	5	77.9	**20**	3799	6	76.5	31	5304	10	79.4	23	4851	8
Hypothyroid	98.6	**12**	1698	14	**98.9**	**12**	**1357**	**10**	97.9	15	1698	13	97.3	19	1757	19	94.1	33	5505	45
Ionosphere	88.8	**10**	2511	5	86.3	14	2603	6	86.3	13	**1593**	**4**	**89.7**	20	4242	13	83.8	21	4843	14
Iris	94.0	**6**	93	0	**96.0**	**6**	**35**	0	**96.0**	**6**	43	0	94.0	**6**	**35**	0	94.0	7	56	0
Lymphography	**84.0**	9	1072	1	78.0	8	874	1	**84.0**	**7**	**845**	1	78.0	8	874	1	**84.0**	8	1050	1
Segment	93.1	**31**	8565	53	**93.9**	43	**7551**	**43**	91.2	64	9620	64	76.2	117	22114	154	91.6	62	9751	70
Shuttle	99.5	**14**	1932	260	**99.6**	17	2334	278	99.5	30	4012	351	95.6	21	2348	283	91.6	17	**1449**	**243**
Sick-euthyroid	96.7	25	4235	25	95.5	34	4763	27	**97.3**	**20**	**2863**	**13**	96.4	31	3973	23	83.5	45	9072	56
Sonar	**72.9**	**10**	2538	7	71.4	12	**1262**	**4**	71.4	12	**1262**	**4**	67.9	26	8178	51	65.7	20	6870	27
Tokyo	91.7	**10**	3210	11	**92.9**	18	4335	19	90.4	13	**2672**	**8**	88.1	16	4889	26	89.0	17	3375	14
Vehicle	**69.9**	**38**	**9942**	**20**	67.4	61	12583	25	64.9	57	11179	22	58.2	82	13053	27	60.6	73	15576	33
Total	**2116.4**	**525**	97469	735	2096.6	576	89090	**581**	2085.8	541	**80380**	604	2012.5	760	121402	835	2025.6	740	125133	686

Table 4.2 Performance of discretisation methods when used in SRI.

efficient implementation of the search procedure. The proposed on-line discretisation method and four other state-of-the-art off-line discretisation methods have been tested on well-known machine learning data sets consisting of continuous and mixed-mode attributes. In all cases, the SRI algorithm has been used to generate the rule set, but in the four off-line methods the data has been pre-processed using the corresponding procedure to discretise all continuous attributes. The tests have shown that the proposed method significantly improves the classification accuracy. It also achieves results comparable with those of the best off-line method in terms of execution time and compactness of the rule set.

CHAPTER 5

MDL-BASED PRUNING OF RULE SETS

5.1 Motivation

For any learning method to work successfully with large data sets, it must be capable of learning accurately in the presence of noise. Existing rule learning systems are computationally expensive when applied to large noisy data sets (Cohen, 1995). Noisy data are also a problem for most learning algorithms because it is hard to distinguish between rare exceptions and erroneous examples. Pruning is a standard way of dealing with noisy data so as to avoid overfitting the training data set. Pruning is a method for reducing the error and complexity of induced models.

Several pruning techniques have been developed. They can be categorised as pre-pruning, post-pruning and hybrid pruning (Fürnkranz, 1996). Pre-pruning techniques deal with noise during concept generation. Their basic idea is to stop the specialisation of rules, although rules so produced may be over-general. Rules are therefore allowed to cover a few negative examples, if the alternative is deemed to be too costly.

While pre-pruning techniques deal with noise in the data during rule set construction, post-pruning techniques attempt to improve the rule set after it has been extracted. A commonly used post-pruning technique aims to remove conditions from rules and eliminate certain rules from the rule set. The basic idea is to test whether the removal of a single condition or even of an entire rule would lead to a decrease in the quality of the concept description, usually measured in terms of classification accuracy on the test set. If this is not the case, the condition or rule is removed.

Research has been carried out to combine these two techniques by initially applying pre-pruning to reduce the over specialisation of the rule sets and then using post-pruning to complete the process. This hybrid approach provides a balance between pre-pruning speed and accuracy of the pruned rule set.

Most of the existing pruning techniques were originally designed for decision trees and only a few can be used directly for rule set processing. Moreover, a significant drawback of many of these techniques is the necessity to split the training data set into a growing set and a pruning set. Dividing the data set raises two problems. First, setting aside some data for the pruning set reduces the number of instances available for learning. Second, discarding portions of the generated rules based only on their evaluation on the pruning set makes pruning techniques very sensitive to the size of this set. In the case of a small pruning set, which is always true in the later stages of the processing, the error estimate usually has a high variance and is therefore not reliable.

In this chapter, three different techniques for pruning rule sets based on the Minimum Description Length (MDL) principle are presented. An important advantage of these techniques is the fact that all of the training data can be used for both inducing and evaluating rule sets.

This chapter is organised as follows. Section 2 reviews current pruning techniques in the context of inductive learning. Section 3 presents the MDL criteria used for pruning and discusses previous coding strategies. It also introduces a new MDL measure for rule sets based on the ideas of Quinlan (1995). Section 4 describes the SRI rule induction algorithm using the new MDL measure as a stopping criterion and as a criterion for incremental, post-, and hybrid pruning. Empirical results are reported in section 5.

5.2 Existing Pruning Techniques

Pre-pruning usually employs some stopping criterion for deciding when to stop adding conditions to a rule and when to stop adding rules to the rule set. CN2, for instance, utilises a significance test to check whether the current rule correctly captures the class distribution of the training instances and to decide whether or not to specialise the rule further. Pre-pruning techniques are generally fast, but there is always the danger that a predefined criterion will over-simplify the rule set (Fürnkranz, 1994a; Frank, 2000).

Post-pruning techniques are commonly used in decision tree learning algorithms. Reviews of the most well-known post-pruning techniques can be found in (Mingers, 1989), (Breslow and Aha, 1996) and (Esposito et al., 1997). Generally, post-pruning techniques are more accurate than pre-pruning techniques, but are also more computationally expensive.

The most common post-pruning technique is *Reduced Error Pruning* (REP). This simple technique which was designed for decision tree learning (Quinlan, 1987) has been adopted for rule learning (Pagallo and Haussler, 1990; Brunk and Pazzani, 1991). After the training set is split into a growing and a pruning set according to some user-specified ratio, a consistent rule set that covers all of the positive and none of the negative examples is learned from the growing set. This rule set is then simplified by repeatedly deleting conditions and rules until any further deletion would result in a decrease in predictive accuracy as measured on the pruning set.

Using REP for rule learning has proved effective in raising predictive accuracy in noisy domains (Cohen, 1993; Frank and Witten, 1999; Elomaa and kääriäinen, 2001). However, this technique has several shortcomings. REP is very inefficient because the overly specific rule set it generates in its first phase can be much more complex than the final rule set. Therefore, much work is wasted in learning and subsequently removing superfluous conditions and rules. Another problem with REP, as pointed out by Fürnkranz and Widmer (1994), is that it is not appropriate for rule induction. This is because, in rule learning, the induction of the second rule is based on the instances remaining after the removal of the instances covered by the first rule. If the first rule is to be pruned away, this would affect the induction of the second rule.

To solve the inefficiency problem, Cohen (1993) proposed an alternative overfit-and-simplify technique called *Grow* that was competitive with REP with respect to error rate and was an order of magnitude faster on a set of benchmark problems. However, Grow still suffers from the inefficiency caused by the need to generate an overly specific rule set first. Moreover, it has been shown that *Grow* systematically overfits the target concept on noisy data. Cohen, subsequently, tried to improve *Grow* by adding two stopping heuristics to the initial overfitting stage, thus achieving a further speed-up in the technique. The goal of the stopping conditions in this context is not to prevent overfitting entirely, but to reduce its extent so that the post-pruning phase can start with a better rule set and hence requires less computational effort. Fürnkranz (1994b) proposed a different implementation of this approach. He developed a technique called *Top-Down Pruning* (TDP) that uses a simple method to generate rule sets pruned to different degrees in a top-down, general-to-specific order. The accuracies of the rule sets were evaluated on a separate set of data and the most specific rule set with an accuracy comparable to that of the best rule set up to that point was selected to start the post-pruning phase. TDP was compared to REP in a variety of domains. Experiments have shown that TDP is significantly faster and a little more accurate than REP.

In another attempt to solve the above-mentioned problems of REP, Fürnkranz and Widmer (1994) developed a learning algorithm called *Incremental Reduced Error Pruning* (IREP). IREP integrates pre-pruning and post-pruning in a way that avoids the expensive initial phase of overfitting. When a rule is to be pruned, the training data set is split into a growing set and a pruning set. After a rule is generated from the growing set, it is immediately pruned based on its performance on the pruning set. This ensures that the algorithm can remove training instances covered by the pruned rule before subsequent rules are learned. Thus the influence of these instances on the learning of future rules can be avoided. It has been confirmed experimentally that IREP gives significant run-time improvements over REP and Grow, and learns a much better rule set. However, in domains with a very specific concept description, REP is more appropriate.

5.3 Minimum Description Length (MDL) Principle

The Minimum Description Length (MDL) principle (Rissanen, 1986; Barron et al., 1998; Grunwald, 2000; Tirri, 2001), also called the Minimum Message Length (MML) principle (Georgeff and Wallace, 1984), is a powerful method for inductive inference. It states that the best model derivable from a set of observed data is the one that permits the greatest compression of the data. This is based on the idea that the greater the compression of the data, the greater the ability to discover regularity in the data.

Among the many techniques for pruning, those based on the MDL principle are particularly attractive because they provide a framework for balancing the complexity and the accuracy of a particular induced model. Several authors have proposed pruning techniques based on the MDL principle (Quinlan and Rivest, 1989; Wallace and Patrick, 1993; Forsyth et al., 1994; Kovačič, 1994; Mehta et el., 1995; Pfahringer, 1995b; 1997; Robnik-šikonja and Kononenko, 1998). These methods seek models that maximally compress the data and differ in the coding scheme they employ. In the following sub-sections, existing coding schemes are briefly described and a new scheme for encoding rule sets is presented.

5.3.1 Existing Coding Methods

The application of the MDL principle to the problem of pruning rule sets can be seen as a data communication problem. Let a sender and a receiver both have information on the number of attributes, the number of possible values for each attribute and the number of possible classes and a description of the training instances in terms of attribute values. The sender also knows the label of each instance and must communicate this information to the receiver. Obviously, the sender can transmit each instance together with the label to the receiver. Alternatively, the sender can also find a model that will enable the receiver to determine the labels of the instances, send the receiver a description of this model, and then transmit the instances that are the "exceptions" to the model. The sender may have a series of choices between a more complex model that fits the training data well and a simpler model that is less accurate. The MDL principle states that the best model to infer from a set of data is the one that minimises the sum of the coding length of the model and the coding length of the data given the model. If M is a model derived from the training data D, the total coding length, $L(M, D)$, is defined as:

$$L(M, D) = L(M) + L(D/M) \qquad (5.1)$$

where $L(M)$ is the coding length to describe the model M and $L(D/M)$ is the coding length to describe the data given model M. Both $L(M)$ and $L(D/M)$ are measured in "bits" using an appropriate coding scheme. In the context of rule induction, the models are the different rule sets that can be obtained by pruning the initial rule set, and the data is the training set. The objective of MDL pruning is to find the model that minimises the value

L(M, D). The model with the minimal total coding length is also the most probable model explaining the data as pointed out by Rissanen (1986), and thus will have the greatest accuracy when classifying new unseen instances.

Based on this idea, the coding strategy is now detailed.

5.3.1.1 Model encoding

The coding length of a model *M*, *L(M)*, is the sum of the coding lengths of its rules:

$$L(M) = \sum_{i} L(r_i) \tag{5.2}$$

Each rule consists of a sequence of conditions where each condition is either a value of the nominal attribute or an interval of values (either greater-than, GT, or less-than-or-equal-to, LTE, some threshold value) of the continuous attribute. Therefore, the coding length for all conditions of a single rule is the sum of the coding lengths of these three different possible kinds of conditions:

$$L(r_i) = L(Nominal\_Conds) + L(GT\_Conds) + L(LTE\_Conds) \tag{5.3}$$

The coding length of the nominal conditions is given by Equation 5.4 (adapted from (Pfahringer, 1997) and (Robnik-šikonja and Kononenko, 1998)):

$$L(Nominal\_Conds) = Log_2 \binom{N_n}{m_1} + \sum_{i=1}^{m_1} Log_2\, n_{A_i} \tag{5.4}$$

where N_n is the number of all nominal attributes, m_1 the number of nominal attributes involved in the antecedent of the rule and n_{A_i} the number of possible values that a certain nominal attribute A_i can take.

The first term on the right-hand side of the above equation is the average coding length for selecting a subset of attributes from the set of all nominal attributes. The second term is the coding length for specifying the respective values for each of the selected attributes.

Similarly, the coding lengths of the continuous conditions, *L(GT_Conds), L(LTE_Conds)*, are estimated by first selecting the continuous attributes actually involved in the antecedent of the rule and then encoding the respective thresholds as shown in Equations 5.5 and 5.6 (adapted from (Pfahringer, 1997) and (Robnik-šikonja and Kononenko, 1998)):

$$L(GT\_Conds) = Log_2\binom{N_c}{m_2} + \sum_{i=1}^{m_2} L(t_{ij}) \qquad (5.5)$$

$$L(LTE\_Conds) = Log_2\binom{N_c}{m_3} + \sum_{i=1}^{m_3} L(t_{ij}) \qquad (5.6)$$

where N_c is the number of all continuous attributes, and m_2 and m_3 are the numbers of continuous attributes used to create conditions of the form $A_i > t_{ij}$ and $A_i \leq t_{ij}$ respectively.

The cost of specifying a single threshold for a continuous attribute A_i is given by:

$$L(t_{ij}) = Log_2\,(d\text{-}1) \tag{5.7}$$

where *d-1* is the number of possible thresholds, and *d* is the number of distinct values for the attribute A_i occurring in the examples covered by the current rule.

5.3.1.2 Data encoding

Encoding the data given the model may be thought of either as encoding the data points that are covered by the model or as encoding the exceptional instances that are erroneously classified by the model. There are several different schemes for encoding the classes of the instances covered by model *M*. Let *S* be a set of instances covered by model *M* containing *N* instances, each belonging to one of *k* classes. Let n_{C_j} be the number of instances in class C_j. The cost of encoding the classes for the *N* instances is given in (Quinlan and Rivest, 1989) as:

$$L(S/M) = Log_2\binom{N+k-1}{k-1} + Log_2\binom{N}{n_{C_1}, n_{C_2}, \ldots, n_{C_j}, \ldots, n_{C_k}} \tag{5.8}$$

The first term in Equation 5.8 is the number of bits needed to specify the class distribution of the training instances, that is, the number of instances in each class. The second term is the number of bits required to encode the class for each instance once the class distribution is known.

Krichevsky and Trofimov (1983) proposed another scheme (Equation 5.9):

$$L(S/M) = \sum_{j=1}^{k} n_{c_j} Log_2(\frac{N}{n_{c_j}}) + \frac{k-1}{2} Log_2(\frac{N}{2}) + Log_2(\frac{\pi^{k/2}}{\Gamma(k/2)}) \qquad (5.9)$$

where $\Gamma(\,)$ is the Gamma function and is defined by the integral:

$$\Gamma(z) = \int_0^\infty y^{z-1} e^{-y} dy \qquad (5.10)$$

It has been shown that Equation 5.9 yields more accurate encoding costs than Equation 5.8, especially when some n_{c_j} are close to either *0* or *N* (Mehta et al., 1995).

A scheme for coding the exceptions to the model was first introduced in Quinlan (1993) and then in Quinlan (1994) with a slight modification. The basic idea can be outlined as follows. Given a set of rules *R* describing the positive class, the class of each instance can be given by identifying the misclassified instances of the rules *R*. Assuming a binary-class problem, misclassified instances can be specified by indicating which of the instances covered by the rules *R* are false positives and which of those not covered are false negatives, i.e. two sets of exceptions. It should be noted that learning tasks involving multiple classes, when being dealt with on a class-by-class basis, are essentially a two-class problem in which the goal is to generate rules that cover instances of one of the classes, called the target class, while not covering instances belonging to any other class. The coding length of a sensible encoding scheme for identifying *t* exceptions in *N* instances is (assuming all ways of selecting *t* of the *N* instances are equally likely):

$$L(N, t) = Log_2\,(N+1) + Log_2\binom{N}{t} \qquad (5.11)$$

which may be interpreted as the cost of specifying t in the range 0 to N, plus the cost of specifying which selections of t out of N instances are exceptions.

Let C be the number of instances covered by the rules and $\overline{C}$ the number of instances not covered. Further, let f_p be the number of false positive instances (instances that are covered by the rules but are really negative) and f_n the number of false negative instances (positive instances not covered by the rules). The exceptions cost of specifying the data given the model is then:

$$L(D/M) = L(C, f_p) + L(\overline{C}, f_n) \qquad (5.12)$$

The first term is the number of bits needed to indicate the false positives among the instances covered by the rules and the second term gives a similar expression for identifying the false negatives among the instances not covered. This is called the *divided strategy* since errors are separated into two groups.

This scheme for encoding exceptions has two problems. First, it is symmetric and would give ambiguous results as the cost of encoding f_p and f_n can be the same as that of encoding $(N\text{-}P) - f_p$ and $P - f_n$ respectively, where P is the number of instances belonging to the target class and $(N\text{-}P)$ is the number of negative instances. Second, it could lead to poor choices among contending models. In order to solve the second problem, Quinlan (1994) described a simple approach which attempts to restrict the candidate models from which the final model is selected. He introduced a bias in favour of models whose predicted class distribution matches that observed in the data. This bias was justified in that a model learned from the data should accurately summarise that data. To implement this bias, an ad-hoc penalty function that significantly increases the description length of unsatisfactory models was employed. Empirical results showed that this bias was effective in selecting models with a lower error rate on unseen instances.

Quinlan (1994) also explored an alternative scheme called *uniform coding strategy* for estimating $L(D/M)$. This scheme (Equation 5.13) encodes errors in a single group rather than separating them into false positives and false negatives:

$$L(D/M) = L(N, e) \tag{5.13}$$

where N is the total number of instances in the training data set and e is the total number of errors, calculated as $f_p + f_n$.

Equation 5.13 still exhibits a counter-intuitive symmetry: the cost of encoding e errors is the same as the cost for N - e errors.

Instead of relying on an artificial penalty function, Quinlan (1995) presented a *biased exceptions coding strategy* that achieves the same effect in a manner consistent with the MDL principle itself. It is based on the observation that the proportions of the target class instances predicted by a model and observed in the training data are the same when the numbers of false positive and false negative errors are equal. Before the strategy is defined, a theoretically optimal scheme for coding the exceptions needs to be introduced.

In Equation 5.11, the t exceptions are encoded based on the assumption that t is equally likely *a priori*. The length of an ideal coding scheme in which t may have unequal likelihoods is given by:

$$L(N, t, p) = Log_2 (N+1) + t\, Log_2 (1/p) + (N-t)\, Log_2 (1/(1-p)) \qquad (5.14)$$

where p is the probability of a message selection. Of course, this assumes that p is independent of the previous messages and that it is known to the receiver.

The coding length of the data given the model can then be defined by:

$$L(D/M) = L(C, f_p, e/(2C)) + L(\overline{C}, f_n, f_n /\overline{C}) \qquad (5.15)$$

The term $L(C, f_p, e/(2C))$ is the number of bits needed to specify the error messages for covered instances. The term $L(\overline{C}, f_n, f_n /\overline{C})$ is the number of bits required to encode the error messages for instances not covered. The error probabilities of covered and non-covered instances are derived from the assumption that false positives and false negatives are balanced and that the sender first transmits the errors in the C instances covered by the model and then communicates those in the $\overline{C}$ instances not covered. There is a slight complication: if the number C of covered instances is small, $e/2C$ may be greater than one. To overcome this problem, the above equation is followed when the model covers at least half the instances. If less than half are covered, the following equation is used:

$$L(D/M) = L(\overline{C}, f_n, e/(2\overline{C})) + L(C, f_p, f_p/C) \tag{5.16}$$

where the false negative errors in the instances not covered are transmitted first, using the probability $e/(2\overline{C})$, followed by the false positives using f_p/C.

Adopting the coding scheme represented by Equation 5.14, Equations 5.12 and 5.13 can be rewritten as:

$$L(D/M) = L(C, f_p, f_p/C) + L(\overline{C}, f_n, f_n/\overline{C}) \tag{5.17}$$

$$L(D/M) = L(N, e, e/N) \tag{5.18}$$

The biased strategy and the divided strategy, represented respectively by Equation 5.15 and Equation 5.17, are similar except that the former uses the initial assumption of equal numbers of false positive and false negative errors to derive error probabilities for covered and non-covered instances.

5.3.2 An Alternative Coding Method

The encoding schemes represented by Equations 5.15 to 5.18 can still lead to anomalous choices among candidate models. This can be illustrated by a hypothetical example. Suppose a data set of 1000 instances, of which 300 belong to the target class, has six candidate models that give rise to various numbers of false positive and false negative errors as shown in Table 5.1. The models vary from over-specific to over-general. All six models are further presumed to have the same model cost so that the model with the lowest exceptions cost will be chosen. The number C of instances covered by a model is given by: $P+f_p-f_n$. In this situation, the divided strategy will choose M_5, with 197 errors, instead of the equally complex model M_1 that makes far fewer errors. The uniform approach will find an exact tie between M_1, with 152 errors on the training data, and M_6, with 848 errors. The biased approach has no difficulty in distinguishing between M_1 and M_6. However, it fails to select M_1 and M_2, with 152 and 158 errors respectively, which have much higher predictive accuracy than M_3, M_4, M_5 and M_6 with 170, 186, 197 and 848 errors. The choices made based on MDL in this example are clearly counter-intuitive.

In the above example, where the positive instances are in the minority, the divided strategy tends to select over-general models. Conversely, it has been found that it tends to select over-specific models when the positive instances are in the majority. The biased strategy substantially increases the coding length of the over-general models especially when the assumption of balanced errors is grossly incorrect. However, it also increases the coding length of the over-specific models, thereby still favouring over-general

Model	False Pos	False Neg	Instances Covered	Divided Encoding	Uniform Encoding	Biased Encoding	Alternative Biased Encoding
M_1	6	146	160	613.7	624.8	655.0	613.7
M_2	29	129	200	646.6	639.5	668.3	646.6
M_3	85	85	300	649.1	667.7	649.1	649.1
M_4	134	52	382	632.4	703.0	650.2	650.2
M_5	192	5	487	529.7	725.9	657.3	657.3
M_6	694	154	840	613.7	624.8	886.5	886.5

Table 5.1 Exceptions costs for six competing models.

models. This suggests an alternative one-sided biased coding scheme as follows: if either positive instances are in the minority and the model is too general, or positive items are in the majority and the model is too specific, the biased encoding method, Equation 5.15 or 5.16, should be used. Otherwise, the divided strategy, Equation 5.17, should be employed. The final column of Table 5.1 shows the exceptions costs of the alternative biased coding method for the given six models. In this example, MDL would now place M_1 and M_2 well ahead of the other models, which is an intuitively sensible outcome.

To test the usefulness of the new encoding method, three versions of the SRI rule induction learner were prepared that differ only in the method used to calculate exceptions costs. The first version used the new strategy described in this section. The other two versions employed the strategies of Equation 5.9 and Equation 5.15 or 5.16, respectively. It should be noted that when using Equation 5.9, it is assumed that the last rule of the model is the default rule, which uses the majority class in the training set to assign class labels to examples not covered by any previous rule. Thereby the whole training set is taken into account when estimating the total coding length. The tests showed that the new version led to much better results. Therefore, the new version is adopted for the rest of the experiments in this thesis.

5.4 Proposed Pruning Techniques

Using the encoding scheme proposed in section 5.3.2, three different pruning techniques for rule induction systems have been developed. This section describes the way in which these pruning techniques are used in the SRI inductive learner.

5.4.1 MDL-based Post Pruning (MDL_PP)

MDL_PP is performed after a rule set accurately classifying every instance in the training set has been constructed. The task is to find the rule set for each class that minimises the total coding length. The pruning procedure consists of two phases. In the first phase, the rule set is "greedily" pruned using a "delete-rule" operator. Rules are deleted one after another, starting from the last rule, so long as the total coding length does not increase. In the second phase, each of the remaining rules is pruned using another "delete-condition" operator. Conditions are repeatedly deleted starting from the last condition, whenever this improves (decreases) the total coding length, until no further improvements in the coding length is possible. Figure 5.1 shows an adaptation of the SRI algorithm in order to handle noisy data with post-pruning. The algorithm is identical to the one in Figure 3.1, except for the addition of the procedure *Prune_Rule_Set ()*.

5.4.2 MDL-based Hybrid Pruning (MDL_HP)

This strategy combines pre- and post-pruning. It first uses a stopping heuristic to find an intermediate rule set and then prunes this set to an appropriate level of generality in a subsequent post-pruning phase. The implementation of this approach is as follows.

After an induced rule is added to the current rule set, the total description length is computed. The new version of SRI stops adding rules when this description length is larger than the smallest description length obtained so far, or when there are no more positive examples. The rule set is then simplified by examining each rule in turn starting with the last rule added and deleting conditions from this rule in a greedy fashion,

```
Procedure Induce_Rules (TrainingSet, BeamWidth)
RuleSet = $\varnothing$
For each class in the TrainingSet Do
  CurrentClassRuleSet = $\varnothing$
  Instances = TrainingSet
  While Positives (Instances) $\neq \varnothing$ Do
    Rule = Induce_One_Rule (Instances, CurrentClass, BeamWidth)
    Instances = Instances – Covered_Positives (Rule, Instances)
    CurrentClassRuleSet = CurrentClassRuleSet $\cup$ {Rule}
  End While
  CurrentClassRuleSet = Prune_Rule_Set (CurrentClassRuleSet,TrainingSet)
  RuleSet = RuleSet $\cup$ CurrentClassRuleSet
End For
Return RuleSet
End
```

Figure 5.1 A pseudo-code description of SRI with MDL_PP.

beginning from the last condition, until any further deletion would increase the total coding length. This pruning strategy does not need the delete-rule operator, because the rules are constructed so as to reduce the total coding length, and learning stops when no more useful rules can be found. A pseudo-code version of the SRI algorithm that implements this procedure is given in Figure 5.2.

5.4.3 MDL-based Incremental Pruning (MDL_IP)

This pruning strategy is similar to IREP. It tightly integrates pruning into the learning procedure and thus allows more general rules to be constructed as early as possible during the learning process, as well as requiring less computation to build a rule set. However, unlike IREP, the pruning is based on all the training instances, which avoids splitting the training data set into a growing set and a pruning set.

Figure 5.3 shows a pseudo-code version of the SRI algorithm with incremental pruning. When a rule is generated, conditions in the rule antecedent are examined and an attempt is made to delete one condition at a time starting from the last condition added. If, according to their coding lengths, the new rule is better than the unpruned one, then it is accepted and pruning continues. If one condition cannot be removed, pruning stops. The pruning procedure also stops deleting conditions from the current rule if the number of positive examples covered by the rule resulting from that deletion is less than the number of negative examples covered. This ensures that the overall accuracy of the final rule set increases. The pruned rule is then added to the rule set and all covered positive examples are removed from the training set. From this set the next rule is learned. If the pruned rule

```
Procedure Induce_Rules (TrainingSet, BeamWidth)
RuleSet = $\emptyset$
For each class in the TrainingSet Do
  CurrentClassRuleSet = NewCurrentClassRuleSet = $\emptyset$
  Instances = TrainingSet
  MDL = L (CurrentClassRuleSet, TrainingSet)
  While Positives (Instances) $\neq \emptyset$ Do
    Rule = Induce_One_Rule (Instances, CurrentClass, BeamWidth)
    NewCurrentClassRuleSet = NewCurrentClassRuleSet $\cup$ {Rule}
    New_MDL = L (NewCurrentClassRuleSet, TrainingSet)
    If New_MDL > MDL Then
      Exit While
    Instances = Instances – Covered_Positives (Rule, Instances)
    MDL = New_MDL
    CurrentClassRuleSet = NewCurrentClassRuleSet
  End While
  CurrentClassRuleSet = Prune_Rule_Set (CurrentClassRuleSet,TrainingSet)
  RuleSet = RuleSet $\cup$ CurrentClassRuleSet
End For
Return RuleSet
End
```

Figure 5.2 A pseudo-code description of SRI with MDL_HP.

```
Procedure Induce_Rules (TrainingSet, BeamWidth)
RuleSet = $\varnothing$
For each class in the TrainingSet Do
  Instances = TrainingSet
  While Positives (Instances) $\neq \varnothing$ Do
    Rule = Induce_One_Rule (Instances, CurrentClass, BeamWidth)
    Rule = Prune_Rule (Rule, TrainingSet)
    If Conditions (Rule) = $\varnothing$ Then
      Exit While
    Instances = Instances – Covered_Positives (Rule, Instances)
    RuleSet = RuleSet $\cup$ {Rule}
  End While
End For
Return RuleSet
End
```

Figure 5.3 A pseudo-code description of SRI with MDL_IP.

has an empty body, it is assumed that no further rule can be found that explains the remaining positive examples and the learning process stops for the current class. Thus the MDL principle also serves as a stopping criterion.

The coding length calculation involves scanning all the training data set each time a rule or condition is considered for deletion. By gathering relevant information during the learning process, the efficiency of rule pruning can be significantly increased. Two data structures called *positive-negative* list and *sequence* list are attached to each rule. The sequence list memorises the order in which conditions are added to the antecedent of the rule during the specialisation process. Also, the distribution of instances covered by the rule among different classes is recorded in the positive-negative list when each condition is appended. Using this information, the coding length can be computed without scanning the whole training instance list.

5.5 Experimental Results

This section gives the results of a set of experiments designed to evaluate the performance of the three new pruning techniques when implemented in the SRI algorithm. Initially, the performance of SRI with each of these techniques was compared against that of SRI without pruning. Then, the performance of SRI together with the best of these three techniques was compared to that of C5.0 which, as previously mentioned, is probably the best performing commercially available induction algorithm.

The techniques were tested on 38 data sets selected from the UCI machine learning repository. The domains have nominal, continuous and mixed-type attributes. Table 5.2 summarises the main characteristics of the data sets. For a more detailed description of these data sets, see appendix A. The same experimental method as described in chapter 3 (section 3.4) was followed. SRI was executed with parameter values of 4, 1 and 2 for beam width, MinNegatives and MinPositives respectively. The default parameters of C5.0 were used.

5.5.1 Evaluation of the Different Pruning Techniques

The results obtained are shown in Tables 5.3 – 5.5. Table 5.3 presents the predictive accuracy of SRI without pruning and SRI with each of the pruning techniques. Table 5.4 presents the complexity of the rule sets. Table 5.5 presents the execution time in CPU seconds.

It is clear from the tables that the pruning techniques achieved a substantially lower complexity without sacrificing accuracy for most data sets. The pruning techniques when used with SRI even yielded better overall accuracy than SRI without pruning. In addition, the execution times of SRI when the pruning techniques were employed were significantly lower for all the data sets. Compared to the other pruning techniques, MDL_IP when used with SRI achieved the highest overall accuracy. The worst accuracy over all the data sets was obtained by MDL_HP. This can be explained by the fact that the stopping criterion used in MDL_HP resulted in over-general rule sets, causing a decrease in the classification accuracy. However, MDL_HP could be adopted for large

Data Set Name	No. of Instances	No. of Nominal Attributes	No. of Continuous Attributes	No. of Classes
Abalone	4177	1	7	29
Anneal	898	32	6	6
Australian	690	8	6	2
Auto	205	10	15	6
Balance-scale	625	0	4	3
Breast	699	0	10	2
Breast-cancer	286	9	0	2
Car	1728	6	0	4
Chess	3196	36	0	2
Cleve	303	7	6	2
Crx	690	9	6	2
Diabetes	768	0	8	2
German	1000	13	7	2
German-organisation	1000	12	12	2
Glass2	163	0	9	2
Heart-disease	270	0	13	2
Heart-Hungarian	294	5	8	2
Hepatitis	155	13	6	2
Horse-colic	368	15	7	2
Hypothyroid	3163	18	7	2
Ionosphere	351	0	34	2
Iris	150	0	4	3
Lymphography	148	15	3	4
Monk1	556	6	0	2
Monk2	601	6	0	2
Monk3	554	6	0	2
Mushroom	8124	22	0	2
Promoter	106	57	0	2
Segment	2310	0	19	7
Shuttle	58000	0	9	7
Sick-euthyroid	3163	18	7	2
Sonar	208	0	60	2
Soybean-large	683	35	0	19
Splice	3190	61	0	3
Tic-tac-toe	958	9	0	2
Tokyo	961	0	46	2
Vehicle	699	0	18	4
Vote	435	16	0	2

Table 5.2 Summary of the data sets used in the experiments (Nominal, continuous and mixed-type data).

Data Set Name	SRI without pruning	SRI with MDL_PP	SRI with MDL_HP	SRI with MDL_IP
Abalone	23.1	24.5	19.8	**25.1**
Anneal	**97.7**	92.7	93.7	**97.7**
Australian	81.7	86.5	83.9	**87.8**
Auto	**63.8**	59.4	58.0	62.3
Balance-scale	**88.5**	81.3	84.2	81.8
Breast	94.0	94.0	93.6	**95.3**
Breast-cancer	67.4	73.7	**76.8**	73.7
Car	**95.3**	91.7	90.8	92.0
Chess	**99.0**	98.9	98.9	98.1
Cleve	70.3	**81.2**	77.2	76.2
Crx	75.5	83.0	**83.5**	82.5
Diabetes	67.6	**69.5**	64.8	67.6
German	71.2	**74.8**	69.4	70.6
German-organisation	71.8	72.7	**73.6**	72.1
Glass2	74.5	74.5	**83.6**	80.0
Heart-disease	**80.0**	77.8	76.7	76.7
Heart-Hungarian	74.5	76.5	**77.6**	**77.6**
Hepatitis	80.8	82.7	80.8	**86.5**
Horse-colic	82.4	**85.3**	82.4	**85.3**
Hypothyroid	**99.0**	98.9	98.9	**99.0**
Ionosphere	83.8	84.6	**86.3**	84.6
Iris	94.0	94.0	94.0	94.0
Lymphography	**84.0**	**84.0**	80.0	**84.0**
Monk1	100.0	100.0	100.0	100.0
Monk2	63.3	**65.7**	62.1	**65.7**
Monk3	**100.0**	**100.0**	**100.0**	99.2
Mushroom	100.0	100.0	100.0	100.0
Promoter	74.3	74.3	74.3	**77.1**
Segment	**93.8**	91.6	91.7	93.5
Shuttle	**99.9**	**99.9**	99.6	99.7
Sick-euthyroid	97.1	**97.4**	**97.4**	95.4
Sonar	70.0	71.4	71.4	**74.3**
Soybean-large	**91.2**	89.5	90.4	90.4
Splice	89.0	89.9	89.4	**92.4**
Tic-tac-toe	97.8	97.8	**98.1**	**98.1**
Tokyo	92.7	**93.3**	**93.3**	91.5
Vehicle	**70.9**	67.4	67.4	66.3
Vote	**97.8**	**97.8**	**97.8**	97.0
Total	3157.5	3178.1	3161.1	**3191.0**

Table 5.3 Summary of predictive accuracies (%).

	SRI without pruning		SRI with MDL_PP		SRI with MDL_HP		SRI with MDL_IP	
Data Set Name	# Rules	# Conditions	# Rules	# Conditions	# Rules	# Conditions	# Rules	# Conditions
Abalone	376	1807	45	202	**24**	**87**	64	329
Anneal	21	50	**15**	**28**	16	32	17	41
Australian	47	156	**9**	**21**	10	23	15	42
Auto	17	57	**12**	**34**	14	38	13	39
Balance-scale	39	115	23	65	23	64	**16**	**44**
Breast	14	32	8	15	11	23	**6**	**7**
Breast-cancer	38	107	**2**	**1**	3	3	**2**	**1**
Car	82	315	73	285	59	215	**39**	**136**
Chess	37	159	30	111	**19**	**62**	21	74
Cleve	25	71	10	18	**8**	**16**	12	21
Crx	43	131	14	33	9	22	**5**	**8**
Diabetes	49	134	22	55	**3**	**3**	5	9
German	102	335	45	109	**2**	**2**	6	13
German-organisation	89	351	38	122	**6**	**14**	8	25
Glass2	11	19	10	17	10	16	**9**	**14**
Heart-disease	18	47	11	26	**5**	9	6	**8**
Heart-Hungarian	22	57	6	8	**4**	**6**	6	10
Hepatitis	11	24	3	3	11	23	**2**	**2**
Horse-colic	35	93	25	62	4	8	**3**	**4**
Hypothyroid	22	58	9	18	10	20	**6**	**13**
Ionosphere	14	27	11	21	9	15	**8**	**12**
Iris	6	9	**4**	**4**	6	9	6	9
Lymphography	13	30	10	22	**7**	14	**7**	**13**
Monk1	23	61	23	61	23	61	**8**	**10**
Monk2	54	183	6	15	2	1	**1**	**0**
Monk3	13	23	13	23	6	9	**3**	**2**
Mushroom	15	28	15	28	15	28	**14**	**25**
Promoter	9	14	9	14	4	6	**3**	**3**
Segment	36	135	**20**	**66**	**20**	**66**	27	90
Shuttle	25	83	21	64	**7**	**16**	9	24
Sick-euthyroid	37	122	18	54	**7**	**25**	10	39
Sonar	16	27	**10**	**17**	**10**	**17**	**10**	18
Soybean-large	33	92	**26**	**56**	28	59	27	69
Splice	106	372	60	173	27	119	**19**	**76**
Tic-tac-toe	24	76	21	62	**9**	**24**	**9**	**24**
Tokyo	17	35	9	16	**6**	9	**6**	**8**
Vehicle	52	180	**12**	**29**	**12**	**29**	31	101
Vote	14	35	8	20	10	24	**4**	**5**
Total	1605	5650	706	1978	**459**	**1217**	463	1368

Table 5.4 Summary of the complexities of the rule sets.

	SRI without pruning		SRI with MDL_PP		SRI with MDL_HP		SRI with MDL_IP	
Data Set Name	# Rules explored	CPU Time (s)	# Rules explored	CPU Time (s)	# Rules explored	CPU Time (s)	# Rules explored	CPU Time (s)
Abalone	68915	1234	68915	1278	**8677**	**123**	21086	256
Anneal	1401	11	1401	11	**1291**	**9**	**1291**	11
Australian	5105	10	5105	10	**1608**	**4**	1622	**4**
Auto	2865	**5**	2865	6	2808	**5**	**2511**	**5**
Balance-scale	1541	3	1541	4	1202	**2**	**834**	**2**
Breast	876	2	876	3	812	2	**528**	**1**
Breast-cancer	3018	4	3018	4	356	1	**285**	**0**
Car	9752	18	9752	18	8158	14	**4171**	**9**
Chess	11911	50	11911	52	**5538**	**27**	6224	29
Cleve	2072	2	2072	2	**915**	**1**	1259	**1**
Crx	4631	9	4631	10	1482	**2**	**724**	**2**
Diabetes	4188	10	4188	10	**450**	**1**	572	2
German	17001	39	17001	40	**799**	**2**	1744	4
German-organisation	18580	44	18580	46	**1949**	**3**	2592	6
Glass2	531	**0**	531	1	531	**0**	**500**	1
Heart-disease	1461	2	1461	2	**626**	**1**	629	**1**
Heart-Hungarian	1190	2	1190	2	**363**	**0**	562	**0**
Hepatitis	712	1	712	1	712	1	**277**	1
Horse-colic	4417	8	4417	8	1010	**1**	**758**	2
Hypothyroid	2206	15	2206	16	1371	10	**662**	**7**
Ionosphere	2753	7	2753	7	2407	**5**	**2078**	**5**
Iris	93	0	93	0	93	0	93	0
Lymphography	1103	1	1103	1	847	1	**809**	1
Monk1	1838	1	1838	1	1838	2	**505**	**0**
Monk2	4697	3	4697	3	345	1	**225**	**0**
Monk3	647	0	647	0	288	0	**158**	0
Mushroom	1856	**41**	1856	42	1856	42	**1717**	**41**
Promoter	2380	5	2380	5	1748	3	**1234**	**2**
Segment	8892	89	8892	92	**6504**	**57**	6868	73
Shuttle	2517	1766	2517	1794	**928**	**644**	1928	1188
Sick-euthyroid	4769	35	4769	36	**1910**	**14**	2179	20
Sonar	3066	9	3066	9	**2538**	**6**	2544	**6**
Soybean-large	5577	18	5577	18	**4940**	**16**	5396	17
Splice	160041	754	160041	766	35823	199	**29877**	**177**
Tic-tac-toe	4079	4	4079	5	**1834**	3	**1834**	**2**
Tokyo	4079	14	4079	14	2180	8	**2081**	**7**
Vehicle	11994	33	11994	33	**3640**	**9**	8264	22
Vote	1092	1	1092	1	891	1	**427**	**0**
Total	383846	4250	383846	4351	**111268**	**1220**	117048	1905

Table 5.5 Summary of the total number of rules searched and the execution time in CPU seconds.

data sets because it is much faster than the other pruning techniques and produces rules that are more compact. It is also clear that MDL_IP when used with SRI yielded comparable results in terms of rule sets complexity and execution time with MDL_HP, outperforming the MDL_PP method. Overall, the MDL_IP method appears to be the best choice for use with the SRI algorithm.

5.5.2 Comparison with C5.0

The performance of SRI with MDL_IP was compared against that of C5.0. In addition to the data sets of Table 5.2, the *Adult* data set (see appendix A for details) was also used in the comparison. The results are shown in Table 5.6. In terms of classification accuracy, SRI with MDL_IP outperformed C5.0 for 20 out of 39 data sets. On 5 other data sets, SRI with MDL_IP developed rule sets with accuracies similar to C5.0. Only on 14 data sets, did C5.0 outperform SRI with MDL_IP. In terms of the number of rules created, SRI with MDL_IP obtained fewer rules 28 times and C5.0, 7 times. Taking into account both the classification accuracy and the number of rules, it can be concluded that SRI with MDL_IP outperformed C5.0 in the classification experiments conducted.

5.6 Summary

This chapter has reviewed existing pruning techniques for inductive learning algorithms. Three new techniques that employ the MDL principle for pruning rule sets have been presented. These techniques have a distinct feature compared with other pruning procedures in that they do not require the training data set to be split into two separate growing and pruning sets. The proposed techniques are also capable of dealing with

Data Set Name	C5.0		SRI with MDL_IP	
	Acc. (%)	No. of Rules	Acc. (%)	No. of Rules
Abalone	23.4	522	**25.1**	**64**
Adult	**86.4**	100	81.1	**5**
Anneal	93.3	**11**	**97.7**	17
Australian	87.4	20	**87.8**	**15**
Auto	62.3	23	62.3	**13**
Balance-Scale	81.3	19	**81.8**	**16**
Breast	95.0	9	**95.3**	**6**
Breast-cancer	**75.8**	17	73.7	**2**
Car	91.8	58	**92.0**	**39**
Chess	97.2	21	**98.1**	21
Cleve	**77.2**	13	76.2	**12**
Crx	**84.5**	23	82.5	**5**
Diabetes	**70.7**	14	67.6	**5**
German	**72.7**	15	70.6	**6**
German-organisation	71.8	17	**72.1**	**8**
Glass2	69.1	9	**80.0**	9
Heart-disease	**78.9**	12	76.7	**6**
Heart-Hungarian	74.5	7	**77.6**	**6**
Hepatitis	76.9	5	**86.5**	**2**
Horse-colic	83.8	10	**85.3**	**3**
Hypothyroid	94.8	**5**	**99.0**	6
Ionosphere	**89.7**	**6**	84.6	8
Iris	92.0	**5**	**94.0**	6
Lymphography	76.0	7	**84.0**	7
Monk1	100.0	17	100.0	**8**
Monk2	65.7	1	65.7	1
Monk3	**100.0**	6	99.2	**3**
Mushroom	99.8	**10**	**100.0**	14
Promoter	74.3	7	**77.1**	**3**
Segment	93.4	**24**	**93.5**	27
Shuttle	**99.9**	12	99.7	**9**
Sick-euthyroid	90.4	**8**	**95.4**	10
Sonar	74.3	11	74.3	**10**
Soybean-large	**93.4**	32	90.4	**27**
Splice	**92.7**	60	92.4	**19**
Tic-tac-toe	92.2	34	**98.1**	**9**
Tokyo	**92.3**	8	91.5	**6**
Vehicle	**69.9**	46	66.3	**31**
Vote	97.0	5	97.0	**4**
Total	3242.0	1229	**3272.0**	**468**

Table 5.6 Results for C5.0 and SRI with MDL_IP.

continuous data and multi-valued classes, making them applicable to a wide range of real-world problems. Experiments using the SRI classifier have demonstrated the performance improvements achieved.

CHAPTER 6

RULES-6: A SIMPLE RULE INDUCTION ALGORITHM FOR DATA MINING

6.1 Motivation

RULES-3 Plus is a simple rule induction algorithm for extracting IF-THEN rules from a set of training examples. The algorithm still suffers from problems that limit its efficiency and widespread use. One of the main problems is that RULES-3 Plus induces rules that are both consistent and complete (i.e., covering all positive examples and no negative examples) with regard to the training data. In the case of noisy data, this leads to the generation of over-specific rules that overfit the training data. A second problem is that continuous-valued attributes are discretised using a simplistic equal-width method before data is passed to the learning algorithm. This discretisation method is arbitrary and does not seek to discover any information inherent in the data, thereby hampering the ability of RULES-3 Plus to learn. Finally, RULES-3 Plus does not employ any methods for dealing with noisy data.

This chapter presents RULES-6, a new rule induction algorithm which addresses the weaknesses of the RULES-3 Plus algorithm. In particular, it employs a new search method which relaxes the consistency constraint and uses search-space pruning rules which significantly reduce the search time. It also adopts effective methods for handling continuous and noisy data. These enhancements enable the efficient generation of accurate and compact rule sets.

The chapter is organised as follows. Section 2 briefly reviews RULES-3 Plus. Section 3 gives a detailed description of RULES-6. Section 4 discusses the evaluation of the performance of RULES-6 using real data.

6.2 The RULES-3 Plus Algorithm

RULES-3 Plus (RULe Extraction System – Version 3 Plus) is a simple rule induction algorithm belonging to the RULES family. RULES-3 Plus and previous versions in the family have been employed for the extraction of classification rules for solving different manufacturing and engineering problems, e.g., the recognition of design form features in CAD models for computer aided process planning (Pham and Dimov, 1998), the mapping of manufacturing information to design features (Pham and Dimov, 1998) and the classification of defects in automated visual inspection (Jennings, 1996). This section gives a brief description of RULES-3 Plus.

6.2.1 Algorithm Description

RULES-3 Plus extracts a set of classification rules from a collection of examples, each belonging to one of a number of given classes. The examples together with their associated classes constitute the set of training examples from which the algorithm induces general rules. Every example is described in terms of a fixed set of attributes, each with its own set of possible values.

In RULES-3 Plus, an attribute-value pair constitutes a condition. If the number of attributes is N_a, a rule may contain between one and N_a conditions, each of which must be a different attribute-value pair. Only conjunction of conditions is permitted in a rule and therefore the attributes must all be different if the rule comprises more than one condition.

RULES-3 Plus works in an iterative fashion. In each iteration, it takes a "seed" example not covered by previously created rules to form a new rule. Having found a rule, RULES-3 Plus marks those examples that the rule covers and appends the new rule to its rule set. The algorithm stops when all examples in the training set are covered. This produces an unordered set of complete and consistent rules. Note that the examples covered by previously formed rules are marked in order to stop RULES-3 Plus from repeatedly finding the same rule. However, these examples continue to be used to guide the specialisation process and to assess the accuracy and generality of newly formed rules. By considering the whole set of examples when forming rules, RULES-3 Plus is less prone to the *fragmentation* problem (i.e., the amount of available data reducing as induction progresses) (Pagallo and Haussler, 1990; Domingos, 1997b) and the *small disjuncts* problem (i.e., rules covering few training examples having a high error rate) (Holte et al., 1989; Weiss, 1995; Weiss and Hirsh, 1998; 2000; Frayman et al., 1999). As a result, a compact rule set is obtained. Also, RULES-3 Plus does not work on a class-per-class basis. The class of the selected seed example is considered positive and all the remaining classes are regarded as negative.

To form a rule, RULES-3 Plus performs a general-to-specific beam search for the most general and consistent rule. It starts with the most general rule and gradually specialises it considering only conditions extractable from the selected seed example. The aim of specialisation is to construct a rule that covers the seed example and as many positive examples as possible while excluding all negative examples. The result is a rule that is consistent and as general as possible. To do this, an array called *AttributesAndValues* is constructed, the elements of which are attribute-value pairs associated with the seed example under consideration. The total number of elements in the array is equal to the number of attributes in the example. In the first step, each element of the array is examined to decide whether it can form a rule with that element as a condition. If any of the formed rules is consistent, it is taken as a candidate rule and the search process stops. Otherwise, if the formed rules pertain to more than one class, they are added to a list called *PartialRules*. The maximum number of rules in *PartialRules*, called the beam width, is specified by the user and determines how many alternatives are considered in each step. Only the rules in *PartialRules* are considered for further specialisations by appending new conditions to them. These rules have the highest information content among all the partial rules formed in each specialisation step. It should be noted that the appended condition has to differ from the conditions already included in the rule to be specialised.

To assess the information content of each newly formed rule, RULES-3 Plus uses a metric called the *H* measure (Lee, 1994). For a particular rule, this measure is defined as:

$$H = \sqrt{\frac{n_{covered}}{N}}[2 - 2\sqrt{\frac{Pn_{class}}{Nn_{covered}}} - 2\sqrt{(1 - \frac{n_{class}}{n_{covered}})(1 - \frac{P}{N})}] \tag{6.1}$$

where $n_{covered}$ is the number of instances covered by the rule, N is the total number of instances, n_{class} is the number of instances covered by the rule and belonging to the target class C_t and P is the number of instances in the training set belonging to the target class C_t.

In Equation (6.1), the first term represents the generality of the rule, and the second term represents its accuracy.

During the rule forming procedure, the rules in *PartialRules* are ordered according to their H measure values. If the H measure value of a newly formed rule is higher than that of any rule in *PartialRules*, the new rule replaces the rule having the lowest H measure value.

The specialisation process can lead to the following three outcomes:

- **No consistent rules.** All rules in *PartialRules* are specialised further by repeating the same process.
- **Only one consistent rule.** The rule is added to the rule set and the search stops.
- **More than one consistent rule.** The rule having the highest value for the H measure is added to the rule set and the search stops.

An example to illustrate the operation of the RULES-3 Plus algorithm can be found in Pham and Dimov (1997a).

RULES-3 Plus deals with attributes having continuous values by dividing the range of each attribute into a fixed number of intervals using the *equal-width* discretisation method. With this method, the number of intervals for each attribute is specified by the user. From the given set of examples, RULES-3 Plus constructs a new set for which the values of all continuous attributes are represented by appropriate intervals. Induction is then carried out with the new set of examples, the intervals being treated as any other value. A pseudo-code description of the RULES-3 Plus algorithm is given in Figure 6.1.

6.2.2 Missing Attribute Values

In many real problems, there can be examples in which the values of some attributes are unknown. Several methods have been developed to overcome this problem (Quinlan, 1989). In RULES-3 Plus, the following procedures are implemented:

- **Create a new condition from the seed example.** If the seed example does not have some attributes, no conditions are created for these attributes.
- **Check if an example is covered by a rule.** If an example does not have attributes for which conditions exist in a rule, the example is considered to be not covered by the rule.

The implementation of these procedures in RULES-3 Plus allows the algorithm to handle missing values.

```
Quantize attributes that have nominal values.                                                    (step 1)
Take a seed example not covered by the rule set formed so far and form array AttributesAndValues. (step 2)
Initialise PartialRules with the most general rule (a rule with no conditions) and set $n_c = 0$.       (step 3)
If $n_c < N_a$ Then                                                                                (step 4)
  $n_c = n_c + 1$
  Initialise T_PartialRules to empty and set $n = 0$.
Else
  Take the example itself as a rule and go to step (7).
Do                                                                                               (step 5)
  $n = n + 1$
  Form a list of rules (T_Rules), the elements of which are combinations of rule $n$ in PartialRules
  with conditions from AttributesAndValues that differ from the conditions already included in rule $n$
  (the number of elements in T_Rules is: $N_a - n_c$) and set $l = 0$.
  Do
    $l = l + 1$
    Compute the $H$ measure for rule $l$ in T_Rules.
    If number of rules in T_PartialRules < $w$ Then
      Store rule $l$ into T_PartialRules.
    Else
      If the $H$ measure of rule $l$ is higher than the $H$ measure of any rule in T_PartialRules Then
        Replace the rule having the lowest $H$ measure in T_PartialRules with rule $l$.
      Else Discard rule $l$.
  While $l < N_a - n_c$
While PartialRules $\neq \varnothing$ AND $n < w$
If there are consistent rules in T_PartialRules Then                                              (step 6)
  Add to the rule list the consistent rule that has the highest $H$ measure and discard the others.
  Mark the examples covered by this rule and go to step (7).
Else Copy T_PartialRules into PartialRules and go to step (4).
If there are no more examples not covered Then Stop.                                              (step 7)
Else Go to step (2).
```

Figure 6.1 A pseudo-code description of RULES-3 Plus.

n_c: number of conditions, ***N_a***: number of attributes, ***w***: number of rules stored in *PartialRules* (the maximum value of w is user-defined), ***T_PartialRules***: a temporary list of rules with the same size as *PartialRules*, and ***T_Rules***: a temporary list of rules.

6.2.3 Classification Procedure

There are three possible outcomes when using the rule set formed by RULES-3 Plus to classify a new example:

- **Only one rule covers the new example.** The example belongs to the class of the covering rule.
- **More than one rule covers the example.** The rule with the highest *H* value is used to classify the example.
- **No rules cover the example.** The class of the "closest" rule in the rule set is assigned to the example. A simple distance measure to find the closest rule is introduced by Bigot (2003) and employed in the RULES-3 Plus algorithm. In the original RULES-3 Plus algorithm, the example is added to the training set and the induction process reinitiated.

6.3 The RULES-6 Algorithm

Although RULES-3 Plus has been successfully employed for different applications as mentioned in section 6.2, several drawbacks have been identified. These drawbacks reduce the applicability of the algorithm to many real-world applications. In this section, based on the ideas presented in the last three chapters, a new rule induction algorithm called RULES-6 (RULe Extraction System – Version 6) is proposed to overcome the drawbacks of RULES-3 Plus. A pseudo-code description of RULES-6 is given in Figure 6.2. Like its predecessors in the RULES family, RULES-6 extracts rules by processing one example at a time. The algorithm first selects a seed example, the first example in the training set not covered by previously created rules, and then calls the *Induce-One-Rule ()*

```
Procedure Induce_Rules (TrainingSet, BeamWidth)
RuleSet = $\emptyset$
While all the examples in the TrainingSet are not covered Do
  Take a seed example s that has not yet been covered.
  Rule = Induce_One_Rule (s, TrainingSet, BeamWidth)
  Mark the examples covered by Rule as covered.
  RuleSet = RuleSet $\cup$ {Rule}
End While
Return RuleSet
End
```

Figure 6.2 A pseudo-code description of RULES-6.

procedure to learn a rule that covers that example. Following this, all covered examples are marked, the learned rule is added to the rule set and the process repeated until all examples in the training set have been covered. The *Induce-One-Rule ()* procedure is outlined in Figure 6.3.

The *Induce-One-Rule ()* procedure searches for rules by carrying out a pruned general-to-specific search. The search aims to generate rules which cover as many examples from the target class and as few examples from the other classes as possible, while ensuring that the seed example remains covered. As a consequence, simpler rules that are not consistent, but are more predictive on unseen data, can be learned. This contrasts with the rule forming procedure of RULES-3 Plus, which restricts its search to only those rules that are completely consistent with the training data, leading to overfitting if the data is noisy.

A beam search is employed to find the best rule. This is done by using two rule lists named *PartialRules* and *NewPartialRules*. *PartialRules*, which is the same size as the beam width w, stores the w best rules during the specialisation process. Only the rules in this list are considered for further specialisation. *NewPartialRules* is used to save valid partial rules obtained by specialising the rules in *PartialRules*. The learning of rules starts with the most general rule whose body is empty (step (1) in Figure 6.3) and specialises it by incrementally adding conditions to its body (step (3) in Figure 6.3). Possible conditions are attribute-value pairs of the selected seed example. In the case of nominal attributes, conditions of the form $[A_i = v_{is}]$ are created, where v_{is} is the value of A_i in the selected seed example s. In the case of continuous attributes, an off-line discretisation

```
Procedure Induce_One_Rule (s: Seed example, Instances: Training set, w: Beam width)
PartialRules = NewPartialRules = ∅
BestRule = most general rule (the rule with no conditions)                          (step 1)
PartialRules = PartialRules ∪ {BestRule}
While PartialRules ≠ ∅ Do                                                            (step 2)
  For each Rule ∈ PartialRules Do
  {First, generate all specialisations of the current rule, save useful ones and determine all the
   InvalidValues according to one of the conditional tests in steps (5), (6) or (7).}
    For each nominal attribute $A_i$ that does not appear in Rule Do
      If $v_{is}$ ∈ Rule.ValidValues, where $v_{is}$ is the value of $A_i$ in s Then
        NewRule = Rule ∧ [$A_i = v_{is}$]                                              (step 3)
        If NewRule.Score > BestRule.Score Then                                        (step 4)
          BestRule = NewRule
        If Covered_Positives (NewRule) ≤ MinPositives OR                              (step 5)
          Covered_Negatives (Rule) – Covered_Negatives (NewRule) ≤ MinNegatives OR    (step 6)
          Consistency (NewRule) = 100% Then                                           (step 7)
          Parent (NewRule).InvalidValues = Parent (NewRule).InvalidValues + {$v_{is}$}  (step 8)
        Else
          NewPartialRules = NewPartialRules ∪ {NewRule}                               (step 9)
    End For
  End For
  Empty PartialRules
```

Figure 6.3 A pseudo-code description of the *Induce_One_Rule ()* procedure of RULES-6.

PartialRules: a list of rules to be specialised, ***NewPartialRules***: a new list of rules to be used for further specialisations and ***T_NewPartialRules***: a temporary list of rules.

```
  For each Rule ∈ NewPartialRules Do
  {Next, delete partial rules that cannot lead to an improved rules and determine all the
   InvalidValues according to the conditional test in step (10).}
    If Rule.OptimisticScore ≤ BestRule.Score Then                                (step 10)
      NewPartialRules = NewPartialRules – {Rule}                                 (step 11)
      Parent (Rule).InvalidValues = Parent (Rule).InvalidValues + Last_Value_Added (Rule)
                                                                                 (step 12)
  End For
  For each Rule ∈ NewPartialRules Do
  {Finally, remove from the ValidValues set of each rule all the values that will lead to
   unnecessary construction of useless specialisations at subsequent specialisation steps.}
    Rule.ValidValues = Rule.ValidValues – Parent (Rule).InvalidValues            (step 13)
  End For
  If $w > 1$ Then
    Remove from NewPartialRules all duplicate rules
  Select $w$ best rules from NewPartialRules and insert into PartialRules        (step 14)
  Remove all rules from NewPartialRules
End While
Return BestRule
End
```

Figure 6.3 A pseudo-code description of the *Induce_One_Rule ()* procedure of RULES-6 (continued).

method (described in section 6.3.2) is used to split the range of each attribute into a number of smaller intervals that are then regarded as nominal values. For each condition, a new rule is formed by appending a condition to the current rule that differs from the conditions already included in the rule. The score of each new rule is computed and the rule with the best accuracy is remembered (step (4) in Figure 2). In RULES-3 Plus, the *H* measure is used to select the best rule. However, this measure is computationally complex and does not lead to the highest level of predictive accuracy and generality. On the other hand, the *m-probability-estimate* (defined in chapter 3, Equation 3.1) is found to produce better results when used in the RULES-6 algorithm. As a result, it is employed as a replacement for the *H* measure.

Some of the new rules are pruned according to one of the conditional tests in steps (5), (6), (7) or (10). The pruned rules are regarded as useless and thus excluded from further specialisations at subsequent specialisation steps. By only considering useful rules at each specialisation step, the search effort for a good rule is effectively restricted, and this significantly speeds the search and improves performance without affecting the quality of the learned rules. The effect of the above pruning is maximised through the additional processing in steps (8), (12) and (13). The conditions for pruning and how their effect is maximised are detailed in chapter 3 (section 3.2.4).

After appropriate rules in the *PartialRules* list are specialised, the best *w* rules from *NewPartialRules* are chosen to replace all rules in the *PartialRules* list (step (14) in Figure 6.3). The comparison between rules is based on the employed quality measure. In the case of a tie, the simplest rule is selected. If there is a tie again, the general rule that covers the highest number of examples not already covered by rules formed so far is chosen. Unlike RULES-3 Plus, rules are not ordered when inserted into *NewPartialRules*, which accelerates the specialisation process. Scanning the *NewPartialRules* list *w* times to obtain the best *w* rules is much faster than ordering the rules in *NewPartialRules* each time a new rule is inserted into it.

Further specialisation ceases when the *PartialRules* list becomes empty (step (2) in Figure 6.3) due to the tests at steps (5), (6), (7) and (10). The best rule is returned at the end.

The following sections discuss the key ideas underlying RULES-6 in more detail.

6.3.1 The Search Method

As presented in section 6.2, RULES-3 Plus accepts only fully consistent rules and creates a complete cover. In real-world applications, however, data is often noisy and insisting on full completeness and consistency of the rule set is no longer essential. This section introduces a novel noise-tolerant search method, which enables the efficient generation of a more accurate and compact rule set.

6.3.1.1 Relaxing the consistency requirement

As mentioned previously, RULES-3 Plus uses a beam search strategy to find near-optimal generalisations of a seed example. After a set of consistent rules has been built, the best one is selected and added to the rule set. Once the best consistent rule that covers the seed example has been selected, the rule generation process stops. Thus, the result of this process is a rule set which is totally consistent with the training data.

One change to relax the consistency requirement would be to continue the rule generation process to the end (i.e., until the *PartialRules* list becomes empty (step (2) in Figure 6.3) and then choose the rule with the highest value of the quality measure as the best one. In this way, rules are allowed to cover a few negative examples, sacrificing the consistency with regard to the training data. In return, simplicity of rules and often better performance are gained. It should be noted that consistent rules having a very low coverage might be found in the early stages of the rule generation process and stopping the search process once a consistent rule has been found might lead to the generation of non-optimal rules. On the other hand, if the search process continues, more general rules could be created.

Another way to introduce inconsistency is through the use of post-pruning techniques, described in section 6.3.3. By dropping certain conditions and rules from the rule set, the resulting set will generally have lower consistency but may have a much higher coverage value and will score higher on future examples.

6.3.1.2 Learning incomplete rule sets

In simple machine learning problems, a complete rule set – one that covers all of the positive examples – is usually desired (Michalski and Kaufman, 2001). In any real-world data mining application, the data often contains errors and full completeness is not required, as overly complex and detailed rules may otherwise be produced. Such "complete" rules may depend strongly on the particular training set and, consequently, may perform poorly in classifying unseen examples. For that reason, when learning from noisy data, it is often better to produce rules that are only partially complete.

In order to relax the completeness requirement, stopping criteria are often used to halt the search for rules before all the training examples are covered. Such halting occurs when no further good rules can be found. This approach has been implemented in the SRI algorithm, described in chapter 3. However, it is risky to implement such an approach in RULES-6. This is because, with randomly selected seed examples from the set of examples yet to be covered, it is not certain that the most general rules will be generated first as the coverage level of the rules does not decrease consistently.

The approach followed in RULES-6 is to continue generating rules until a complete rule set is built. After that, a post-pruning technique is applied to remove rules that have low coverage. However, this approach is computationally expensive.

6.3.2 The Discretisation Method

As presented in chapter 4, there are two approaches for discretisation of continuous attributes, namely, on-line and off-line discretisation. The on-line discretisation method developed in chapter 4 was shown to outperform a number of off-line discretisation methods when incorporated into the SRI algorithm. However, there are at least two problems that make this method inappropriate if it is applied in the RULES-6 algorithm.

Recall first that, in order to discretise a continuous attribute during learning, SRI divides the range of that attribute into two intervals by selecting one threshold value from the attribute domain. The best threshold for a certain attribute is chosen as the one that best separates the classes of the training instances belonging to that attribute. Two possible conditions are then created and new rules are formed by adding these conditions to the current rule. On the other hand, the specialisation process of RULES-6 aims to cover a specific seed example. By adopting this discretisation method, only one condition that covers the seed example will therefore be considered by RULES-6 for specialisation of the current rule and the other condition will be deleted, resulting in a loss of valuable information. It should also be noted that more appropriate conditions might be produced if the objective of covering the seed example is taken into consideration when selecting the best threshold value. Second, as explained in chapter 4, the main advantage of on-line discretisation, in comparison with off-line discretisation, is that higher-order correlations between attributes can be discovered in the learning process. However, the limited search of RULES-6, due to its dependence on specific training examples during its search, reduces the possibility of finding such correlations. As a result, the high computational cost resulting from discretising continuous attributes during learning is not justifiable. In this case, it would be better to employ an off-line discretisation method.

Several off-line discretisation methods have been developed. The experimental results of many studies (Ventura and Martinez, 1995; Cai, 2001) have indicated that the choice of which discretisation method to use depends both on the data to be discretised as well as on the learning algorithm. This section presents a study of the performance of the four off-line discretisation methods used in the experiments conducted in chapter 4 (section 4.4) in order to identify the most appropriate method to be applied in RULES-6.

Experiments were carried out on the same data sets described earlier in Table 4.1, chapter 4. All the data sets were first discretised using each of the discretisation methods and then passed to the learning algorithm. Table 6.1 summarises the results obtained for each discretisation method when used in RULES-6. From the table, it can be seen that the entropy method obtained better accuracy in total than the other methods over the 26 test domains. Moreover, it produced in total significantly fewer rules than the other discretisation methods. In terms of the total execution times, the entropy method was faster than the 1R Discretizer and the equal-width method, but slower than the optimum method. However, the time required by the optimal method to preprocess the data was significantly larger than that required by the entropy method. Overall, the entropy method seems to be the best choice among the descretisation methods tested. Consequently, this discretisation method is adopted for use with RULES-6.

By comparing the results from Table 6.1 with the corresponding results in Table 4.2, it can also be seen that the effect of the off-line discretisation methods was different for the two learning algorithms. For example, the 1R Discretizer method did quite well when combined with RULES-6, but not so well when combined with SRI. Thus, it can be concluded that the choice of the best discretisation method is dependent on the learning algorithm for which the preprocessing is being performed. This supports the findings in (Ventura and Martinez, 1995) and (Cai, 2001).

Data Set Name	Optimum				Entropy				IRD				Equal-width			
	Acc. (%)	# Rules	# Rules explored	Time (s)	Acc. (%)	# Rules	# Rules explored	Time (s)	Acc. (%)	# Rules	# Rules explored	Time (s)	Acc. (%)	# Rules	# Rules explored	Time (s)
Abalone	24.7	21	1250	1	**25.3**	21	1012	1	24.7	**20**	**189**	**0**	24.8	22	1235	1
Anneal	**95.7**	22	2499	**1**	93.3	**16**	1912	**1**	94.3	**16**	**1739**	**1**	92.0	21	2772	2
Australian	81.3	31	3447	0	85.2	**29**	2892	0	**89.1**	**29**	**1864**	0	85.7	34	3179	0
Auto	56.5	**11**	**738**	0	**62.3**	14	1582	0	59.4	17	1117	0	49.3	19	1225	0
Balance-scale	74.6	21	261	1	64.6	**11**	**155**	**0**	65.6	**11**	161	**0**	**79.9**	39	363	1
Breast	92.7	14	261	0	92.3	**10**	**257**	0	**97.4**	12	380	0	93.6	18	278	0
Cleve	76.9	16	838	0	**82.2**	17	913	0	**82.2**	20	1031	0	79.7	**10**	**488**	0
Crx	**81.7**	30	3233	1	79.5	34	3624	1	80.0	26	**2121**	**0**	77.5	**25**	2457	1
Diabetes	65.6	13	559	**0**	**71.5**	**12**	**305**	**0**	65.2	38	907	**0**	68.4	42	1779	1
German	**75.7**	95	10398	**3**	**75.7**	**77**	**8402**	**3**	72.2	104	10630	4	70.9	103	14432	5
German-org.	74.6	70	11166	5	**76.6**	**58**	9684	4	76.0	62	**9545**	**3**	72.7	101	18283	7
Glass2	74.5	6	67	0	**78.2**	**5**	**43**	0	72.7	17	248	0	72.7	16	380	0
Heart-disease	77.8	18	730	0	83.3	**16**	**725**	0	**84.4**	20	999	0	82.2	23	1248	0
Heart-Hungarian	**79.6**	12	476	0	**79.6**	**11**	**396**	0	78.6	13	604	0	77.6	20	1031	0
Hepatitis	**84.6**	11	569	0	82.7	11	**519**	0	**84.6**	11	555	0	76.9	**10**	581	0
Horse-colic	73.2	**28**	3238	1	**80.9**	31	3902	1	75.0	34	**3011**	1	70.6	38	3840	1
Hypothyroid	95.7	**17**	1257	3	95.5	**17**	**1000**	**2**	**96.7**	23	1561	3	95.8	37	4216	8
Ionosphere	92.3	20	1899	**0**	**94.0**	**15**	1588	**0**	93.2	25	**1063**	**0**	92.3	29	2778	1
Iris	**96.0**	**4**	17	0	**96.0**	**4**	20	0	**96.0**	**4**	**15**	0	94.0	6	23	0
Lymphography	74.0	**11**	**661**	0	**86.0**	15	882	0	74.0	**11**	**661**	0	84.0	14	884	0
Segment	**93.1**	**41**	3136	**3**	89.6	42	3529	**3**	79.6	90	**2339**	**3**	87.3	50	4912	5
Shuttle	98.9	**19**	**503**	**28**	**99.7**	27	1101	46	94.7	55	1327	50	89.9	25	1001	43
Sick-euthyroid	95.5	**21**	1776	3	**97.2**	22	**1678**	**2**	94.9	30	3070	3	89.9	32	3834	7
Sonar	72.9	**13**	**921**	**0**	70.0	**13**	**921**	**0**	67.4	31	3263	1	**75.7**	29	4619	1
Tokyo	90.8	**16**	**1285**	2	89.4	19	3673	2	87.1	27	3667	**1**	**91.9**	30	3715	2
Vehicle	**70.6**	40	**3546**	**1**	68.1	**31**	4032	**1**	61.7	68	5002	**1**	63.8	60	6650	2
Total	2069.7	621	**54731**	**53**	**2098.8**	**578**	54747	67	2046.8	814	57069	71	2038.9	853	86203	88

Table 6.1 Performance of descritisation methods as pre-processors to RULES-6.

6.3.3 The Pruning Technique

In chapter 5, three new pruning techniques and their application to the SRI algorithm were described. The aim of this section is to discuss the appropriateness of these techniques for use in the RULES-6 algorithm. Incremental and hybrid pruning techniques cannot be applied to rule sets created by RULES-6 because of the dependence of their pruning strategies on stopping heuristics. The use of stopping heuristics in RULES-6 is not appropriate as explained in section 6.3.1.2. In that section, it was suggested that post-pruning should be used to improve the performance of RULES-6 in the presence of noisy data. The post-pruning technique given in chapter 5 (MDL_PP) is therefore adopted for use in RULES-6. One modification is that, instead of employing a greedy strategy for deleting rules until no further deletion can improve the total coding length, each rule is examined to see whether it could be omitted or not. If a rule cannot be omitted, it is simplified using the delete-condition operator. A pseudo-code of RULES-6 with the post-pruning technique is presented in Figure 6.4. The *Prune_Rule_Set ()* procedure simplifies the rules for each class in turn.

Note that the problem mentioned in chapter 5 (section 5.2), namely that post-pruning techniques are incompatible with the search strategy employed in rule induction systems, does not apply to the RULES-6 algorithm. This is because, in RULES-6, all of the training set is taken into account each time a new rule is formed as mentioned in section 6.2.1. Consequently, all rules are independent and each can be pruned without affecting the rest of the rule set.

```
Procedure Induce_Rules (TrainingSet, BeamWidth)
RuleSet = $\emptyset$
While all the examples in the TrainingSet are not covered Do
  Take a seed example $s$ that has not yet been covered.
  Rule = Induce_One_Rule ($s$, TrainingSet, BeamWidth)
  Mark the examples covered by Rule as covered.
  RuleSet = RuleSet $\cup$ {Rule}
End While
RuleSet = Prune_ Rule_Set (RuleSet, TrainingSet)
Return RuleSet
End
```

Figure 6.4 A pseudo-code description of RULES-6 with MDL_PP.

6.4 Empirical Evaluation of RULES-6

This section describes an empirical study testing RULES-6 with the MDL_PP technique against RULES-3 Plus and C5.0. The algorithms were tested on the same data sets given in Table 5.2, chapter 5 as well as the *Adult* data set (see appendix A for details). The experimental method described in chapter 3 (section 3.4) was used for estimating the performance criteria. RULES-3 Plus was executed with a value of 4 for beam width. RULES-6 was tested using parameter values of 4, 1 and 2 for beam width, MinNegatives and MinPositives respectively. C5.0 was run with the default settings. Tables 6.2 and 6.3 list the results obtained.

As can be seen from Table 6.2, the performance obtained by RULES-6 with MDL_PP was impressive. There were considerable improvements in classification accuracy for 25 data sets. For the *Breast-cancer*, *German-organisation*, *Glass2, Hepatitis, Horse-colic, Promoter, Shuttle* and *Sick-euthyroid* data sets the improvements were most obvious. The accuracy degraded for 12 data sets. For the remaining 2 data sets, equivalent results were obtained. It can also be seen from the table that RULES-6 with MDL_PP produced much more compact rule sets than RULES-3 Plus. The total number of rules decreased by 93.7% from 10739 to 677. Also, the total number of conditions dropped by 98.0% from 93794 to 1874. The reduction in the number of rules and number of conditions for the *Adult* data set was particularly notable. The fewer and more general rules created by RULES-6 made it much faster than RULES-3 Plus as indicated in Table 6.2. The total number of evaluations fell by 94.9% from 3168403 to161726 and this was accompanied by a total 97.8% reduction in the execution time from 31351 seconds to 700 seconds.

	RULES-3 Plus					RULES-6 with MDL_PP				
Data Set Name	Acc. (%)	# Rules	# Conditions	# Rules explored	CPU Time (s)	Acc. (%)	# Rules	# Conditions	# Rules explored	CPU Time (s)
Abalone	18.5	313	1947	26853	28	**23.9**	**12**	**27**	**1012**	**1**
Adult	77.5	6686	70144	1986685	29938	**82.6**	**53**	**141**	**16193**	**435**
Anneal	**99.7**	37	119	10119	3	98.3	**16**	**35**	**1912**	**1**
Australian	83.9	148	807	26301	4	**86.5**	**7**	**18**	**2892**	**0**
Auto	**62.3**	48	94	5534	0	59.4	**11**	**31**	**1582**	0
Balance-scale	**77.0**	213	691	3341	1	72.7	**9**	**21**	**155**	**0**
Breast	**95.7**	40	94	2023	1	95.3	**8**	**17**	**257**	**0**
Breast-cancer	68.4	86	284	5674	1	**77.9**	**12**	**31**	**1311**	**0**
Car	88.4	165	801	7826	2	**90.1**	**35**	**114**	**1374**	**1**
Chess	**99.0**	108	2164	176109	347	97.7	**21**	**79**	**8728**	**19**
Cleve	77.7	33	73	2214	0	**80.2**	**11**	**25**	**913**	0
Crx	80.0	142	863	30277	4	**83.0**	**10**	**27**	**3624**	**1**
Diabetes	66.8	190	739	12399	2	**72.3**	**5**	**8**	**305**	**0**
German	70.9	247	1043	57120	13	**74.9**	**29**	**95**	**8402**	**3**
German-organisation	66.4	252	1381	90770	29	**76.0**	**23**	**103**	**9684**	**4**
Glass2	69.1	46	154	2894	0	**81.8**	**4**	**5**	**43**	0
Heart-disease	**81.1**	60	158	4985	1	80.0	**8**	**18**	**725**	**0**
Heart-Hungarian	72.4	48	196	5611	1	**77.6**	**2**	**3**	**396**	**0**
Hepatitis	61.5	25	47	2023	0	**84.6**	**6**	**13**	**519**	0
Horse-colic	75.0	91	223	12526	1	**85.3**	**6**	**19**	**3902**	1
Hypothyroid	94.9	138	1743	88221	164	**98.3**	**10**	**19**	**1000**	**3**
Ionosphere	92.3	48	94	7654	2	**94.9**	**13**	**31**	**1588**	**1**
Iris	94.0	13	25	122	0	**96.0**	**4**	**5**	**20**	0
Lymphography	80.0	26	56	2431	0	**84.0**	**11**	**25**	**882**	0
Monk1	100.0	22	61	759	0	100.0	22	61	**652**	0
Monk2	**98.8**	262	1504	13709	1	77.2	**38**	**142**	**1572**	**0**
Monk3	95.1	12	23	270	0	95.1	12	23	**263**	0
Mushroom	**100.0**	**25**	**37**	**1556**	**5**	99.7	27	80	2779	15
Promoter	74.3	14	26	3481	1	**82.9**	**5**	**8**	**1146**	**0**
Segment	**90.5**	172	1198	51880	35	88.7	**33**	**82**	**3136**	**3**
Shuttle	91.7	63	289	4689	87	**99.8**	**52**	**100**	**1927**	**61**
Sick-euthyroid	89.4	195	3119	154065	291	**97.2**	**9**	**23**	**1678**	**3**
Sonar	68.6	37	67	9293	1	**74.3**	**2**	**3**	**921**	**0**
Soybean-large	**93.9**	76	542	46253	13	86.0	**22**	**51**	**3953**	**1**
Splice	**91.8**	239	1127	209203	340	89.8	**67**	**190**	**66354**	**144**
Tic-tac-toe	94.7	89	374	7970	1	**98.1**	**29**	**100**	**1757**	**0**
Tokyo	91.3	83	478	48551	27	**92.9**	**6**	**15**	**3673**	**2**
Vehicle	59.6	214	875	42013	7	**66.0**	**22**	**76**	**4032**	**1**
Vote	**97.0**	33	134	4999	0	95.6	**5**	**10**	**464**	0
Total	3189.0	10739	93794	3168403	31351	**3296.4**	**677**	**1874**	**161726**	**700**

Table 6.2 Results for RULES-3 Plus and RULES-6 with MDL_PP.

Data Set Name	C5.0		RULES-6 with MDL_PP	
	Acc. (%)	No. of Rules	Acc. (%)	No. of Rules
Abalone	23.4	522	**23.9**	**12**
Adult	**86.4**	100	82.6	**53**
Anneal	93.3	**11**	**98.3**	16
Australian	**87.4**	20	86.5	**7**
Auto	**62.3**	23	59.4	**11**
Balance-Scale	**81.3**	19	72.7	**9**
Breast	95.0	9	**95.3**	**8**
Breast-cancer	75.8	17	**77.9**	**12**
Car	**91.8**	58	90.1	**35**
Chess	97.2	21	**97.7**	21
Cleve	77.2	13	**80.2**	**11**
Crx	**84.5**	23	83.0	**10**
Diabetes	70.7	14	**72.3**	**5**
German	72.7	**15**	**74.9**	29
German-org	71.8	**17**	**76.0**	23
Glass2	69.1	9	**81.8**	**4**
Heart	78.9	12	**80.0**	**8**
Heart-Hungarian	74.5	7	**77.6**	**2**
Hepatitis	76.9	**5**	**84.6**	6
Horse-colic	83.8	10	**85.3**	**6**
Hypothyroid	94.8	**5**	**98.3**	10
Ionosphere	89.7	**6**	**94.9**	13
Iris	92.0	5	**96.0**	**4**
Lymphography	76.0	**7**	**84.0**	11
Monk1	100.0	**17**	100.0	22
Monk2	65.7	**1**	**77.2**	38
Monk3	**100.0**	**6**	95.1	12
Mushroom	**99.8**	**10**	99.7	27
Promoter	74.3	7	**82.9**	**5**
Segment	**93.4**	**24**	88.7	33
Shuttle	**99.9**	**12**	99.8	52
Sick-euthyroid	90.4	**8**	**97.2**	9
Sonar	74.3	11	74.3	**2**
Soybean-large	**93.4**	32	86.0	**22**
Splice	**92.7**	**60**	89.8	67
Tic-tac-toe	92.2	34	**98.1**	**29**
Tokyo	92.3	8	**92.9**	**6**
Vehicle	**69.9**	46	66.0	**22**
Vote	**97.0**	5	95.6	5
Total	3242.0	1229	**3296.4**	**677**

Table 6.3 Results for C5.0 and RULES-6 with MDL_PP.

These results confirm that RULES-6 with MDL_PP is more robust with respect to noise and more accurate than RULES-3 Plus.

It is clear from Table 6.3 that the accuracy obtained by RULES-6 with MDL_PP was in total higher than that of C5.0. In addition, RULES-6 with MDL_PP achieved higher accuracies for 23 out of 39 data sets, while C5.0 yielded better accuracies for 14 out of 39 data sets. Both algorithms achieved similar accuracies for the remaining 2 data sets. It is also clear from the table that RULES-6 with MDL_PP gave fewer rules in total than C5.0 did. The number of rules was lower for 22 data sets and higher for 15 data sets for RULES-6 with MDL_PP when compared with C5.0.

6.5 Summary

This chapter has presented RULES-6, a simple rule induction algorithm designed for the efficient extraction of comprehensible IF-THEN rules in domains where noise may be present. It employs a simple but robust rule search method which relaxes the search for consistency and reduces the problem of overfitting the induced rules to the training data. It also adopts an alternative method for continuous attributes handling based on a comparative study of the most frequently used methods. Finally, RULES-6 uses a new MDL-based post-pruning technique to address the problem of dealing with noisy data.

RULES-6 with MDL_PP has been empirically compared with its immediate predecessor RULES-3 Plus using a large number of real-world data sets from the UCI machine learning repository. The results have shown that RULES-6 with MDL_PP is better than

RULES-3 Plus in terms of accuracy in classifying unseen instances, size of the generated rule sets and learning time.

In addition, a comparison with the commercial software C5.0 has been made. The results have indicated that RULES-6 in combination with MDL_PP outperforms C5.0.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This chapter summarises the main contributions and the conclusions reached in this work. It also provides suggestions for future work.

7.1 Contributions

This research addressed the problem of scaling up rule induction algorithms so that they can be successfully applied to large data sets. Its contributions include:

- *A thorough analysis of the issue of scaling up inductive learning techniques.* A critical study of the currently available inductive learning techniques and a discussion of their usefulness for data mining applications were made. This led to the design of learning algorithms that can handle large-scale data.

- *An efficient heuristic-search method for rule learning algorithms.* The proposed method employed advanced search heuristics, optimisation techniques and search-space pruning strategies that significantly reduced search time.

- *An on-line discretisation method for rule learners.* The new method took advantage of the bias inherent in the learning algorithm to improve its performance. It produced simpler and more accurate rule sets as well as comparable run-time efficiency when incorporated into a rule learning system.

- *Incremental, hybrid and post-pruning techniques for rule induction algorithms.* The presented techniques were built on the theoretically sound Minimum Description Length (MDL) principle. They adopted an alternative MDL-based formula for rule sets based on the ideas of Quinlan (1995). This formula overcame drawbacks in the formula employed in C4.5. The improved version of the MDL principle and its new usage resulted in a significant reduction in execution time and rule-set sizes as well as an improved accuracy.

- *Classification rule induction algorithms appropriate for data mining.* Two new scalable rule induction algorithms were developed, SRI and RULES-6. SRI follows the approach of CN2-like learning algorithms and is based on the integration of the three techniques mentioned above. Such integration led to efficient and effective induction of classification rules from large data sets. RULES-6 is an improved version of the RULES-3 Plus algorithm which follows the approach of AQ-like learning algorithms. The main advantageous features of RULES-6 over RULES-3 Plus are the relaxation of the requirement for a perfect training set, the more efficient noise-tolerant search method and the elegant handling of continuous and noisy data. Enriched with these new features, RULES-6 should be a powerful practical tool for data mining applications.

- *Demonstration of improved inductive performance.* A comprehensive empirical evaluation of all the algorithms and techniques presented in this thesis was made. The results obtained demonstrated the significant performance improvements achieved.

7.2 Conclusions

Rule induction as a method for constructing classifiers is particularly attractive in data mining applications, where the comprehensibility of the generated models is very important. Most existing algorithms were designed for small data sets and thus are not practical for direct use on very large data sets because of their computational cost. Scaling up rule induction algorithms to handle such data sets is a formidable challenge. This study presented new algorithms for rule induction that can efficiently extract accurate and comprehensible models from large and noisy data sets. These algorithms were tested on several complex real-world data sets and the results proved that they scaled up well and were extremely effective learners.

Chapter 3 focused on developing a scalable rule induction algorithm that is suitable for data mining applications. Rule induction extracts IF-THEN rules directly from the data. The proposed algorithm presented a new search method, which employs several novel search-space pruning rules and rule evaluation techniques. This not only minimised the search space, but also considerably improved the whole induction process.

Handling of both nominal and continuous attributes is a central issue for practical applications of classification learning. Most of the work on discretisation of continuous-valued attributes has been done under the framework of decision trees. Very little work has been done in treating continuous-valued attributes and nominal attributes in a consistent manner in the framework of inductive rule learning systems. The objective of chapter 4 was to explore a way in which continuous-valued attributes and nominal attributes can be treated consistently in inductive rule learning systems. To reach this objective, the discretisation of continuous attributes in the context of inductive learning was studied in detail. Current discretisation methods were categorised based on predefined properties, such as supervised vs. unsupervised, multivariate vs. univariate, etc. A new on-line discretisation method was presented, aiming to take into account the bias underlying the rule learning algorithm. Experiments indicated that the proposed method substantially improved performance.

When learning is based on noisy data, the induced rule sets have a tendency to overfit the training data, and this degrades the performance of the resulting classifier. Consequently, the ability to tolerate noise is a necessity for robust, practical learning algorithms. Pruning is a common way of handling noisy data. Based on an analysis of the existing techniques for pruning rule sets, chapter 5 introduced three new pruning techniques. These techniques were built on the sound foundation of the Minimum Description Length (MDL) principle. The proposed pruning techniques have the advantage that they do not require the set of examples employed for pruning to be distinct from the set used to build the rule set. The new techniques were tested in the SRI rule induction algorithm and showed an improved performance.

RULES-3 Plus is a member of the RULES family of simple inductive learning algorithms, which has proven useful in several manufacturing and engineering applications. However, it requires modification in order to be a practical tool for data mining applications. In particular, mechanisms for effectively handling continuous attributes and noisy data are needed. Chapter 6 presented a new rule induction algorithm called RULES-6, derived from the RULES-3 Plus algorithm. The innovation in RULES-6 is that it has the ability to handle noise in the data, which is achieved by employing a search method that tolerates inconsistency in the rule specialisation process and by using a post-pruning technique that removes unreliable components from the generated rules at the expense of the completeness and/or consistency. This makes the rule sets extracted by RULES-6 both more accurate and substantially simpler than those produced using RULES-3 Plus. RULES-6 also employs effective search-space pruning rules to avoid useless specialisations and to terminate a non-productive search during rule construction. This substantially increases the efficiency of the learning process. Finally, RULES-6 adopts a robust method for handling attributes with continuous values, which further improves the performance of the algorithm. The new features of RULES-6 make it not only more robust and effective but also more efficient, thus enhancing the usefulness of the algorithm for data mining applications.

7.3 Future Work

This section discusses some of the ways in which the methods and algorithms developed in this thesis could be enhanced.

- The rule induction algorithms developed in this work, SRI and RULES-6, employ heuristic search techniques with different learning biases and rule-space pruning strategies that significantly reduce the proportion of the search space examined during the learning process, resulting in substantial performance benefits. An area for further research is the investigation of the learning biases and the evaluation of their effect on the performance of the learning algorithms. Additional rule-space pruning strategies could be considered to improve the performance of the learning algorithms. It may also be possible to speed up the search process even more through the use of efficient data structures (e.g., bit vectors (Segal, 1997)), which reduce the amount of CPU time required to process each example, and through optimisation techniques (e.g., bookkeeping techniques (Aronis and Provost, 1997; Graefe et al., 1998)), which concentrate on eliminating unnecessary components.

- In chapter 4, an on-line method for discretisation of continuous attributes was introduced. Further work could be carried out to increase the efficiency and effectiveness of this method. First, when a continuous attribute is being evaluated, a threshold must be selected. This entails a sorting operation, which must be performed for each attribute in turn, for each rule in the search space. These sorting operations usually account for a large portion of the learning time for large data sets with continuous attributes. This can be avoided by careful bookkeeping: it is only actually necessary to sort the data once for each attribute. The fundamental limitation of this approach is its memory requirement, because the examples covered by a rule need to be stored as example lists. Second, it is possible to generalise the method by extending it to extract multiple intervals, rather than just two, in a single discretisation pass (see, for example, Fayyad and Irani, 1993 and Berka and Bruha, 1996). The motivation for doing this is to obtain rule sets with smaller sizes and higher accuracies.

- The Minimum Description Length (MDL) principle, used in the study in chapter 5, forms the basis of a criterion to evaluate rule quality in the proposed incremental, hybrid and post-pruning techniques. There are two possibilities in this direction worth further exploration. First, a more optimal coding strategy could be attempted. Such a coding strategy could be derived once the strengths and weaknesses of using the MDL principle are better understood. Second, more research could be carried out to combine other criteria with an MDL-based metric in order to overcome the tendency, shown in this work, of pure MDL strategies to over-prune the generated rules.

- Further developments of SRI and RULES-6 could include methods for increasing representational power (e.g., the *attributional calculus* employed in AQ19 (Michalski and Kaufman, 2001), a much richer and highly expressive description language based on variable-valued logic system VL1 (Michalski, 2001)), methods for dealing with missing values (Ramoni and Sebastiani, 2001) and mislabelled training data (Brodley and Friedl, 1999; Mitchell, 1999b), methods for incorporating domain knowledge to control search (Clearwater and Provost, 1990) and alternative conflict resolution schemes and stopping criteria suitable for various trade-offs between accuracy, generality and complexity. Other work that could be considered in this direction include methods for learning in vague environments (e.g., through inducing a set of fuzzy rules from fuzzy data (Wang et al., 1996; Tsang et al., 2000; Drobics and Bodenhofer, 2002; Hoffmann, 2004)) and methods for choosing useful features (Almuallim and Dietterich, 1994; Baluja, 1994; 1995; Caruana and Fritag, 1994; John et al., 1994; Aha and Bankert, 1995; Yang and Honavar, 1997; Hall and Smith, 1998) and constructing new ones, referred to as *constructive induction* (Sethi and Savarajudu, 1982; Rendell, 1989; de raedt, 1992; Kramer, 1994; Wneck and Michalski, 1994), from the available features in order to not only reduce the number of features used by the learning algorithms, but also improve their generalisation ability.

♦ In this work, only one approach of scaling up rule induction algorithms was utilised, namely, designing fast and effective learners. Other approaches that could be considered include *sampling* (Lewis and Catlett, 1994; John and Langley, 1996; Provost et al., 1999), which selects a single subset from the initial data set using different strategies such as random or stratified sampling, *partitioning*, which divides the data into disjoint subsets, learns a model (rule set) from each subset and aggregates the results obtained by either combining the learned models (typically by merging) (Hall et al., 1998) or their predictions (typically by voting) (Chan and Stolfo, 1993; 1997), *incremental batch learning* (Clearwater et al., 1989; Domingos, 1996), which is closely related to partitioning, but processes subsets of examples in sequence, taking advantage of knowledge learned in one iteration to guide learning in a subsequent iteration, *cooperative learning* (Provost and Hennessy, 1994; 1996), which is similar to partitioned-data techniques, but allows cooperation between the group of learners to obtain a global view of the problem and *parallelisation* (Cook and Holder, 1990; Freitas and Lavington, 1998; zaki et al., 1999), which decomposes the search of the rule space such that different processors search different portions of the rule space in parallel.

- More work could be carried out to improve the predictive accuracy of SRI and RULES-6 through the use of techniques such as bagging and boosting (described in chapter 2). Improvements in accuracy should not be obtained at the expense of efficiency and comprehensibility. Examples of work in this area can be found in Domingos (1997a; 1998), Cohen (1999) and Ferri et al., (2002).

- Another promising direction for research is to extend SRI and RULES-6 to perform regression. A number of techniques for doing this are possible, e.g., each rule predicts the average value of the training examples it covers (Breiman et al., 1984) or forms a local linear regression function from those examples (Karalic, 1992). More recently, an efficient fuzzy inductive learning algorithm for continuous output prediction has been developed by Bigot (2003).

APPENDIX A

DATA SETS

All data sets used in this work were obtained from the University of California at Irvine (UCI) repository of machine learning databases (Blake and Merz, 1998). These databases were contributed by many researchers, mostly from machine learning fields, and collected by the Machine Learning group in the University of California, Irvine. These data sets are described below.

Abalone: This data set is used to predict the age of abalone from physical measurements. There are 4177 instances in the data; each is described by 8 attributes.

Adult: There are 48842 instances in this data set. Each instance is described by 14 attributes, such as age, work class, native country, education, marital status, and so on. These attributes are used to predicate whether a person will earn a salary of greater or less than $50,000 in US.

Anneal: This data set concerns appropriate actions to take during coating of steel products. The data set contains 898 instances described in terms of 38 attributes that cover aspects such as the width of the steel, its type, hardness, composition, surface quality etc. There are six classes corresponding to alternative coating sub-procedures.

Australian: This data set is almost the same as the original *Crx* data, but all missing values have been replaced with the medians.

Auto: This data set consists of three types of entities: a) the specification of an auto in terms of various characteristics, b) its assigned insurance risk rating and c) its normalised losses in use as compared to other cars. The second rating corresponds to the degree to which the auto is more risky than its price indicates. Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably quite safe. The third factor is the relative average loss payment per insured vehicle year. This value is normalised for all autos within a particular size classification (two-door small, station wagons, sports/speciality, etc.), and represents the average loss per car per year. The data set contains 205 instances in 6 classes, 10 nominal attributes and 15 continuous attributes.

Balance-scale: This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance.

Breast: This data set contains 699 instances. Each instance is described by 10 continuous attributes. There are two classes, which identify that the tumor is benign or malignant.

Breast-cancer: This data set includes 201 instances of one class (no-recurrence-events) and 85 instances of another class (recurrence-events). The instances are described by 9 nominal attributes that cover aspects such as the age of the patient, tumor size, menopause etc.

Car: This data set is used to evaluate cars according to attributes that describe the price, technical characteristics, and safety of the car. There are 1728 instances; each is described by 6 attributes and can be categorised into one of 4 classes.

Chess: This data set has 36 nominal attributes to describe the board positions, and the task is to determine which position will lead to a win.

Cleve: This data set contains instances on patients who may suffer from heart disease. It contains 303 instances in two classes (healthy or sick), 7 nominal attributes and 6 continuous attributes.

Crx: This data set was originally used by Quinlan in C4.5. The task is to determine whether to give a credit card to an applicant. All the attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.

Diabetes: There are 768 instances in this data set, each is described by 8 continuous attributes, such as number of times pregnant, diastolic blood pressure, body mass index, etc. The data is used to classify whether the patient tests are positive or negative for diabetes.

German: This data set classifies people described by a set of attributes as good or bad credit risks. There are 1000 instances in the data; each is described by 20 attributes of which 13 are nominal.

German-organisation: This data set is similar to the German data set but in a slightly different format.

Glass2: This is a collection of data from crime lab reports. There are 163 instances in this data set; each is described by 9 continuous attributes. The attributes are measures of mineral content such as Mg, Al, et. and refractive index of different samples of glass. The problem is to determine whether the glass was used for windows, container, head lamp, etc.

Heart-disease: This data set contains 13 continuous attributes which have been extracted from a larger set of 75. The class refers to the presence of heart disease in the patient.

Heart-Hungarian: This data set is very close to *Cleve*. It contains 294 instances in two classes, 5 nominal attributes and 8 continuous attributes.

Hepatitis: This data contains 155 instances; each instance is represented by 19 attributes, describing age, sex and 17 other symptoms. The task is to determine whether the patient is at risk of death.

Horse-colic: There are 368 instances in this data set. 22 attributes are used to describe information on the horses, including their age, pulse, rectal temperature etc, and the task is to classify whether a lesion is surgical or not.

Hypothyroid: This data set comes from an assay screening service related to the thyroid function, and concerns one aspect of thyroid diagnosis. The 25 attributes are a mixture of measured values and information obtained from the referring physician. There are two classes (hypothroid, negative).

Ionosphere: This data set concerns classification of radar returns from the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. There are 351 instances; each is described by 34 continuous attributes.

Iris: This is the most widely used data set in the literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. Each instance is described by four continuous attributes, namely, sepal length, sepal width, petal length and petal width.

Lymphography: This data set contains 148 instances in 4 classes (normal find, metastases, malign lymph and fibrosis), 15 nominal attributes and 3 continuous attributes.

Monkl, 2 and 3: The Monk data sets are a collection of three binary classification problems over a six-attribute nominal domain. These data sets include 556, 601 and 554 instances respectively.

Mushroom: This data set consists of descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota family. Each species is identified as definitely edible or definitely poisonous. There are 8124 instances; each instance is described by 22 nominal attributes.

Promoter: This data set consists of 106 instances and 57 nominal attributes representing nucleotides of the DNA sequence (*a*, *t*, *c* or *g*). The task is to decide whether a sequence is a promoter region (+) or not (-).

Segment: This is an image segmentation data drawn randomly from a database of 7 outdoor images. There are 2310 instances; each is described by 19 continuous attributes and can be categorised into one of 7 classes.

Shuttle: This data set contains 58000 instances in 7 classes. Each instance is described by 9 continuous attributes.

Sick-euthyroid: This data set has approximately the same data format and set of attributes as the hypothyroid data set. The classes of this data set are sick-euthyroid and negative.

Sonar: This data set consists of 208 instances and 60 continuous attributes representing measures of the energy within a particular frequency band. The task is to determine whether the sonar image is a rock or a mine.

Soybean-large: This data set consists of 683 instances and 35 nominal attributes describing leaf properties and various abnormalities. The task is to diagnose soybean disease based on the measures and observations.

Splice: There are 3190 instances in this data set. Each instance is described by 61 nominal attributes (the instance name and the sequential DNA nucleotide positions). There are three classes (donors, acceptors and neither).

Tic-tac-toe: This data set encodes the complete set of possible board configurations at the end of tic-tac-toe games. There are 958 instances; each is described by 9 nominal attributes and can be categorised into one of 2 classes.

Tokyo: This is performance co-pilot data for the Tokyo server at SGI. There are 961 instances in two classes (good, bad) and 46 continuous attributes.

Vehicle: This data set consists of 699 instances and 18 continuous attributes. The task is to classify a given silhouette as one of four types of vehicles, using a set of features extracted from the silhouette.

Vote: This data set includes votes for each of the U.S. House of Representatives Congressmen on 16 key votes, such as water project cost sharing, crime and duty-free exports. The problem is to identify whether a person is republican or democrat based on these votes.

APPENDIX B

A PSEUDO-CODE OF THE AQ15 ALGORITHM

```
Let POS be a set of positive examples of class C.
Let NEG be a set of negative examples of class C.

Procedure AQ15 (POS, NEG):
Let COVER be the empty cover.
While COVER does not cover all positive examples in POS Do
    Select a SEED, a positive example previously not covered by COVER.
    Let STAR be STAR (SEED, NEG), a set of maximally general complexes that cover
        SEED but no examples in NEG.
    Let BEST be the best complex in STAR according to the user-defined preference criteria.
    Add BEST as an extra disjunct to COVER.
Return COVER.

Procedure STAR (SEED, NEG):
Let STAR be the set containing the empty complex, which covers the whole domain.
While any complex in STAR covers some negative examples in NEG Do
    Select a negative example $E_{neg}$ covered by a complex in STAR.
    Specialize complexes in STAR to exclude $E_{neg}$ by:
        Let EXTENSION be the set of all selectors that cover SEED but not $E_{neg}$.
        Let STAR be the set $\{x \wedge y \mid x \in \text{STAR}, y \in \text{EXTENSION}\}$.
        Remove all complexes in STAR subsumed by other complexes.
        Remove the worst complexes from STAR until the size of STAR is less than or equal
        to the user-defined maximum star size, maxstar.
Return STAR.
```

Table B.1 The AQ15 algorithm (Clark and Niblett, 1989).

APPENDIX C

THE CONTROL PROCEDURE OF THE CN2 ALGORITHM FOR BOTH ORDERED AND UNORDERED RULES AS WELL AS THE BEAM SEARCH PROCEDURE

```
Procedure CN2_Ordered (EXAMPLES, CLASSES):
Let RULE_LIST be the empty list.
Repeat
    Let BEST_COMPLEX be FindBestComplex (EXAMPLES).
    If BEST_COMPLEX is not null Then
        Let CLASS be the most common class of examples covered by BEST_COMPLEX.
        Add rule "If BEST_COMPLEX Then predict CLASS" to the end of the RULE_LIST.
        Remove from EXAMPLES all examples covered by BEST_COMPLEX.
    End If
Until BEST_COMPLEX is null.
If there are any examples left in EXAMPLES Then
    Let CLASS be the most common class in EXAMPLES.
    Add the default rule "Predict CLASS" to the end of the RULE_LIST.
End If
Return RULE_LIST.
```

Table C.1 The CN2 ordered rules algorithm (Clark and Boswell, 1991).

```
Procedure CN2_Unordered (ALL_EXAMPLES, CLASSES):
Let RULE_SET be the empty list.
For each class in CLASSES Do
    Let RULES be CN2_ForOneClass (ALL_EXAMPLES, CLASS).
    Add RULES to RULE_SET.
End For
Return RULE_SET


Procedure CN2_ForOneClass (ALL_EXAMPLES, CLASS):
Let RULES be the empty set.
Repeat
    Let BEST_COMPLEX be FindBestComplex (EXAMPLES, CLASS).
    If BEST_COMPLEX is not null Then
        Add the rule "If BEST_COMPLEX Then predict CLASS" to RULES.
        Remove from EXAMPLES all examples in CLASS covered by BEST_COMPLEX.
    End If
Until BEST_COMPLEX is null
Return RULES
```

Table C.2 The CN2 unordered rules algorithm (Clark and Boswell, 1991).

```
Procedure FindBestComplex (EXAMPLES [, CLASS][a]):
Let MGC be the most general complex (= "true").
Let STAR be the set containing only MGC (= {MGC}).
Let BEST_COMPLEX be null.
While STAR is not empty Do
    Let NEW_STAR be the empty set (= {}).
    For each COMPLEX in STAR Do
        For each possible attribute test TEST not already tested on in COMPLEX Do
            Let NEW_COMPLEX be a specialization of COMPLEX, formed by adding
              TEST as an extra conjunct to COMPLEX (i.e. NEW_COMPLEX =
              COMPLEX & TEST).
            If NEW_COMPLEX is better than BEST_COMPLEX AND
              NEW_COMPLEX is statistically significant Then
                Let BEST_COMPLEX = NEW_COMPLEX.
                Add NEW_COMPLEX to NEW_STAR.
                If size of NEW_STAR > maxstar (a user-defined constant) Then
                    Remove the worst complex in NEW_STAR.
                End If
            End If
        End For
    End For
    Let STAR = NEW_STAR.
End While
Return BEST_COMPLEX.
```

[a] CLASS is only required for generating unordered rules.

Table C.3 The CN2 rule search algorithm (Clark and Boswell, 1991).

REFERENCES

Agrawal, R., Imielinski, T. and Swami, A. (1993). Mining association rules between sets of items in large databases. *Proc. of ACM SIGMOD Conf. on Management of Data*, Washington D.C., pp. 207-216.

Aha, D. W. (1997). *Artificial Intelligence Review* - Special issue on lazy learning. Vol. 11, pp. 7-10.

Aha, D. W. and Bankert, R. (1995). A comparative evaluation of sequential feature selection algorithms. *Proc. of the 5th Int. Workshop on Artificial Intelligence and Statistics,* Ft, Lauderdale, pp. 1-7.

Aha, D., Kibler, D. and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, Vol. 6, No. 1, pp. 37-66.

Akyol, D. E. (2004). Application of neural networks to heuristic scheduling algorithms. *Computers & Industrial Engineering*, Vol. 46, pp. 679-696.

Almuallim, H. and Dietterich, T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence,* Vol. 69, No. 1-2, pp. 279-305.

An, A. and Cercone, N. (1999). Discretisation of continuous attributes for learning classification rules. *Proc. of the 3rd Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD-99)*, Kyoto, Japan, pp. 509-514.

Apté, C. and Weiss, S. (1997). Data mining with decision trees and decision rules. *Future Generation Computer Systems*, Vol. 13, pp. 197-210.

Aronis, J. and Provost, F. (1997). Increasing the efficiency of data mining algorithms with breadth-first marker propagation. *Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining*, Newport Beach, CA, pp. 119-122.

Auer, p., Holte, R. C. and Maass, W. (1995). Theory and application of agnostic PAC-learning with small decision trees. *Proc. of the 12th Int. Conf. on Machine Learning*, Tahoe City, California, USA, pp. 21-29.

Baluja, S. (1994). Population based incremental learning: A method for integrating genetic search based function optimisation and competitive learning. *Technical Report*, Carnegie Mellon University, CMU-CS-94-163.

Baluja, S. (1995). An empirical comparison of seven iterative and evolutionary function optimisation heuristics. *Technical Report*, Carnegie Mellon University, CMU-CS-95-193.

Barron, A., Rissanen, J. and Yu, B. (1998). The minimum description length principle in coding and modelling. *IEEE Transactions on Information Theory*, Vol. 44, No. 6, pp. 2743-2760.

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

Bergadano, F., Giordana, A. and Saitta, L. (1988). Automated concept acquisition in noisy environments. *IEEE Transactions on Pattern Analysis and Machine Learning*, Vol. 10, pp. 555-578.

Bergadano, F., Matwin, S., Michalski, R. S. and Zhang, J. (1992). Learning tow-tiered descriptions of flexible concepts: The POSEIDON system. *Machine Learning*, Vol. 8, pp. 5-43.

Berka, P. and Bruha, I. (1998). Empirical comparison of various discretization procedures. *Int. J. of Pattern Recognition and Artificial Intelligence*, Vol. 12, No. 7, pp. 1017-1032.

Bigot, S. (2003). *New Techniques for Handling Continuous Values in Inductive Learning*. Ph.D. thesis, Systems Engineering Division, University of Wales, Cardiff, UK.

Birkendorf, A. (1997). On fast and simple algorithms for finding maximal subarrays and applications on computational learning theory. *Lecture Notes in Artificial Intelligence*, Vol. 1208, Heidelberg, Springer-Verlag, pp. 198-209.

Blake, C. L. and Merz, C. J. (1998). *UCI Repository of Machine Learning Databases*. University of California, Department of Information and Computer Science, Irvine, CA. Available from: http://www.ics.uci.edu/~mlearn/MLRepository.html [Accessed: 1 February 2003].

Blockeel, H. and Sebag, M. (2003). Scalability and efficiency in multi-relational data mining. *ACM SIGKDD Explorations Newsletter*, Vol. 5, No. 1, pp. 17-30.

Braha, D. (2001). *Data Mining for Design and Manufacturing: Methods and Applications*. Kluwer Academic Publishers, Boston, MA.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, Vol. 24, No. 2, pp. 123-140.

Breiman, L. (1996b). Stacked regressions. *Machine Learning*, Vol. 24, No. 1, pp. 49-64.

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

Breslow, A. and Aha, D. W. (1996). Simplifying decision trees: A survey. *Knowledge Engineering Review*, Vol. 12, pp. 1-40.

Brodley, C. E. and Friedl, M. A. (1999). Identifying mislabeled training data. *J. of Artificial Intelligence Research*, Vol. 11, pp. 131-167.

Brunk, C. A. and Pazzani, M. J. (1991). An investigation of noise-tolerant relational concept learning algorithms. *Proc. of the 8th Int. Workshop on Machine Learning,* Evanston, Illinois, pp. 389-393.

Buntine, W. (1991). *A Theory of Learning Classification Rules*. Ph.D. Thesis, School of Computer Science, University of Technology, Sydney, Australia.

Cai, Z. (2001). *Technical Aspects of Data Mining*. Ph.D. thesis, Systems Engineering Division, University of Wales, Cardiff, UK.

Caruana, R. and Freitag, D. (1994). Greedy attribute selection. *Proc. of the 11th Int. Conf. on Machine Learning*, New Brunswick, NJ, pp. 28-36.

Catlett, J. (1991a). *Megainduction: Machine Learning on Very Large Databases*. Ph.D. Thesis, School of Computer Science, University of Technology, Sydney, Australia.

Catlett, J. (1991b). On changing continuous attributes into ordered discrete attributes. *Proc. of the European Working Session on Learning*, Porto, Portugal, Springer-Verlag, pp. 164-178.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *Int. J. of Man-Machine Studies*, Vol. 27, pp. 349-370.

Cerquides, J. and Lopez de Mantaras, R. (1997). Proposal and empirical comparison of a parallelizable distance based discretization method. *Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining*, Newport Beach, CA, pp. 139-142.

Cervone, G., Panait, L. A. and Michalski, R. S. (2001). The development of the AQ20 learning system and initial experiments. *Proc. of the 10th Int. Symposium on Intelligent Information Systems*, Poland, pp. 13-29.

Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. *Proc. of the 3rd European Conf. on Artificial Intelligence (ECAI-90)*, pp. 147-149.

Chan, P. and Stolfo, S. (1993). Toward parallel and distributed learning by meta-learning. *Working Notes of AAAI Workshop on Knowledge Discovery in Databases*, pp. 227-240.

Chan, P. and Stolfo, S. (1997). On the accuracy of meta-learning for scalable data mining. *J. of Intelligent Information Systems*, Vol. 8, pp. 5-28.

Chauvin, y. and Rumelhart, D. (1995). *Backprobagation: Theory, Architecture and Applications*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Ching, J. Y., Wong, A. K. C. and Chan, C. C. (1995). Class dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 7, pp. 641-651.

Chmielewski, M. R. and Grzymala-Busse, J. W. (1994). Global discretization of continuous attributes as preprocessing for machine learning. *Proc. of the 3rd Int. Workshop on Rough Sets and Soft Computing*, San Jose, CA, pp. 294-301.

Chryssolouris, G. and Subramaniam, V. (2001). Dynamic scheduling of manufacturing job shops using genetic algorithms. *J. of Intelligent Manufacturing*, Vol. 12, pp. 281-293.

Clark, P. and Boswell, R. (1991). Rule induction with CN2: Some recent improvements. *Proc. of the 5th European Conf. on Artificial Intelligence*, Porto, Portugal, pp. 151-163.

Clark, P. and Niblett, T. (1987). Induction in noisy domains. *Proc. of the 2nd European Working Session on Learning*, Sigma, Wilmslow, UK, pp. 11-30.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, Vol. 3, pp. 261-284.

Clearwater, S., Cheng, T., Hirsh, H. and Buchanan, B. (1989). Incremental batch learning. *Proc. of the 6th Int. Workshop on Machine Learning*, San Mateo, CA, Morgan Kaufmann, pp. 366-370.

Clearwater, S. and Provost, F. (1990). RL4: A tool for knowledge-based induction. *Proc. of the 2nd Int. IEEE Conf. on Tools for Artificial Intelligence*, IEEE Press, pp. 24-30.

Cohen, W. W. (1993). Efficient pruning methods for separate-and-conquer rule learning systems. *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence*, Chambery, France, pp. 988-994.

Cohen, W. W. (1995). Fast effective rule induction. *Proc. of the 12th Int. Conf. on Machine Learning*, Tahoe City, California, USA, pp. 115-123.

Cohen, W. W. and Singer, Y. (1999). A simple, fast and effective rule learner. *Proc. of the 16th National Conf. on Artificial Intelligence,* Menlo Park, CA, AAAI/MIT Press, pp. 335-342.

Cook, D. and Holder, L. (1990). Accelerated learning on the connection machine. *Proc. of the 2nd Int. IEEE Conf. on Tools for Artificial Intelligence*, San Mateo, CA, Morgan Kaufmann, pp. 366-370.

Craven, M. W. and Shavlik, J. W. (1997). Using neural networks for data mining. *Future Generation Computer Systems*, Vol. 13, pp. 211-229.

Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, Vol. 1, pp. 131-156.

Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand, New York.

De Raedt, L. (1992). Interactive concept-learning and constructive induction by analogy. *Machine Learning*, Vol. 8, pp. 107-150.

Devijver, P. A. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. Prentice Hall, Englewood Cliffs, London.

Dietterich, T. G. (1997). Machine learning research: Four current directions. *Artificial Intelligence Magazine*, Vol. 18, No. 4, pp. 97-136.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, Vol. 40, pp. 139–157.

Dietterich, T. G. and Bakiri, G. (1995). Solving multi-class problems via error-correcting output cods. *J. of Artificial Intelligence Research*, Vol. 2, pp. 263-286.

Domingos, P. (1996). Efficient specific-to-general rule induction. *Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, Menlo Park, CA, AAAI Press, pp. 319-322.

Domingos, P. (1997a). Knowledge acquisition from examples via multiple models. *Proc. of the 14th Int. Conf. on Machine Learning*, San Francisco, CA, Morgan Kaufmann, pp. 98-106.

Domingos, P. (1997b) *A Unified Approach to Concept Learning.* Ph.D. Thesis, University of California, Irvine, USA.

Domingos, P. (1998). Knowledge discovery via multiple models. *Intelligent Data Analysis*, Vol. 2, No. 1-4, pp. 187-202.

Domingos, P. and Pazzani, M. (1996). Beyond independence: Conditions for the optimality of the simple Bayesian classifier. *Proc. of the 13th Int. Conf. on Machine Learning*, Bari, Italy, Morgan Kaufmann, pp. 105-112.

Dougherty, J., Kohavi, R. and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Proc. of the 12th Int. Conf. on Machine Learning*, Tahoe City, California, USA, pp. 194-202.

Drobics, M. and Bodenhofer, U. (2002). Fuzzy modeling with decision trees. *IEEE Int. Conf. on Systems, Man and Cybernetics*, Vol. 4, pp. 6-9.

Duch, W., Setiono, R. and Zurada, J. M. (2004). Computational intelligence methods for rule-based data understanding. *Proc. of the IEEE*, Vol. 92, No. 5, pp. 771-805.

Efron, B. and Tibshirani, R. (1993). *An Introduction to the Bootstrap.* Chapman & Hall, USA.

Elomaa, T. and Kääriäinen, M. (2001). An analysis of reduced error pruning. *J. of Artificial Intelligence Research*, Vol. 15, pp. 163-187.

Elomaa, T. and Rousu, J. (1999a). Speeding up the search for optimal partitions. *Lecture Notes in Artificial Intelligence*, Vol. 1704, Springer-Verlag, pp. 89-97.

Elomaa, T. and Rousu, J. (1999b). General and efficient multisplitting of numerical attributes. *Machine Learning*, Vol. 36, No. 3, pp. 201-244.

Esposito, F., Malerba, D. and Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 5, pp. 476-491.

Fayyad, U. M., Haussler, D. and Stolorz, P. (1996). KDD for science data analysis: Issues and examples. *Proc. of the 2nd Int. Conf. on Data Mining and Knowledge Discovery*, Menlo Park, CA, AAAI Press, pp. 50-56.

Fayyad, U. M. and Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, Vol. 8, pp. 87-102.

Fayyad, U. M. and Irani, K. B. (1993). Multi-inteval discretization of continuous-valued attributes for classification. *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence*, Chambery, France, pp. 1022-1027.

Fayyad, U. M., Piatetsky-Shapiro, G. and Smyth, P. (1996a). From data mining to knowledge discovery: An overview. In: *Advances in Knowledge Discovery and Data Mining* (U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, (Eds.)), Menlo Park, CA, AAAI Press.

Fayyad, U. M., Piatetsky-Shapiro, G. and Smyth, P. (1996b). Knowledge Discovery and Data-Mining: Towards a unifying framework *Proc. of the 2nd Int. Conf. on Data-Mining and Knowledge Discovery*, Menlo Park, CA, AAAI Press, pp. 82-88.

Fern, X. Z. and Brodley, C. E. (2003). Boosting lazy decision trees. *Proc. of the 20th Int. Conf. on Machine Learning*, Washington, DC, pp. 178-185.

Fern, A. and Givan, R. (2003). Online ensemble learning: An empirical study. *Machine Learning*, Vol. 53, pp. 71-109.

Ferri, C., Hernández-Orallo, J. and Ramírez-Quintana, M. J. (2002). From ensemble methods to comprehensible models. *Lecture Notes in Computer Science*, Vol. 2534, pp. 165 – 177.

Forsyth, R. S., Clarke, D. D. and Wright, R. L. (1994). Overfitting revisited: An information theoretic approach to simplifying discrimination trees. *J. of Experimental and Theoretical Artificial Intelligence*, Vol. 6, No. 3, pp. 289-302.

Frank, E. (2000). *Pruning Decision Trees and Lists*. Ph.D. Thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand.

Frank, E. and Witten, I. H. (1998). Making better use of global discretization. *Proc. of the 15th Int. Conf. on Machine Learning,* Madison, Wisconsin, USA, pp. 152-160.

Frank, E. and Witten, I. H. (1999). Reduced-error pruning with significant tests. *Working paper*, Department of Computer Science, University of Waikato, Hamilton, New Zealand.

Frayman, Y., Ting, K. M. and Wang, L. (1999). A fuzzy neural network for data mining: Dealing with the problem of small disjuncts. *IEEE Int. Joint Conf. on Neural Networks (IJCNN-99)*, Vol. 4, pp. 2490-2493.

Freitas, A. A. (2002). *Data mining and knowledge discovery with evolutionary algorithms*. Springer-Verlag, Berlin, New York.

Freitas, A. A. and Lavington, S. H. (1998). Mining very large databases with parallel processing. *Kluwer Academic Publishers*, Boston, MA.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. *Proc. of the 13th Int. Conf. on Machine Learning,* pp. 148-156.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalisation of on-line learning and an application to boosting. *J. of Computer and System Sciences,* Vol. 55, No. 1, pp. 119-139.

Friedman, J. H. (1996). On bias, variance, 0/1-loss, and the curse-of-dimensionality, *Technical report,* Department of Statistics and Stanford Linear Accelerator Center, Stanford University, Stanford, CA.

Fulton, T., Kasif, S. and Salzberg, S. (1995). Efficient algorithms for finding multi-way splits for decision trees. *Proc. of the 12th Int. Conf. on Machine Learning,* Tahoe City, California, USA, Morgan Kaufmann, pp. 244-251.

Fürnkranz, J. (1994a). *Efficient Pruning Methods for Relational Learning.* Ph.D. Thesis, Technisch-Naturwissens chaftlichen Fakultät, Techniscen Universität Wien.

Fürnkranz, J. (1994b). Top-down pruning in relational learning. *Proc. of the 11th European Conf. on Artificial Intelligence,* Amsterdam, The Netherlands, pp. 453-457.

Fürnkranz, J. (1996). Pruning algorithms for rule learning. *Machine Learning,* Vol. 27, pp. 139-171.

Fürnkranz, J. (1998). Integrative windowing. *J. of Artificial Intelligence Research,* Vol. 8, pp. 129-164.

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review,* Vol. 13, No. 1, pp. 3-54.

Fürnkranz, J. and Widmer, G. (1994). Incremental reduced error pruning. *Proc. of the 11th Int. Conf. on Machine Learning,* New Brunswick, NJ, pp. 70-77.

Gehrke, J., Ramakrishnan, R. and Ganti, V. (1998). Rainforest – A framework for fast decision tree construction of large datasets. *Proc. of the 24th Int. Conf. on Very Large Data Bases (VLDB)*, New York, USA, pp. 416-427.

Georgeff, M. P. and Wallace, C. S. (1984). A general selection criterion for inductive inference. *Proc. of the 8th European Conf. on Artificial Intelligence (ECAI-88)*, New York, pp. 473-482.

Giudici, P. (2003). *Applied Data Mining: Statistical Methods for Business and Industry*. John Wiley, England.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.

Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, Cambridge, MA.

Graefe, G., Fayyad, U. and Chaudhuri, S. (1998). On the efficient gathering of sufficient statistics for classification of large SQL databases. *Proc. of the 4th Int. Conf. on Knowledge Discovery and Data-Mining*, New York, NY, AAAI Press, pp. 204-208.

Grunwald, P. (2000). Model selection based on minimum description length. *J. of Mathematical Psychology*, Vol. 44, pp. 133-170.

Guha, S., Rastogi, R. and Shim, K. (1998). CURE: An efficient clustering algorithm for large databases. *Proc. of the ACM SIGMOD Conf. on Management of Data*, Seattle, WA, pp. 73-84.

Hall, L. O., Chawla, N. and Bowyer, K. W. (1998). Combining decision trees learned in parallel. *Working Notes of the Knowledge Discovery in Databases (KDD-97) Workshop on Distributed Data Mining*, pp. 10-15.

Hall, M. A. and Smith, L. A. (1998). Practical feature subset selection for machine learning. *Proc. of the 21st Australian Computer Science Conf.*, Perth, Australia, pp. 181-191.

Han, J. and kamber, M. (2001). *Data Mining: Concepts and Techniques*. Academic Press, USA.

Hansen, L. K. and Salamon, P. (1990). Neural Networks ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 10, pp. 993-1001.

Hassoun, M. H. (1995). *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA.

Haykin, S. S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing, New York.

Heckerman, D. (1996). Bayesian networks for knowledge discovery. In: *Advances in Knowledge Discovery and Data Mining* (U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, (Eds.)), Menlo Park, CA, AAAI Press, pp. 273-305.

Ho, K. M. and Scott, P. D. (1997). Zeta: A global method for discretization of continuous variables. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 5, pp. 718-730.

Hoffmann, F. (2004). Combining boosting and evolutionary algorithms for learning of fuzzy classifications rules. Fuzzy Sets and Systems, Vol. 141, pp. 47-58.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used data sets. *Machine Learning*, Vol. 11, pp. 63-90.

Holte, R. C., Acker, L. E. and Porter, B. W. (1989). Concept learning and the problem of small disjuncts. *Proc. of the 11th Int. Joint Conf. on Artificial Intelligence,* Detroit, Michigan, USA, pp. 813-818.

Ip, C. Y., Regli, W. C., Sieger, L. and Shokoufandeh, A. (2003). Automated learning of model classification. *Proc. of the 8th ACM Symposium on Solid Modeling and Applications,* Seattle, Washington, USA, ACM Press, pp. 322-327.

ISL. (1998). *Clementine Data Mining Package*. SPSS UK Ltd., 1st Floor, St. Andrew's House, West Street, Woking, Surrey, United Kingdom.

Jennings, N. R. (1996). Automated visual inspection of engine valve stem seals. *Internal Report*, Systems Engineering Division, University of Wales, Cardiff, UK.

Jiang, Y., Zhou, Z. H., and Chen, Z. Q. (2002). Rule learning based on neural network ensemble. *Proc. of the Int. Joint Conf. on Neural Networks,* Honolulu, HI, pp. 1416-1420.

John, G., Kohavi, R. and Pfleger, K. (1994). Irrelevant features and the subset selection problem. *Proc. of the 11th Int. Conf. on Machine Learning*, New Brunswick, NJ, pp. 121-129.

John, G. and Langley, P. (1996). Static versus dynamic sampling for data minig. *Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, AAAI Press, pp. 367-370.

Jun, B. H., Kim, C. S. and Kim, J. (1997). A new criterion in selection and discretisation of attributes for the generation of decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 12, pp. 1371-1375.

Kalbfleish, J. (1979). *Probability and Statistical Inference*. Vol. 2, Springer-Verlag, New York.

Karalic, A. (1992). Employing linear regression in regression tree leaves. *Proc. of the 6th European Conf. on Artificial Intelligence*, Vienna, Austria, pp. 440-441.

Kerber, R. (1992). ChiMerge: Discretization of numeric attributes. *Proc. of the 10th National Conf. on Artificial Intelligence*, San Jose, CA, pp. 123-128.

Klösgen, W. and Żytkow, J. M. (2002). *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, New York.

Kohavi, R. (1995a*). Wrappers for Performance Enhancements and Oblivious Decision Graphs*. Ph.D. Thesis, Department of Computer Science, Stanford University, Palo Alto, CA.

Kohavi, R. (1995b). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence,* Montreal, Canada, Morgan Kaufmann, pp. 1137-1143.

Kohavi R. and John G. H., (1997), Wrappers for feature subset selection. *Artificial Intelligence J.*, Vol. 97, No. 1, pp. 273-324.

Kohavi, R. and Sahami, M. (1996). Error-based and entropy-based discretization of continuous features. *Proc. of the 2nd Int. Conf. on Knowledge Discovery in Databases (KDD),* Montreal, Canada, AAAI Press, pp. 114-119.

Kontkanen, P., Myllymaki, P., Silander, T. and Tirri, H. (1997). A Bayesian approach to discretisation. *Proc. of the European Symposium on Intelligent Techniques*, Bari, Italy, pp. 265-268.

Kovačič, M. (1994). *Stochastic Inductive Logic Programming*. Ph.D. Thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana.

Kramer, S. (1994). CN2-MCI: A two-step method for constructive induction. *Proc. of the Workshop on Constructive Induction and Change of Representation, 11th Int. Conf. on Machine Learning (ML-94/COLT-94)*, New Brunswick, New Jersey.

Krichevsky, R. E. and Trofimov, V. K. (1983). The performance of universal coding. *IEEE Transactions on Information Theory*, Vol. IT-27, No. 2, pp. 199-207.

Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, Vol. 51, pp. 181-207.

Kurgan, L. and Cios, K. J. (2001). Discretisation algorithm that uses class-attribute independence maximization. *Proc. of the 17th Int. Conf. on Artificial Intelligence (IC-AI 2001),* Las Vegas, Nevada, pp. 980-987.

Langley, P. and Kibler, D. (1988). Machine learning as empirical science. *Machine Learning*, Vol. 3, No. 1, pp. 5-8.

Lavrač, N., Motoda, H., Fawcett, T., Holte, R., Langley, P. and Adriaans, P. (2004). Introduction: Lessons learned from data mining applications and collaborative problem solving. *Machine Learning*, Vol. 57, pp. 13-34.

Lee, C. (1994). Generating classification rules from databases. *Proc. of the 9th Conf. on Application of Artificial Intelligence in Engineering*, PA, USA, pp. 205-212.

Lewis, D. D. and Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. *Proc. of the 11th Int. Conf. on Machine Learning*, New Brunswick, NJ, Morgan Kaufmann, pp. 148-156.

Liu, H., Hussain, F., Tan, C. L. and Dash, M. (2002). Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, Vol. 6, pp. 393-423.

Liu, J. and Kwok, J. T. (2000). An extended genetic rule induction algorithm. *Proc. of the 2000 Congress on Evolutionary Computation*, California, USA, pp. 458-463.

Liu, J. and Setiono, R. (1997). Feature selection via discretisation. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 4, pp. 642-645.

Liu, H. and Setiono, R. (1998). Some issues on scalable feature selection. *Expert Systems with Applications*, Vol. 15, pp. 333-339.

Maass, W. (1994). Efficient agnostic PAC-learning with simple hypotheses. *Proc. of the 7th Annual ACM Conf. on Computational Learning Theory*, New Brunswick, New Jersey, USA, pp. 67-75.

Madigan, D., Raftery, A. E., Volinsky, C. T. and Hoeting, J. A. (1996). Bayesian model averaging. *Proc. of the Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, Portland, OR, AAAI Press, pp. 77-83.

Markham, I. S., Mathieu, R. G. and Wray, B. A. (2000). Kanban setting through artificial intelligence: A comparative study of artificial neural networks and decision trees. *Integrated Manufacturing Systems*, Vol. 11, No. 4, pp. 239-246.

Megiddo, N. and Srikant, R. (1998). Discovering predictive association rules. *Proc. of the 4th Int. Conf. on Knowledge Discovery in Databases and Data Mining*, New York, pp. 274-278.

Mehta, M., Agrawal, R. and Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. *Proc. of the 5th Int. Conf. on Extending Database Technology*, Avignon, France, pp. 18-32.

Mehta, M., Rissanen, J. and Agrawal, R. (1995). MDL-based decision tree pruning. *Proceedings of the 1st Int. Conf. on Knowledge Discovery in Databases and Data Mining,* Montreal, Canada, 216-221.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* 3rd Edition, Springer-Verlag, Berlin.

Michalski, R. S. (1969). On the quasi-minimal solution of the general covering problem. *Proc. of the 5th Int. Symposium on Information Processing,* Vol. A3 (Switching Circuits), Bled, Yugoslavia, pp. 125-128.

Michalski, R. S. (2001). Natural induction: A theory and methodology. *Reports of the Machine Learning and Inference Laboratory, MLI01-1,* George Mason University, Fairfax, VA.

Michalski, R. S. and Kaufman, K.A. (1999). A measure of description quality for data mining and its implementation in the AQ18 learning system. *Proc. of the 1999 Int. ILSC Congress on Computational Intelligence Methods and Applications*, Rochester, NY, USA, pp. 369-375.

Michalski, R. S. and Kaufman, K.A. (2001). The AQ19 system for machine learning and pattern discovery: A general description and user guide. *Reports of the Machine Learning and Inference Laboratory*, MLI 01-2, George Mason University, Fairfax, VA, USA.

Michalski, R. S. and Larson, J. B. (1983). Incremental generation of VL1 hypotheses: The underlying methodology and the descriptions of program AQ11. *ISG 83-5*, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Michalski, R. S., Mozetic, I., Hong, J. and Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains.

American Association of Artificial Intelligence, Los Altos, CA, Morgan Kaufmann, pp. 1041-1045.

Michie, D., Spiegelhalter, D. J. and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, New York.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, Vol. 4, pp. 227-243.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill, New York.

Mitchell, T. M. (1999a). Machine learning and data mining. *Communications of the ACM*, Vol. 42, No. 11, pp. 31-36.

Mitchell, T. M. (1999b). The role of unlabeled data in supervised learning. *Proc. of the 6th Int. Colloquium on Cognitive Science (ICCS-99),* San Sebastian, Spain, pp. 1-8.

Monostori, L. (2002). AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing. *Proc. of the 15th Triennial World Congress*, Barcelona, Spain, pp. 119-130.

Moore, A. and Lee, M. (1998). Cached sufficient statistics for efficient machine learning with large datasets. *J. of Artificial Intelligence Research,* Vol. 8, pp. 67-91.

Morimoto, Y., Fukuda, T., Matsuzawa, H., Tokuyama, T. and Yoda, K. (1998). Algorithms for mining association rules for binary segmentations of huge categorical databases. *Proc. of the 24th Int. Conf. on Very Large Data Bases (VLDB), New York, USA*, pp.380-391.

Muggleton, S. (1995). *Foundations of Inductive Logic Programming.* Prentice Hall, Englewood Cliffs, NJ.

Opitz, D. W. (1999). Feature selection for ensembles. *Proc. of the 16th National Conf. on Artificial Intelligence*, Orlando, FL, pp. 379-384.

Öztürk, N. and Öztürk, F. (2004). Hybrid neural network and genetic algorithm based machining feature recognition. *J. of Intelligent Manufacturing*, Vol. 15, pp. 278-298.

Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, Vol. 3, pp. 71-99.

Peng, Y. (2004). Intelligent condition monitoring using fuzzy inductive learning. *J. of Intelligent Manufacturing*, Vol. 15, pp. 373-380.

Pérez, E., Herrera, F. and Hernández, C. (2003). Finding multiple solutions in job shop scheduling by niching genetic algorithms. *J. of Intelligent Manufacturing*, Vol. 14, pp. 323-339.

Perner, P. and Trautzsch, S. (1998). Multi-interval discretization methods for decision tree learning. In: *Advances in Pattern Recognition* (A. Amin, D. Dori, P. Pudil and H. Freeman, (Eds.)), Vol. 1451 of LNCS, Springer-Verlag, pp. 475-482.

Pfahringer, B. (1995a). Compression-based discretization of continuous attributes. *Proc. of the 12th Int. Conf. on Machine Learning,* Tahoe City, California, USA, Morgan Kaufmann, pp. 339-350.

Pfahringer, B. (1995b). *Practical Uses of the Minimum Description Length Principle in Inductive Learning.* Ph.D. Thesis, Institut Für Med.Kybernetik u. AI, Techniscen Universität Wien.

Pfahringer, B. (1997). Compression-based pruning of decision lists. *Proc. of the 14th European Conf. on Machine Learning,* Nashville, Tennessee, pp. 199-212.

Pham, D. T. and Afify, A. A. (2002). Machine learning: Techniques and trends. *Proc. of the 9th Int. Workshop on Systems, Signals and Image Processing (IWSSIP (02),* Manchester Town Hall, UK, World Scientific, pp. 12-36.

Pham, D. T. and Afify, A. A. (2004). Machine learning techniques and their applications in manufacturing. Submitted to *Proc. of the Institution of Mechanical Engineers*, Part B.

Pham, D. T., Afify, A. A. and Dimov, S. S. (2002). Machine learning in manufacturing. *Proc. of the 3rd CIRP Int. Seminar on Intelligent Computation in Manufacturing Engineering (ICME 2002),* Ischia, Italy, pp. III-XII.

Pham, D. T. and Aksoy, M. S. (1993). An algorithm for automatic rule induction. *Artificial Intelligence in Engineering*, Elsevier Science Limited, pp. 227-282.

Pham, D. T. and Aksoy, M. S. (1995a). RULES: A simple rule extraction system. *Expert Systems with Applications*, Vol. 8, No. 1, pp. 59-65.

Pham, D. T. and Aksoy, M. S. (1995b). A new algorithm for inductive learning. *J. of Systems Engineering,* Vol. 5, pp. 115-122.

Pham, D. T. and Dimov, S. S. (1997a). An efficient algorithm for automatic knowledge acquisition. *Pattern Recognition*, Vol. 30, No. 7, pp. 1137-1143.

Pham, D. T. and Dimov, S. S. (1997b). An algorithm for incremental inductive learning. *Proc. of the Institution of Mechanical Engineers*, Vol. 211, Part B, pp. 239-249.

Pham, D. T. and Dimov, S. S. (1998). An approach to concurrent engineering. *Proc. of the Institution of Mechanical Engineers*, Vol. 212, Part B, pp. 13-27.

Pham, D. T., Bigot, S. and Dimov, S. S. (2003). RULES-5: A rule induction algorithm for problems involving continuous attributes. *Proc. of the Institution of Mechanical Engineers*, Vol. 217, Part C, pp. 1273-1286.

Pham, D. T., Bigot, S. and Dimov, S. S. (2004). RULES-5: A rule merging technique for handling noise in inductive learning. Submitted to *Proc. of the Institution of Mechanical Engineers*, Part (C).

Pham, D. T., Dimov, S. S. and Salem, Z. (2000). Technique for selecting examples in inductive learning. *Proc. of the European Symposium on Intelligent Techniques (ESIT-2000)*, Erudit Aachen Germany, pp.119-127.

Pham, D. T. and Karaboga, D. (2000). *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer-Verlag, London.

Pham, D. T. and Liu, X. (1999). *Neural Networks for Identification, Prediction and Control.* Springer-Verlag, London.

Pham, D. T. and Oztemel, E. (1996). *Intelligent Quality Systems*. Springer-Verlag, London.

Piatetsky-Shapiro, G., Brachman, R., Khabaza, T., Kloesgen, T. and Simoudis, E. (1996). An overview of issues in developing industrial data mining and knowledge discovery applications. *Proc. of the 2^{nd} Int. Conf. on Data Mining and Knowledge Discovery*, Menlo Park, CA, AAAI Press, pp. 89-95.

Provost, F. (1992). *Policies for the Selection of Bias in Inductive Machine Learning.* Ph.D. Thesis, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA.

Provost, F. and Buchanan, B. (1996). Inductive policy: The pragmatics of bias selection. *Machine Learning*, Vol. 20, pp. 35-61.

Provost, F. and Hennessy, D. (1994). Distributed machine learning: Scaling up with coarse-grained parallelism. *Proc. of the 2nd Int. Conf. on Intelligent Systems for Molecular Biology*, AAAI Press.

Provost, F. and Hennessy, D. (1996). Scaling up: Distributed machine learning with cooperation. *Proc. of the 13th National Conf. on Artificial Intelligence*, Menlo Park, CA, AAAI Press.

Provost, F., Jensen, D. and Oates, T. (1999). Efficient progressive sampling. *Proc. of the 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, San Diego, CA, USA, pp. 23-32.

Provost, F. and Kolluri, V. (1999). A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, Vol. 2, pp. 1-42.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames. In: *Machine Learning: An Artificial Intelligence Approach* (R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds.)), Vol. I, Tioga Publishing Co., Palo Alto, CA, pp. 463-482.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, Vol. 1, pp. 81-106.

Quinlan, J. R. (1987). Simplifying decision trees. *Int. J. of Man-Machine Studies*, Vol. 27, pp. 221-234.

Quinlan, J. R. (1989). Unknown attribute values in induction. *Proc. of the 6th Int. Workshop on Machine Learning*, Ithaca, New York, Morgan Kaufmann, pp. 164-173.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, Vol. 5, pp. 239-266.

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

Quinlan, J. R. (1994). The minimum description length principle and categorical theories. *Proc. of the 11th Int. Conf. on Machine Learning*, New Brunswick, NJ, pp. 233-241.

Quinlan, J. R. (1995). MDL and categorical theories (continued). *Proc. of the 12th Int. Conf. on Machine Learning*, Tahoe City, California, USA, pp. 464-470.

Quinlan, J. R. (1996a). Bagging, boosting and C4.5. *Proc. of the 13th National Conf. on Artificial Intelligence*, AAAI Press and the MIT Press, pp. 725-730.

Quinlan, J. R. (1996b). Improved use of continuous attributes in C4.5. *J. of Artificial Intelligence Research*, Vol. 4, pp. 77-90.

Quinlan, J. R. and Cameron-Jones, R. M. (1995). Over-searching and layered search in empirical learning. *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence*, Montreal, Quebec, Canada, pp. 1019-1024.

Quinlan, J. R. and Rivest, R. L. (1989). Inferring decision trees using minimum description length principle. *Information and Computer*, Vol. 80, pp. 227-248.

Ramoni, M. and Sebastiani, P. (2001). Robust learning with missing data. *Machine Learning*, Vol. 45, No. 2, pp. 147-170.

Rastogi, R. and Shim, K. (1998). PUBLIC: A decision tree classifier that integrates building and pruning. *Proc. of the 24th Int. Conf. on Very Large Data Bases (VLDB),* New York, USA, pp. 404-415.

Rendell, L. (1989). Comparing systems and analysing functions to improve constructive induction. *Proc. of the 6th Int. Workshop on Machine Learning*, Ithaca, New York, Morgan Kaufmann, pp.461-464.

Richeldi, M. and Rossotto, M. (1995). Class-driven statistical discretisation of continuous attributes. *Proc. of the 12th European Conf. on Machine Learning*, Heraclion, Crete, Greece, Springer-Verlag, pp. 335-338.

Rissanen, J. (1986). Stochastic complexity and modelling. *Annals of Statistics*, Vol. 14, No. 3, pp. 1080-1100.

Rivest, R. (1987). Learning decision lists. *Machine Learning*, Vol. 2, pp. 229-246.

Robnik-Šikonja, M. and Kononenko, I. (1998). Pruning regression trees with MDLP. *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98),* Brighton, UK, Wiley, pp. 455-459.

Rousu, J. (2001). *Efficient Range Partitioning in Classification Learning*. Ph.D. Thesis, Department of Computer Science, University of Helsinki, Finland.

RuleQuest. (2001). Data Mining Tools C5.0. Pty Ltd, 30 Athena Avenue, St Ives NSW 2075, Australia. Available from: http://www.rulequest.com/see5-info.html [Accessed: 1 February 2003].

Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning*, Vol. 10, pp. 153-178.

Schapire, R. (1997). Using output codes to boost multiclass learning problems. *Proc. of the 14th Int. Conf. on Machine Learning*, San Francisco, CA, Morgan Kaufmann, pp. 313-321.

Schapire, R. (1999). Theoretical views of boosting. *Proc. of the 4th European Conf. on Computational Learning Theory*, Nordkirchen, Germany, pp. 1-10.

Schapire, R., Freund, Y., Bartlett, P. and Lee, W. S. (1997). Boosting the margin: A new explanation for the effectiveness of voting methods. *Proc. of the 14th Int. Conf. on Machine Learning*, San Francisco, CA, Morgan Kaufmann, pp. 322-330.

Segal, R. B. (1997). *Machine Learning as Massive Search.* Ph.D. Thesis, Department of Computer Science and Engineering, University of Washington, USA.

Sethi, I. and Savarajudu, G. (1982). Hierarchical classifier design using mutual information. *IEEE Transactions Pattern Analysis and Machine Intelligence*, Vol. 4, pp. 441-445.

Shafer, J., Agrawal, R. and Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining. *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB)*, Mumbai (Bombay), India, pp. 544-555.

Smyth, P. and Goodman, R. (1992). An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 4, pp. 301-316.

Ting, K. M. (1994). Discretization of continuous-valued attributes and instance-based learning. *Technical Report 491*, Basser Department of Computer Science, University of Sydney, Australia.

Ting, K. M. and Witten, I. H. (1997). Stacked generalization: when does it work? *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence*, Morgan Kaufmann, Nagoya, Japan, pp. 250-265.

Tirri, H. (2001). MDL and classification in machine learning. *Proc. of the Neural Information Processing Systems (NIPS) Workshop on Minimum Description Length*, Whistler, British Columbia, Canada.

Towell, G. G. and Shavlik, J. W. (1993). The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, Vol. 13, No. 1, pp. 71-101.

Trautzsch, S. and Perner, P. (1996). A comparison of different multi-interval discretization methods for decision tree learning. *Institute of Computer Vision and Applied Computer Science*, Germany. Available from: http://lareina.imise.uni-leipzig.de/~perner [Accessed: 1 February 2003].

Tsang, E. C. C., Li, H., Veung, D. S. and Lee, J. W. T. (2000). Fuzzy weighted classification rules induction from data. *IEEE Int. Conf. on Systems, Man, and Cybernetics*, Vol. 1, pp. 230-235.

Ventura, D. (1995). *On Discretization as a Preprocessing Step for Supervised Learning Models*. M.Sc. Thesis, Brigham Young University, Provo, UT.

Ventura, D. and Martinez, T. R. (1995). An empirical comparison of discretization methods. *Proc. of the 104th Int. Symposium on Computer and Information Sciences*, pp. 164-178.

Wallace, C. S. and Patrick, J. D. (1993). Coding decision Trees. *Machine Learning*, Vol. 11, pp. 7-22.

Wang, C., Hong, T. and Tseng, S. (1996). Inductive learning from fuzzy examples. *Proc. of the 5th IEEE Int. Conf. on Fuzzy Systems*, Vol. 1, pp. 13-18.

Wang, K. and Liu, B. (1998). Concurrent discretisation of multiple attributes. *Proc. of the 5th Pacific Rim Int. Conf. on Artificial Intelligence*, Singapore, pp. 250-259.

Webb, G. (1993). Systematic search for categorical attribute-value data-driven machine learning. *Proc. of the 6th Australian Joint Conf. on Artificial Intelligence (AI-94)*, Melbourne, World Scientific, pp. 342-347.

Webb, G. (1995). OPUS: An efficient admissible algorithm for unordered search. *J. of Artificial Intelligence Research*, Vol. 3, pp. 431-465.

Weiss, G. M. (1995). Learning with rare cases and small disjuncts. *Proc. of the 12th Int. Conf. on Machine Learning*, Tahoe City, California, USA, pp. 558-565.

Weiss, G. M. and Hirsh, H. (1998). The problem with noise and small disjuncts. *Proc. of the 15th Int. Conf. on Machine Learning*, Madison Wisconsin, USA, Morgan Kaufmann, pp. 574-578.

Weiss, G. M. and Hirsh, H. (2000). A quantitative study of small disjuncts. *Proc. of the 17th National Conf. on Artificial Intelligence*, Austin, Texas, pp. 665-670.

Weiss, S. and Indurkhya, N. (1991). Reduced complexity rule induction. *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence*, Sydney, Australia, Morgan Kaufmann, pp. 678-684.

Witten, I. H. and Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, USA.

Wneck, J. and Michalski, R. S. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, Vol. 14, No. 2, pp. 139-168.

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

Wong, A. K. C. and Chiu, D. K. Y. (1987). Synthesizing statistical knowledge from incomplete mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 9, No. 6, pp. 796-805.

Wray, B. A., Rakes, T. R. and Rees, L. (1997). Neural network identification of critical factors in dynamic just-in-time kanban environment. *J. of Intelligent Manufacturing*, Vol. 8, pp. 83-96.

Wu, X. (1996). A Bayesian discretizer for real-valued attributes, *Computer Journal*, Vol. 39, No. 8, pp. 688-694.

Wu, X., Chu, C. H., Wang, Y. and Yan, W. (2002). A genetic algorithm for integrated cell formation and layout decisions. *Proc. of the Congress on Evolutionary Computation (CEC-02)*. Vol. 2, pp. 1866-1871.

Wu, X. and Lo, W. H. (1998). Multi-layer incremental induction. *Proc. of the 5th Pacific Rim Int. Conf. on Artificial Intelligence*, Singapore, pp. 24-32.

Yang, J. and Honavar, V. (1998). Feature selection using a genetic algorithm. *IEEE Intelligent Systems*, Vol. 13, No. 2, pp. 44-49.

Ye, N. and Li, X. (2002). A scalable incremental learning algorithm for classification problems. *Computers & Industrial Engineering*, Vol. 43, pp. 677-692.

Zaki, M. (1998). *Scalable Data Mining for Rules*. Ph.D. Thesis, Department of Computer Science, University of Rochester, Rochester, NY.

Zaki, M. J., Ho, C. and Agrawal, R. (1999). Scalable parallel classification for data mining on shared memory multiprocessors. *Proc. of the 15th IEEE Int. Conf. on Data Engineering*, Sydney, Australia, pp.198-205.

Zhang, X., Mesirov, J. and Waltz, D. (1992). A hybrid system for protein secondary structure prediction. *J. of Molecular Biology*, Vol. 225, pp. 1049-1063.

Zhang, T., Ramakrishnan, R. and Livny, M. (1997). BIRCH: A new data clustering algorithm and its applications. *Int. J. of Data Mining and Knowledge Discovery*, Vol. 1, No. 2, pp. 141-182.

Zhang, S. and Wu, X (2001). Large scale data mining based on data partitioning. *Applied Artificial Intelligence*, Vol. 15, pp. 129-139.

Zhou, Z. H., Jiang, Y. and Chen, S. F. (2000). A general neural framework for classification rule mining. *Int. J. of Computers, Systems, and Signals*, Vol. 1, No. 2, pp. 154-168.

Zighed, D. A., Rakotomalala, R. and Feschet, F. (1997). Optimal multiple intervals discretization of continuous attributes for supervised learning. *Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD-97)*, Newport beach, California, USA, AAAI Press, pp. 295-298.