SCALABILITY OF MACHINE LEARNING ALGORITHMS

A thesis submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Science

November 1993

By
Georgios Paliouras
Department of Computer Science

Contents

List of Tables

List of Figures

Abstract

During the last two decades, there has been a significant research activity in Machine Learning, which has mainly concentrated on the task of *empirical concept learning.* This method of learning involves the acquisition of knowledge from a set of examples, the training set, using generalisation techniques.

The task of empirical concept learning can be thought of as being equivalent to the classification task, previously performed by statistical techniques. Despite the existence of a large number of problems which can be considered classification tasks, ML techniques have not been widely applied to real-world problems. One of the possible reasons for this is that learning programs cannot handle large-scale data, used in real applications.

Considering that possibility, the presented thesis examined the *scalability* of five concept-learning algorithms, defining scalabilty by the effect that an increase in the size of the training set has on the computational performance of the algorithm. The programs that were considered are: NewID [Niblett, 1989], C4.5 [Quinlan, 1993], PLS1 [Rendell, 1983a], CN2 [Clark and Niblett, 1989] and AQ15 [Michalski *et al.*, 1986].

The first part of the project involved the theoretical analysis of the algorithms, concentrating on their worst-case computational complexity. The obtained results deviate substantially from those previously presented (e.g. [O'Rorke, 1982] and [Rendell *et al.*, 1989]), providing over-quadratic worst-case estimates.

The second part of the work is an experimental examination, using real and artificial data sets. Two large real data sets have been selected for that purpose, one dealing with *letter recognition* and the other with *chromosome classification.* The experiments that were done, using those two sets, provide an indication of the average-case performance of the programs, which is significantly different from the worst-case one. The artificial data set, on the other hand, provides a near-worst case situation, which confirms the obtained theoretical results.

The results of the theoretical and experimental analyses show that, although their worst-case computational complexity is over-quadratic, most of the examined algorithms can handle large amounts of data. Those which had difficulties

did not do so because of their order of complexity, but because of their standard computational "*unit-cost*", which affects significantly their performance. The size of the training set is only one of the parameters affecting scalability. The examination of other factors (e.g. the complexity of the learning task) is equally interesting.

DECLARATION

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

The Author

Georgios Paliouras graduated from the University of Central Lancashire, obtaining a first class BSc(Hons.) degree in Computing with Economics. His work experience includes a one-year long Industrial Placement in Siemens AG., Karlsruhe, Germany, working on data compression and debugging tools. Having done his final year undergraduate project on Machine Learning algorithms, he joined the Department of Computer Science at the University of Manchester, in October 1992 to work as a research student in the field of Machine Learning. For this work he has been awarded a studentship from the Science and Engineering Research Council.

To my father $A\chi\iota\lambda\lambda\acute{\epsilon}\alpha$ $\Pi\alpha\lambda\iota o\acute{\upsilon}\rho\alpha$.

Acknowledgements

I am greatful to the following people for supplying me with or helping me acquire programs, documentation, data sets and other valuable information:

> G. Blix, E. Bloedorn, R. Boswell, P. Errington, J. Graham, D. Haussler, Y. Makris, R. Michalski, C. Moore, S. Muggleton, R. Nakhaeizadeh, T. Niblett, P. O'Rorke, T. Parsons, R. Quinlan, L. Rendell, M. Rissakis, D. Slate, D. Sleeman, S. Wrobel, X. Wu

This work was greatly facilitated by the exchange of materials available within the *Concerted Action of Automated Cytogenics Groups*, supported by the European Community, Project No. II.1.1/13, and the use of materials from the *UCI Repository of machine learning databases* in Irvine, CA: University of California, Department of Information and Computer Science.

I would also like to thank C. Casey, R. Lee, S. Nicklin, D. Rydeheard, R. Sakellariou and G. Theodoropoulos for suggesting corrections and proof-reading the thesis.

This work was partly funded by a studentship received from the Science and Engineering Research Council, the support of which is kindly acknowledged.

Most of all I would like to thank my supervisor Prof. D.S. Brée, my family and $\Delta\acute{\eta}\mu\eta\tau\rho\alpha$ for supporting me throughout my work.

Chapter 1

Introduction

During the last two decades, there has been a significant amount of research activity in ML, which has mainly concentrated on the task of *empirical concept-learning*. This method of learning involves the acquisition of knowledge from a set of examples, the *training set*, using several generalisation techniques. The presented thesis examines the scalability of five concept-learning algorithms, where scalability is defined by the effect that an increase in the size of the training set has on the computational performance of the algorithm. This chapter introduces briefly some important aspects of Machine Learning (ML) and outlines the aims of the project.

1.1 Definition of Learning

One of the sources of difficulty when trying to set the objectives of Machine Learning (ML) is the definition of learning. The concept of learning is rather abstract and those who have tried to define it (philosophers, psychologists, AI workers, etc.) have usually only managed to uncover one of the many "faces" of the complicated process.

However, there are some aspects of learning which have been agreed upon by most of the people who have dealt with the problem and these provide, for many purposes, a good description of the process. Some of those aspects are the following:

- There is always a system that is able to improve itself, manipulating information, provided by its environment.
- The information provided to the system can usually take more than one form and the system has more than one way of changing its current state

(i.e. there is more than one type of learning).

- The system is usually able to remember and recall things that it has experienced.

This general description of learning however does not contain much information about the way in which the process is achieved and the elements into which it can possibly be decomposed. It is on those aspects of the learning process that the opinions of different researchers are diverge.

1.2 The objectives of ML

Bearing in mind the diversity of opinions as to what learning is and how it can be achieved, one can understand the difficulty in defining the purpose of ML and setting some clear-cut objectives for it. Thus, although the main idea is well-defined (i.e. man-made systems that are able to learn), different groups of people have approached ML differently. In doing that, they have each set their own expectations about the outcome of ML.

Thus one can distinguish between the following views of ML:

1. **The Philosophical view.**
   The main concerns of philosophers about ML are:

   whether artificial learning systems can be produced,

   what is the purpose of learning in human beings

   and what would be the consequences of developing learning machines.

2. **The psychological view.**
   Psychologists (Cognitive Scientists) are interested in the mental processes involved in human learning. They would like to be able to model or mimic them, using machines, in order to enhance their understanding of learning.

3. **The neurophysiological view.**
   Learning is one of the most complicated functions of the brain and physiologists who are dealing with it are very interested in implementing their ideas, using machines to produce brain-models and observe their behaviour. A successful product of this research is Neural Networks (NNs), which are brain-model based learning systems.

4. **The Artificial Intelligence (AI) view.**
   AI workers are interested in developing artificial learning systems, since learning is one of the most important intelligent processes. The way in

which they are trying to achieve their target is not limited to models of human learning processes, although these provide good guidance for their research.

5. **The engineering view.**
   From an engineering (and/or business) point of view, learning machines might prove to be the solution for some of the problems of current information systems. One might expect that adaptive and self-improving systems would increase the efficiency and decrease the effort that has to be made by humans, in several tasks (e.g. prediction, diagnosis, etc.).

These views overlap in many ways. For example, work in AI incorporates philosophical, cognitive and physiological ideas and at the same time its products are sometimes business/engineering-oriented. Another example is the use of NNs for practical applications.

1.3 Approaches taken so far

Due to the variety of objectives set for ML, a number of different approaches to learning systems have emerged. These approaches could be classified in the following three types:

1. **Brain-modelling.**
   This was one of the first approaches in ML. It was based on the theory of cybernetics and neurobiological brain-models, producing highly-connected "neural structures", that interacted with each other in a near-random fashion, similar to the way that the brain was thought to work.

   Originally this approach was not successful, but recently similar approaches have regained popularity, producing systems (NNs) that are much less ambitious than their predecessors (i.e. they are set to solve specific tasks, rather than achieving general-purpose learning) and which have had some positive results.

2. **Learning algorithms.**
   The bulk of the work in what is called *symbolic ML*, which is the classical AI approach to learning, was done on individual algorithms, using many different methods, in order to achieve learning. Most of those algorithms fall under one of the following categories:

   - Learning by deduction.
     This type of learning algorithm assumes a large amount of background knowledge about the problem, which it analyses, deducing rules and models that can be used later to solve specific problems.

- Learning by analogy.
  Algorithms falling into this category also assume some general background knowledge, which is usually provided in the form of example situations, followed by corresponding explanations. The system structures this information in such a way as to be able to use it to explain new experiences. In other words there is a mapping of new information onto what is already available in the system, causing a continuous extension of the system's knowledge.
- Learning by induction.
  This is the type of learning that has been paid the most attention. Algorithms falling under this category learn by generalising on specific examples (the training set), using a number of different generalisation techniques. No initial knowledge is assumed as the system extracts information from the training set in a near-statistical way. *Empirical concept learning*, which will be further discussed in chapter 2, belongs to this category.

3. Application-oriented learning systems.
   Some of the practical problems that have been attacked by ML are the following:

   - **Knowledge Acquisition (KA) for Expert Systems (ESs).**
     KA is one of the most difficult tasks in building an ES, due to the fact that experts have large amounts of knowledge which they find difficult to transfer. On the other hand, in many cases large pools of past data are available, which can be used for the extraction of information about the problem. One way to acquire information in this case is to use inductive learning systems.
   - **Adaptive and Self-improving Systems.**
     There is a number of situations where a system is required to adapt its knowledge according to new data that become available. This means either an improvement in the system's performance (e.g. self-improving ESs) or adaptability of the system to changing circumstances (e.g. adaptive control systems). The initial knowledge, in this case is either directly provided to the system or induced from examples by the learning program itself.
   - **Forecasting Systems.**
     Forecasting systems that have been produced are mainly at an experimental stage; they make use of learning systems that induce forecasting rules based on past data. Examples of situations where such systems could be used are: weather, economic and business forecasting.
   - **Pattern Recognition Systems.**
     Pattern Recognition systems are also at an experimental stage and are

mainly used in Machine Vision systems, where patterns and objects need to be recognised in different situations. NNs are the main method used for this purpose, replacing traditional statistical classification approaches.

Experience with ML applications has shown that in many cases a single learning algorithm cannot provide an adequate solution to the problem. As a result, some of the recent research in ML has been oriented towards systems that combine more than one learning method. This is what is called *multi-strategy learning* and has become very popular lately.

1.4 Motivation for the project

The task of empirical concept learning can be thought of as being equivalent to the classification task (see section 2.3), previously performed by statistical techniques. Since there is a large number of problems which can be considered to be classification tasks, one would expect ML techniques to be widely applied to real-world problems. However this is not the case. There are only a few real-world applications of ML and most of them are not large-scale ones. There are a number of possible reasons why this happens:

1. ML techniques do not provide adequate solutions to real-world problems.
2. Statistical classifiers achieve a better performance than the classifiers generated by ML programs.
3. ML algorithms make assumptions about the structure of the problem and the provided data that do not hold in real problems.
4. ML algorithms cannot be applied to large-scale data.
5. The existing ML programs have not been designed to handle large-scale data.

In the first few years of ML research, most of the above claims were true. However, subsequent research has led to the improvement of learning systems, overcoming most of these problems (see section 2.5). One problem to which little attention has been paid is the behaviour of learning algorithms on large-scale data. There have been analyses and comparisons of ML algorithms in the past (e.g. [O'Rorke, 1982], [Gams and Lavrac, 1987], [Rendell *et al.*, 1989]), but none has looked in detail at the scalability of the algorithms.

The aim of the project was to address this neglected issue, examining the truth of the last two of the above list of claims about ML algorithms and programs.

For this purpose five ML programs were selected, which perform similar types of learning (i.e. empirical concept-learning) and their behaviour was analysed both theoretically and experimentally. The theoretical analysis involved a thorough analysis of the computational complexity of the programs, providing a worst-case estimate of their performance on different scales of data. The experimental investigation examined the behaviour of the programs when applied to data sets of varying scale. Three data sets were used for this purpose, two real and one artificial. By combining the results of the theoretical and the experimental analyses a complete picture of the scalability of the algorithms was formed.

1.5 The Structure of the Thesis

Following this brief introduction to ML and the objectives of the project, **chapter 2** takes a closer look at *inductive learning* concentrating on *empirical concept-learning*. It first presents the supporting theory for this type of learning, linking it to the problem of classification, then gives a brief account of the different approaches to classification, ranging from statistical methods to genetic algorithms. Following this account, two symbolic learning algorithms are examined, which have been the centre of most of the research activity in the field. Finally, a brief review of the work in Computational Learning Theory, a rapidly growing research area in empirical concept-learning, is given.

Chapter 3 presents the theoretical analysis of the five algorithms. For each of the algorithms the following information is provided:

1. A *description of the algorithm*, focusing on its peculiarities.
2. The *design of the algorithm.*
3. A detailed *worst-case computational complexity analysis.*

Chapter 4 describes the experiments and presents their results. The results are also statistically analysed, to allow the comparison of the algorithms' relative performance and the validation of the theoretical estimates, presented in the previous chapter.

Finally, **chapter 5** summarises the results presented in the thesis and draws conclusions about their importance in the context of the scalability problem.

Chapter 2

Theory of Inductive Learning

2.1 Introduction

One of the main research areas in ML and the one that this thesis concentrates on is *inductive learning*. Inductive learning involves the use of inductive inference for the acquisition of knowledge from experience. It has attracted most of the research done in ML (see [Winston, 1975], [Quinlan, 1986a], [Mitchell, 1977]), resulting in the development of many interesting techniques. Some of those techniques will be described later in this chapter.

The task that is usually set in inductive learning is the acquisition of concepts from examples (*empirical concept-learning*). In empirical concept-learning, the system is provided with a set of positive and negative examples of a concept, as described by a set of features, which can take a range of values. For example, one may describe the concept of a *bird* by the *number of wings*, the *number of legs*, the *size*, the *flying ability*, etc. In this case, some of the positive examples of the concept will be the following :

no. of wings	**no. of legs**	**size**	**fly**
2	2	small	yes
2	2	big	yes
2	2	big	no

while some negative ones could be :

no. of wings	**no. of legs**	**size**	**fly**
4	6	small	yes
2	0	big	yes
0	2	small	no

The outcome of concept-learning is a *concept-description/concept-function*, which is able to discriminate between objects that are instances of the concept and those which are not, based on their feature-values. This task is also called *object-classification* and it has been an active area of research in statistics. Its fields of application include diagnosis (e.g. medical), forecasting, decision-making, etc.

Real-world applications of learning systems are more demanding than the toy-problem described above. As a result, a number of extensions have been made to the basic model, in order to make it more widely applicable. The most common extension is the use of *multiple concepts* (or non-binary classes). In this case more than one concepts need to be distinguished from the same data, while in some cases a continuously-valued class is used (i.e. an "infinite" number of concepts). Examples of such learning problems is the diagnosis of different types of diseases and the forecasting of the closing price of a currency[1].

Another major extension to the basic concept-learning model is '*unsupervised learning*'. In this case the objects that are provided for training are not preclassified and the learning system is required to *cluster* them into groups which share common features. This is a more difficult type of learning (also called *discovery*), because the selection of important classification features is not aided by the pre-classification of the training instances. An application area where such problems are common is '*object identification*' in *Machine Vision*.

This thesis examines *supervised learning* techniques in *multi-concept learning*.

2.2 Induction as a Search

Provided a set E of positive and negative examples of a concept c, an inductive learning system is required to form a hypothesis H (based on E), that will contain the main features of the concept and will correctly distinguish between instances and non-instances of it. This process can be thought of as a search through a state space, where the states are all the possible hypotheses that correspond to the given attribute (feature) set and the goal is the hypothesis that best describes the concept. The operators that lead the search through this space are the inference rules incorporated in the learning algorithm.

[1]Notice that the term classification suits better to those types of learning tasks, since there is really one class that can take more than one values, rather than many different concepts.

2.2.1 The Goal: Hypothesis

The learning system is asked to induce a hypothesis that will be *complete* and *consistent* with the example set. This means that it has to cover all the positive and exclude all the negative examples. Using Michalski's [Michalski, 1983] formulation, the following conditions have to hold:

$$\forall i \in I, (E_i \Rightarrow D_i) \quad (\textit{the completeness condition})$$

$$\forall i, j \in I, (D_i \Rightarrow \neg E_j) \text{ if } j \neq i \quad (\textit{the consistency condition})$$

where I corresponds to the number of class-values, D_i is the induced hypothesis for i^{th} class and E_i is a description satisfied only by the positive events of the class.

In some cases there will be clashes between instances in the training set, caused usually by noisy data. In that case either the completeness or the consistency condition is relaxed, in order to resolve the contradiction. For most non-noisy training sets however more than one hypotheses is expected to satisfy the two conditions. The choice of the final hypothesis depends on the learning algorithm and the search operators that it uses. In this respect there are two main types of learning algorithms, which correspond to the two extreme cases :

- *characteristic* learning algorithms
- *determinant* learning algorithms

Algorithms falling under the former category search for a typical description of the concept that will contain as much information as possible. This is called '*maximal characteristic descriptor*'. These of the latter type aim at a hypothesis that will correctly discriminate between positive and negative instances and will be of minimal information content. This is called '*minimal discriminant descriptor*'. Between these two extremes there are a number of concept learning algorithms, which also make use of other criteria in deciding for the final hypothesis. Such criteria might be the simplicity of the hypotheses or the preference of certain attributes against others.

2.2.2 The Search Space: Hypothesis Space

The nature (i.e. size and complexity) of the search space is determined by the following two factors:

- **The Attribute Set.**
- **The Description Language.**

The Attribute Set

Firstly, the *type* and the *domain* of the attributes that describe the concept affect the size of the search space. One usually distinguishes between three types of attributes:

- *Nominal attributes*
  These are the ones that take nominal values.
  (e.g. *size* = *big* or *normal* or *small*).

- *Numeric attributes*
  These take numeric values, which will usually be either integer or real
  (e.g. *length* = 5).

- *Tree-structured attributes*
  These are the ones that can be organised hierarchically.
  (e.g. figure 2.1)

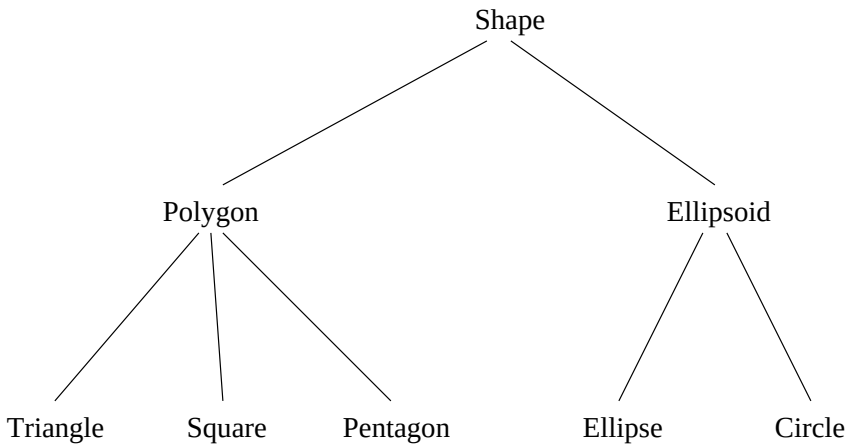

Figure 2.1: An example of a tree-structured attribute. In this case *shape* can take the values: *polygon*, *triangle*, *square*, etc.

On the other hand, the *domain* of the attribute gives some information about its "meaning" and can be used to restrict the search space. For example a range can be specified for a numeric attribute and/or the interval between the values that it can take.

Moreover, the *relevance* of the attributes to the problem affects the complexity of the learning task. The existence of many irrelevant attributes is a type of noise, which will cause significant deterioration to the performance of most inductive learning algorithms. In most cases however the learning algorithm is able to discard irrelevant attributes by examining the training data. This type of induction is known as *selective concept learning* (examples can be found in [Michalski, 1983] and [Quinlan, 1986a]) and is the one most commonly met in the ML literature. A different type of learning, which requires a more complicated inductive procedure, is *constructive concept learning* (see [Rendell, 1985], [Hong *et al.*, 1986]). In this type of learning, the attributes are assumed to contain less encoded knowledge and better performance can be achieved by combining them in several ways. A simple example of this would be the concept '*right-angled triangle*', for which the length of its sides is given. Although one cannot classify a shape as a '*right-angled triangle*', by considering the length of each side individually, one can calculate the squares of the given lengths and compare the sum of the two shorter ones with the third to decide whether the triangle is right-angled.

The Description Language

The language that is used for building hypotheses, by combining attributes, also affects the size and the complexity of the space. The more expressive the language is, the larger the number of hypotheses that can be induced and the larger the search space. There are various different representation schemes that have been examined in concept-learning and classification research, resulting from different approaches to the problem (e.g. Statistical Classification, Neural Networks, Symbolic Learning, etc.; see next section). The field of *Computational Learning Theory* examines the complexity of different types of concepts and the extend to which each of these is learnable (section 2.6).

2.2.3 The operators

Inductive learning is mainly based on generalisation. In other words, the hypothesis that will be induced is a generalisation of the positive examples of a concept (which must, at the same time, be specific enough as to exclude the negative examples). The inductive process starts with an initial hypothesis, which is being modified by specialisation and generalisation operators in order to achieve a better fit to the training data. This search for a good fit is guided by one or

more heuristic functions which are incorporated in the learning algorithm. The construction of the initial hypothesis, the search operators and the heuristics that are used differ between approaches. Some of those will be examined in sections 2.3 and 2.4.

2.3 Approaches

The problem of concept-learning/classification has been approached from different perspectives. This section gives a brief overview of the most common of those approaches:

- Statistical Classification.
- Similarity Based Learning.
- Neural Networks.
- Genetic Algorithms.

For more detailed description of the methods that are discussed here, the reader is referred to [Weiss and Kulikowski, 1991] and [Nakhaeizadeh *et al.*, 1993].

2.3.1 Statistical Classification

This field of research in statistics is the predecessor of concept-learning and has contributed a number of interesting classification methods. Some of the common features of these methods are the following:

- Only *numeric attributes* are used.
- Each instance of a class must have a value associated with each of the attributes (i.e. no '*missing values*' are allowed).
- The methods are grouped into *parametric* and *nonparametric*. The former assume a specific type of discrimination function, which they try to fit to the data, by adjusting its parameters. The latter do not make this assumption.

The following are some of the methods that are commonly used for statistical classification:

1. **Linear Discriminant.**
   This method attempts to generate hyperplanes which will achieve good discrimination between the classes. The number of hyperplanes that are used depends on the number of classes, each separating one class from the rest. The calculation of the parameters, specifying the location of each hyperplane, is usually done by regression analysis. Figure 2.2 illustrates the way in which a hyperplane (straight line in this case) is used to discriminate between two classes, in a problem with two attributes. This is an ideal case, where the classes can be discriminated perfectly, using a straight line.

2. **Logistic Discriminant.**
   This is an improved version of the linear discriminant method, which uses a different criterion for regression (i.e. maximising *conditional likelihood*, instead of optimising a quadratic cost function).

3. **Quadratic Discriminant.**
   Real classification problems are not always solvable by linear discriminants. Figure 2.3 presents a situation which falls into that category. Because of that, another parametric classification method was developed which generates quadratic curves for discriminating between classes.

4. **K-Nearest Neighbour**
   This is a simple non-parametric classification method, which makes use of the preclassified instances, in order to classify new ones. It examines the K closest neighbours of the new instance and classifies it according to the most common class amongst them. The most important feature of this method is the formula which is used for calculating the distance between instances. Some alternatives that have been used are the following:

   - **Absolute distance:** The sum of the absolute differences between the attribute values of the two instances.
   - **Euclidean distance.**
   - **Normalised distances.** For example, the number of standard deviations from the mean of each feature.

   Figure 2.4 illustrates the use of this method.

5. **Conceptual Clustering.**
   This is a method which is very close to the symbolic ML approach to the problem. According to that, instances which share common features and common class are grouped together, forming clusters, which can be used for classifying new instances. Clustering can also be used for unsupervised learning, in which case instances are grouped according to their feature-values and a class-label is attached to the generated clusters. One of the algorithms examined in this thesis (PLS1) is a conceptual clusterer; it will be described in detail in the following chapter.

Figure 2.3: A situation where linear discrimination cannot provide an adequate solution.

Figure 2.4: Classification using the K-Nearest Neighbour method. In this case $k = 4$ and the new instance is assigned to the negative class, because 3 out of the 4 examined neighbours belong to that class.

2.3.2 Similarity Based Learning (SBL)

This is a symbolic approach to inductive learning, which is also the approach taken in this thesis. Most of the methods developed under this paradigm produce classifiers which can be interpreted by a set of clusters, similar to the conceptual clustering approach. There are however a number of differences between the SBL and conceptual clustering methods:

- SBL methods can handle nominal and structured attributes. Early versions of SBL algorithms could not handle numeric attributes at all, while more recent ones usually discretise them and treat them in a similar way to nominal ones. This approach imposes a substantial overhead on the computational requirements of the algorithms (see chapter 3) and does not make effective use of the attributes (see [de Merckt, 1993]).
- The clusters that are generated in SBL are usually *orthogonal hyperrectangles* formed by the introduction of dichotomising hyperplanes, which are parallel to the feature axes (e.g. figure 2.5). The conceptual clusterer that is examined in this project is of the same type, but others use different shapes

Figure 2.5: Orthogonal clustering in a binary-class problem with two attributes (one nominal and the other numeric).

> This restriction in the shapes of the clusters, imposed by SBL methods, has recently been realised as an important problem and some work has been done in order to overcome it (e.g. [Murthy *et al.*, 1993]). The restriction is imposed by the description languages that are used, which limit the ways in which attributes can be combined to simple conjunctions and disjunctions between attribute-tests[2]. This makes sense when nominal attributes are used, since their values cannot always be ordered. The situation however is different with numeric attributes, between which there may be a relationship, defined by a linear or other numerical function.

A number of different conventions have been used in SBL research for describing the induced concepts, the most popular of which are *decision trees* and *decision lists*. These representation schemes, together with the heuristics that are used in some SBL algorithms are described in section 2.4.

[2]An attribute-test is the association of a value or a range of values to an attribute.

Figure 2.6: Elliptic clustering.

2.3.3 Neural Networks

Neural Networks achieve a similar classification result to the statistical classifiers, using a different representation scheme. The scheme contains a number of "*neurons*", arranged into several layers. Each of the neurons in one layer is usually connected to all the neurons in the previous layer, from which it receives input, and all the neurons in the next layer, to which it feeds its output (Figure 2.7). The first layer of neurons is used for input to the system. Neural Networks accept only numeric input and for this reason nominal attributes are translated into the set of all possible attribute-tests, each of which is binary valued. Following this pre-processing stage, each input node accepts values for one attribute or attribute-test. These values are fed to the next layer of nodes, which perform a weighted summation and generate an output value, according to some function. The output values are passed to the next layer of neurons (if it exists), which process them in the same way and continue the *forward-feeding* process until the *output layer* is reached. At that stage, the generated output is compared to the desired output[3], provided by the preclassified instance and their difference is fed back to the previous layers, causing the adjustment of the connection weights and other parameters that are used in the calculations taking place in each node.

[3]This is not the case for unsupervised learning.

Figure 2.7: A two-layered fully-connected network. a_i stands for the input neuron which corresponds to the i^{th} attribute/attribute-test and c_i for the output neuron which corresponds to the i^{th} class.

The calculations involved in the forward-feeding and the weight-adjustment stage differ between different types of networks. The following is a very brief account of some commonly used networks:

1. **Perceptron.**
   This is a very simple type of network, invented by Rosenblatt [Rosenblatt, 1962]. It consists of only two layers of neurons (an input and an output one) and its behaviour is identical to the linear discriminant, with the difference that it is a non-parametric method (i.e. it does not make any assumptions about the shape of the class probability distributions). The generated value at each neuron in the output layer is given by the following formula:

$$v_j = \begin{cases} 1 & \text{if } \sum_i w_{ji} I_i - \theta_j \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

   where I_i is the i^{th} input value, w_{ji} is the weight associated to the connection between the i^{th} input and the j^{th} output neuron, and θ_j is a threshold value associated with the j^{th} output neuron. There is a different threshold value for each output node, which gets updated at the weight-adjustment stage.

Another idea that is similar between the perceptron algorithm and the linear discriminant is the use of the square difference for calculating the distance between the generated and the expected output value. This calculated value is used for updating the weights and the thresholds. With respect to the weight-adjustment process, there are mainly two approaches:

- **Batch Learning:** All the examples in the set are examined before any adjustment takes place. In that case the *mean of square errors* is calculated and used in the adjustment.
- **Incremental Learning:** The adjustment takes place after each example has been considered. The examples are either processed sequentially or randomly. In this method the absolute difference is used in the adjustment.

The former method is expected to give more reliable results, since an overview of the whole training set is maintained, but it is computationally more expensive than the latter.

2. **Multi-layer Perceptron (MLP).**
As mentioned in the statistical approaches, there is a number of problems which cannot be solved with linear discrimination. In order to overcome this problem in neural networks, several perceptrons are combined together. The result is a network with a number of *hidden layers* between the input and the output ones, which can approximate non-linear functions.

In parallel to the introduction of more than one layer, the calculation of the feed-forward values at each layer and the weight-adjustment method have been improved. The output of each node is now calculated by the *logistic* or *sigmoidal* function:

$$v_j = \frac{1}{1 + e^{-n_j}} \tag{2.2}$$

where e is the base of the natural logarithm and n_j is the weighted sum (including the threshold) involved in equation 2.1 also.
The output of this function is within the range $(0, 1)$ and can be used in the calculation of the difference between actual and expected output value, without being translated in a binary form as in equation 2.1. The result of this is a smoother adjustment of the weights and the other parameters.

The adjustment of the weights is now done with the use of the *Back Propagation* algorithm [Rumelhart *et al.*, 1986], which is similar to the one used in the simple perceptron, but incorporates more parameters. The aim is still to minimise the sum of least square errors for the training set, but the errors are now propagated more than one layers back, in order to adjust all the weights and the thresholds in the network. The new update function is given by the following equation:

$$w'_j = w_j - \eta e_j + \alpha \delta w_{t-1} \tag{2.3}$$

where η is the *learning rate* or *step-size*, which controls the speed of learning, e_j is the proportion of error propagated to node j, δw_{t-1} is the last change of the weight and α the *momentum* parameter, which controls the effect of the previous weight-change.
The Back Propagation of errors and subsequent adjustment of the weights continues until no substantial difference between the amount by which weights change at step t and step $t-1$ exists. This iterative error minimisation method is known as *gradient descent*. If the step-size (η) is too large this method will lead to an oscillation around the desired minimum error-value. This problem is solved by the inclusion of the most recent weight-change in the calculation.

3. **Radial Basis Function Networks (RBFN).**
This is a new type of network, the main characteristic of which is that it does not use an iterative learning method, as in Back Propagation. The RBFN is provided with a number of points in the feature space, each of which is used as the centre of an interpolating function. The set of all those functions can be used as a classifier. A major problem in RBFNs is the determination of the function centres. A number of methods have been developed for solving that problem, which vary from arbitrary and random approaches to unsupervised learning ones. For a detailed description of RBFNs the reader is referred to [Nakhaeizadeh *et al.*, 1993].

4. **Kohonen Networks.**
Kohonen's network [Kohonen, 1984] provides an unsupervised learning method, using Neural Networks. Usually in unsupervised learning, the network is provided with the number of desired clusters and associates a single cluster with each output neuron, by adjusting only this neuron's weights when it achieves the highest output value (*winner-takes-all* network). Kohonen's network however updates also the weights of the output neurons which are architecturally close to the "*winner*", achieving an interpolation effect, which arranges the clusters according to their arrangement in the feature space. This effect is similar to that of a traditional statistical algorithm, called the k-means clustering algorithm, which generates a partition of the feature space into "*patches*", called the *Voronoi tesselation*.

2.3.4 Genetic Algorithms (GAs)

GAs are search methods, which are inspired by Darwin's evolution theory about the survival of the fittest. Figure 2.8 summarises the main elements of a genetic algorithm.

Special interest has been shown in the following five elements of a GA :

Figure 2.8: The process performed by GAs.

Representation scheme. The main representation scheme, that has been used, (proposed by Holland, who is one of the main contributors in the field) is bit-strings, which are called 'chromosomes'. Each chromosome represents a rule or an example, which is encoded in a bit-form. Each attribute of an example (or condition of a rule) is assigned a bit (or a group of bits) in the chromosome, which will hold the values that the attribute (condition) takes in specific examples (rules).

Initialisation. The first stage of the evolutionary process involves the generation (or acquisition) of an initial set of rules. For research purposes, random generation of those rules from the algorithm itself (e.g. by random assignment of values in the bit strings) is favoured. The reason is that it is a good test for the algorithm to start its evolution from a random population that may have nothing to do with the desired optimal one. In this case, if the algorithm manages to find its way to the optimal rule, it is considered to have performed well in searching through the space.

In real-world applications however, where safety and processing time are important, the initial population is usually supplied to the algorithm by the user. In this case, the initial population may be derived from the user's personal experience, or from a different learning algorithm.

Evaluation function. The evaluation function is one of the most important elements of a GA. Small changes to it can improve or worsen the algorithm substantially. The objective of the evaluation function is to assign a worth to each rule, according to its success in classifying examples from the training set. In order to achieve that, there is a number of factors that can be taken into account. For example:

- The classification correctness of the rule.
- The complexity of the rule.
- The performance of the rule "in the past".

Genetic Operators There are three main reproduction operators (proposed by Holland) that are used in GAs:

1. **Crossover**
   This operator is inspired by the sexual reproduction of species in the real world. It involves the recombination of the "genetic" information of two rules, in order to produce new rules. The newly generated rules do not contain any new "genetic" material, but they could be more (or less) successful than their parents, because they contain different combinations of their "genetic information". The following example illustrates the way in which the operator works for binary strings:

Rule_1 : 1001101
Rule_2 : 1110110
Applying crossover at the 4^{th} bit:
Rule_3 : 100-0110
Rule_4 : 111-1101

2. **Mutation**
   This operator is used for the introduction of "new ideas" to the search, avoiding thus the concentration of the search at local maxima. This is done by moving (in a random way) the search to different areas of the search space. The way it operates is by altering the genetic information of a rule at some random position, with a randomly generated value.

3. **Inversion**
   This is a complementary operator to crossover but is applied rarely. What it does is to rearrange the genetic information within a chromosome, by reversing a substring so that pieces of genetic information, that were far from each other are brought together and can be used in a substring selected by crossover. For example:

   Rule_1 : 1001101
   Reversing substring between the 4^{th} and the 6^{th} bit:
   Rule_2 : 100-011-1

Parameters. There is a number of parameters used in various components of a GA (e.g. the size of the population, the probabilities of selecting each of the operators, the condition that will have to be satisfied, in order for the search to stop, etc.). Most of these parameters are highly dependent on the application and are likely to affect substantially the performance of the algorithm.

2.4 Two Popular SBL Algorithms

This section describes two algorithms, which have been the basis of most SBL learning systems. Most of the systems that are used in the project are also descendants of these algorithms.

2.4.1 The AQ Algorithm

AQ was one of the first successful algorithms in symbolic ML. It is based on the STAR method [Michalski, 1983], which was developed by Michalski in the late '70s. One of the interesting features of AQ is the comprehensibility of its

concept-description language, *Annotated Predicate Calculus (APC)*[4]. In terms of the APC formalism, examples and hypotheses are expressed, using the following building blocks:

Selectors. These are the elementary blocks of the language. They correspond to attribute-tests, where an attribute is associated to one or a range of its values, by means of some "*relational predicates*" (e.g. $=, \leq, >$, etc.). Examples of selectors are the following:

$(shape\text{=sphere})$
$(weight > 5)$
$(colour = \text{green})$

Complexes. A conjunction of selectors forms a complex. Instances are represented by complexes. The following are examples of complexes:

$[\,(shape\text{=sphere}) \wedge (weight > 5) \wedge (colour = \text{green})\,]$
$[\,(weight > 5) \wedge (weight \leq 10) \wedge (colour = \text{red})\,]$

Covers. Covers are disjunctions of complexes and they are used for representing the induced hypotheses. The following is a possible cover:

$$\{\,[\,(shape\text{=sphere}) \wedge (weight > 5) \wedge (colour = \text{green})\,] \vee$$
$$[\,(weight > 5) \wedge (weight \leq 10) \wedge (colour = \text{red})\,] \vee$$
$$[\,(shape\text{=pyramid}) \wedge (colour = \text{green})\,]\,\}$$

The reason why these high-level structures are called covers has to do with the inductive method adopted in the algorithm. AQ is considering each positive example in turn, attempting to generalise it as much as possible, excluding at the same time all the negative examples. During this process, it builds a cover, which is a description that includes all positive examples and excludes any negative ones. Each time the algorithm is examining a positive example, which is not "covered" by the cover constructed so far, a new complex is added to the cover. This complex is produced by the STAR algorithm and is potentially selected out of a number of candidate ones, according to a number of criteria, e.g. "coverage" (i.e. number of positive examples covered), simplicity (usually measured by its size, i.e. number of selectors). In the end a disjunction of those *best-complexes* (i.e. a cover) is induced, which covers all positive and excludes all negative examples of the concept[5]. This is taken as the best approximation to the concept. Figures 2.9 and 2.10 describe the basic AQ and the STAR algorithm.

[4]APC is an improvement of the *Variable Logic system 1 (VL_1)*, which made use of propositional calculus expressions.

[5]Some versions of the algorithm allow for a misclassification error, in order to handle noise in the training set.

Input: A set of examples E, a set of attributes A, a set of class values C, user-defined criteria LEF for the selection of the best complexes, user-defined star size m.
Output: A set of covers, one for each class in C.

For each class c_i in C do:

1. **Split** E into positive POS and negative NEG examples of c_i.
2. Set $COVER$ to empty set.
3. **While** POS is not empty **do**
   (a) Randomly **select a seed** example E_+ from POS.
   (b) Use the **star generating** algorithm $STAR(E_+, NEG, m)$ to generate a set of complexes of size m (called the $STAR$), which cover E_+ and exclude all examples in NEG[a].
   (c) Use LEF to **select the best complex** *BEST-COMP* in $STAR$.
   (d) **Append** *BEST-COMP* to $COVER$, as a new disjunct.
   (e) **Subtract from** POS all the examples covered by *BEST-COMP*.
4. **Return** $COVER$.

[a]Note that the algorithm assumes that no contradictions exist in the training set.

Figure 2.9: Design of the basic AQ algorithm

The main idea underlying the algorithm is the generation of stars and covers. Each of the positive examples acts potentially as a seed expanded against the negative examples. Thus there is an initial generalisation stage, which produces a star from the seed example. Following the selection of "the best" element of this star, the algorithm performs a specialisation of this "best complex", making use of the subset of positive examples that is covered by it. In terms of space search, this could be described by a brief "oscillation" within the space existing between the seed and the negative examples. The resulting star[6] is "optimal" in the sense that it covers completely and in a minimal way all the positive examples, which are covered by the initial very general star. This type of search, which combines a generalisation and a specialisation stage is called *beam search*.

[6]Note that the existence of noise, may render the generation of such a star impossible.

$STAR(E_+, NEG, m)$

1. Set $STAR$ to contain the empty complex.
2. Randomly select a negative example E_- from NEG, covered by a complex in $STAR$[a].
3. Find the set $EXTENSION$ of selectors, which cover E_+ and exclude E_-.
4. **Specialise** $STAR$, by replacing it with its cartesian product with $EXTENSION$.
5. **Trim** $STAR$ by removing:
   (a) Nil complexes (i.e. complexes containing contradictions).
   (b) Complexes subsumed by others in $STAR$.
   (c) All but the m best complexes.
6. If $STAR$ excludes all examples in NEG **return** $STAR$. Otherwise continue at **step 2**.

[a] At the beginning the empty complex covers all the examples.

Figure 2.10: The $STAR$ algorithm

Finally, an interesting aspect of the algorithm is the way in which it handles **completeness and consistency**. It attempts to achieve completeness, maintaining consistency at all times. Consistency is the primary goal and, at least in this standard version of the algorithm, there is no tolerance of inconsistent descriptions, making the algorithm incapable of coping with noise. The second goal is completeness which must also be achieved in order for a valid concept description to be generated. This is an absolute requirement too and must be relaxed if noise in the training set is to be handled.

2.4.2 The ID3 Algorithm

ID3 makes use of a simple learning method, called the *Concept Learning System (CLS)* [Hunt *et al.*, 1966], that was initially designed to perform single-concept learning. This method uses a decision tree to represent the acquired knowledge. Each node of the tree represents an attribute of the concept and

Input: A set of examples E, a set of attributes A, a set of class values C.
Output: A decision tree T.

Initially $S = E$

Dichotomise(S)**:**

1. **If** all the members of S belong to the same class make current node a *leaf node* and stop dichotomisation.
   Else select attribute a_i that does best in discriminating between positive and negative examples in S.
2. **Create** new node in the tree T.
3. **For each** value of a_i: v_{ij} **do:**
   (a) **Dichotomise**(S_{ij}), where S_{ij} is the subset of examples corresponding to v_{ij}.
4. **Return** T.

Figure 2.11: Design of the basic CLS algorithm

each branch one value of the attribute. Each attribute-value[7] is examined as to how well it does in discriminating between positive and negative examples of the concept. Figure 2.11 describes briefly the algorithm.

As an example of the process, assume the following training set, describing the concept of a '*ball*':

Positive Examples:

Shape	Size	Bouncy
round	small	Y
round	big	Y
round	small	N

Negative Examples:

Shape	Size	Bouncy
square	small	N
round	big	N
triangular	small	N

[7]The original version of ID3 allows only nominal attributes.

Figure 2.13: The second step of the generation of a decision tree, by CLS.

The only case in which discrimination is incomplete is when *shape* = *round*. If '*bouncy*' is selected as the next attribute then the tree will take the form of figure 2.13.

Finally, if *size* is used perfect discrimination will be achieved (figure 2.14).

It is possible to fail to achieve perfect discrimination after having used all the attributes. In this case, CLS cannot provide an accurate description of the concept. This inefficiency has been overcome in some versions of ID3 by allowing

Figure 2.14: The decision tree finally generated by CLS.

probabilistic answers (e.g. IF *shape* = *round* THEN *ball*, by 75%).

Another source of inefficiency, which can be detected from the previous example, is that, although a fairly good discrimination has been achieved by the first attribute that was selected, the quest for perfect discrimination has led the algorithm to use all 3 attributes. This may not be a substantial problem in such a simple situation, but it is bound to cause inefficiency in large learning problems. Moreover, it forces the algorithm to make use of attributes that are not characteristic of the concept (e.g. the 'size'), making the algorithm very prone to noise in the training set.

In order to choose the best discriminating attributes, an information theoretic criterion is used, namely Shannon's entropy, which is given by the following formula:

$$\text{entropy} = -\sum_{i=1}^{n} v_i^+ \log(\frac{v_i^+}{v_i^+ + v_i^-}) + v_i^- \log(\frac{v_i^-}{v_i^+ + v_i^-}) \tag{2.4}$$

where n is the number of possible values that an attribute can take and v_i^+ and v_i^- are the number of positive and negative examples for each attribute-value[8]. Chapter 3 presents several variations of this measure.

[8]Usually the notation p_{ij} is used for the proportion of instances of class j having the i^{th} value of the attribute. Note also that log is used for $\log_2$ in the entropy calculations.

To illustrate the use of entropy in calculating the discriminatory power of different attributes, assume that one wants to find the most discriminatory out of the three attributes in the '*ball*' example used in the previous section (at the beginning of the learning process). The entropy for each one of them will be calculated as follows:

$$\begin{array}{ll}
\text{entropy}(\textit{shape}) & -(3\log(3/4)+1\log(1/4))-(0+1\log(1/1)) \\
 & -(0+1\log(1/1))=2.25+0+0=2.25 \\
\text{entropy}(\textit{size}) & -(2\log(2/4)+2\log(2/4))-(1\log(1/2)+1\log(1/2)) \\
 & =2.8+1.4=4.2 \\
\text{entropy}(\textit{bouncy}) & -(2\log(2/2)+0\log(0/2))-(1\log(1/4)+3\log(3/4)) \\
 & =0+2.25=2.25
\end{array}$$

The attribute that will be selected is the one with the lower entropy. In this case '*shape*' and '*bouncy*' have the same entropy and the algorithm will have to choose between the two, based on other criteria (e.g. number of attribute values).

ID3 is a greedy divide-and-conquer algorithm, performing a hill-climbing search, which uses the entropy criterion as a search heuristic. This has proved an efficient method for inducing effective discrimination functions (i.e. the decision trees) [Quinlan, 1983a]. Since its birth, the algorithm has attracted much research in ML and has been used in numerous learning systems. The original version of the algorithm has a few problems (e.g. overspecialisation, only nominal attributes and binary classes are used, etc.). Some of these problems have been investigated, resulting in several improved versions of the algorithm. A few of those extensions are discussed in the following section.

2.5 Extensions

The first SBL algorithms, like the ones described in the previous section, could only deal with simple situations and did not perform very well on real data. The research which followed that initial stage addressed some of those problems, improving the standard versions of the algorithms. Some of the most important extensions to the basic algorithms are the following:

- Handling numeric data, by discretisation.
- Prunning (i.e. reducing) the induced concept-description, in order to minimise the effect of noise.
- Incremental learning, in the presence of new information.
- Constructive induction, based on primitive features.

This section discusses each of those extensions, describing the work done so far.

2.5.1 Handling Numeric Data

One of the basic problems of ID3 and AQ was the fact that they could only deal with nominal (and tree-structured, in the case of AQ) attributes. Since, most of the real-world classification problems involve numeric data, one of the early changes to the algorithms was the handling of numeric attributes. The way in which this is done is by discretisation of the attributes into value-ranges, which makes it possible to process them as if they were nominal ones. Usually, only two ranges are used, leading to binary attribute-tests. Some of the discretisation methods, which are used in the algorithms that are examined in the project, are described in chapter 3.

A similar problem exists with the handling of numeric classes. Most of the learning algorithms cannot handle numeric classes, which can take an infinite number of values. An exception to this are *regression trees* [Breiman *et al.*, 1984] and the NewID algorithm, described in chapter 3.

2.5.2 Overfiting and Pruning

One of the main deficiencies with ID3 is that it cannot handle noise in the training set. It always tries to achieve perfect discrimination, resulting in very complicated trees in the case where the examples do not provide clear-cut discrimination points for the different classes. In order to avoid this, several methods have been proposed, which provide mechanisms for stopping the growth of the tree or for pruning some of its already grown branches, when these do not seem to provide any substantial increase in discrimination.

One of the early attempts to enhance decision trees by stopping the growth (rather than by pruning), was presented by Quinlan [Quinlan, 1983b] and uses the χ^2 significance test at each node of the tree. The way in which he approximates this, is by means of the following formula:

$$\sum_{ij}(n_{ij} - e_{ij})^2/e_{ij}$$

where n_{ij} is the number of examples with class c_i and attribute value v_j, for the attribute handled at the node, and e_{ij} is the expected number of examples, under the null hypothesis that '*the sub-populations of examples created when we split E are drawn from the same population as E*',where E denotes the sub-set of examples at the node. Under this hypothesis $e_{ij} = (\sum_j n_{ij} \times \sum_i n_{ij})/n$.
The problem with this method is that the proposed statistic is not a good approximation of χ^2 for small training sets. [Niblett and Bratko, 1987] propose an

improvement to this measure, by examining the *contingency tables* of the examples, per attribute-value and class, at each node of the tree. The idea is to try and calculate the sum of probabilities of possible distributions of examples, which leads to the calculation of χ^2.

Quinlan [Quinlan, 1986b] has also presented some work in pruning, and proposed '*an optimisation criterion that offsets the complexity of the tree against its observed classificational accuracy on the training examples*'. This is a similar approach to the one taken in [Niblett and Bratko, 1987], which is based on a measure of the classification error $e(N)$ at each node N of the tree. Pruning takes place when

$$e(N) \leq \sum_{i=1}^{k} p(v_i)e(E_i)$$

where k is the number of possible classes, $p(v_i)$ is the probability of the ith value of the attribute and $e(E_i)$ the classification error at the node corresponding to v_i. Note that the error estimates are associated to the number of examples falling into a certain class.

For the calculation of $e(N)$, the following formula is devised, under the assumption that the attribute can take only two values[9] and the prior distribution of the k possible classes is uniform[10]:

$$e(N) = (n - n_c + k - 1)/(n + k)$$

which takes into account the number of classes k and the proportion of examples n_c corresponding to a class c.

The devised formula is used in a pruning algorithm, which can be split into two parts. The first is a recursive calculation of the classification error at each node, starting from the roots and ending at the leaves. The second reconstructs the decision tree, starting from the leaves and working its way to its root. At each stage the algorithm uses the pruning criterion to decide whether a sub-tree should be pruned or retained. Experiments with pruning methods have shown that a significant increase in classification accuracy is possible [Niblett and Bratko, 1987].

A similar overspecialisation problem exists with AQ, which strives for *completeness* and *consistency*. One of the descendants of AQ (CN2), which is used in this project and is described in chapter 3, overcomes this problem by changing the basic algorithm, in order to achieve "pre-mature" search-stopping, when no significant improvement is observed.

[9] [Niblett and Bratko, 1987] claim that the extension to multiple values is straightforward.

[10] The assumption of initial uniform class distribution is not true for internal nodes of the tree and an extension to the basic calculation is proposed in [Niblett and Bratko, 1987] .

2.5.3 Incremental Learning

Another major problem with ID3 is the fact that the decision tree cannot be easily updated in the light of new data. This is necessary in incremental learning situations (e.g. control learning), where data is provided to the system continuously. In such a situation, ID3 would need to recalculate a new tree each time new examples were provided, reconsidering all the previously examined examples. This process is computationally expensive, especially as the training set increases.

The first attempt to develop an incremental version of ID3 was made by Schlimmer and Fisher [Schlimmer and Fisher, 1986], who developed an algorithm named ID4. The new algorithm practically rebuilds the decision tree each time, but it has the advantage that it stores the class distributions at each node, which can be used to calculate the new entropy-measures efficiently without needing to re-examine the training instances. The way in which ID4 works is by updating the probability distributions at each node, starting from the root and discarding the subtree of a node, when a better discriminating attribute exists.

The main problem with ID4 is that it does not always generate the same tree as the pure ID3 algorithm would generate, given the new and the old information. This can happen when the scores of the attributes at some node are not substantially different and the instances are presented in such a way as to change the ordering of the attributes each time. This may result in a repetition of deletion and regeneration of subtrees, which does not converge to a stable situation.

In order to solve that problem, Utgoff has developed another incremental version of the ID3 algorithm named ID5R [Utgoff, 1989]. The first important feature of this algorithm is that it will always produce the same decision tree as the original algorithm would generate. ID5R maintains the same statistical information as ID4 at each node and it also looks for changes in the orderings of the attributes. However, instead of replacing the old subtrees with new ones, it restructures them, moving the new '*best attribute*' at the root of the subtree. The main advantage though of ID5R is that it maintains information about the instances that it examines, at the leaf-nodes. This is done by storing those attribute-tests that do not appear in the path to the node.

Recent versions of the AQ algorithm also perform incremental induction. For example, the algorithm AQ15 that is used in the project, stores the class probability distributions associated with each complex in the cover and accepts the covers as input, together with the new instances. Starting with those covers as the initial hypotheses and using the stored statistical information, the algorithm generates the new set of covers.

2.5.4 Constructive Induction

As mentioned above, constructive induction is the process by which new features are generated, based on the initially specified feature set. There are two main different approaches to the problem:

1. **Knowledge-driven feature construction**
2. **Data-driven feature construction**

In knowledge-driven feature construction background knowledge is provided to the learning system, determining the way in which features can be combined to form new ones. This method is used in AQ15 and is described in section 3.6. Data-driven feature construction is more difficult and it makes use of statistical clustering techniques, for the discovery of new attributes. An example of this approach is Rendell's conceptual clustering system PLS0, described in [Rendell, 1985].

In PLS0 the inductive process is subdivided into a number of stages, which lead gradually to the inductive goal. At each stage the system moves to a higher level of abstraction, '*imposing constraints, reducing complexity, extracting meaning and increasing regularity*'. All of these transformations are very important for the inductive task, since they enable it to discover "hidden" information in the training set and arrange it in a meaningful way. There are thus three levels of abstraction involved in the feature construction that PLS0 performs:

1. **Subobject Relationships**
   In this first stage, the primitive[11] space is arbitrary partitioned into subspaces, for which the utility[12] is calculated.
2. **Pattern Classes**
   In this level, similarities are identified between the calculated subspace (or subobject) utilities, resulting in the construction of more general classes. This is achieved by the extraction of common patterns, using the clustering method.
3. **Pattern Groups**
   At this final stage, some transformation operators are applied (e.g. rotation of the pattern in pattern recognition) on primitive members of the classes, in order for similar classes to be grouped together.

[11] The initial features are assumed to contain very little encoded knowledge about the characteristics of the concept.

[12] The reader is referred to the description of PLS1 in section 3.4 for the definition of this measure.

Constructive induction has recently been realised as an important research area, which can provide a solution to the "*feature-acquisition bottleneck*" of ML.

2.6 Computational Learning Theory (CLT)

2.6.1 Valiant's Probably Approximately Correct (PAC) Learning Model

In contrast with the large amount of work that has been done in formalising and developing concept learning systems, very little has been done to measure the complexity of the learning task. One of the first who attempted to do so, was Les Valiant, who introduced a deductive method for automatic program acquisition [Valiant, 1984b], [Valiant, 1984a]. The most important element of this work is the analysis of the learning task, in order to assess the performance of the method and explore the potential for improvement. This resulted in a measure of the complexity of the learning task in terms of the number of examples that an algorithm needs in order to induce a correct (i.e. complete and consistent) hypothesis. This is called the *sample complexity* of the problem and, as Valiant has proved, in the worst-case analysis it is exponential in nature. However, if one assumes that there is a fixed (though unknown) relative frequency[13] at which positive examples of the concept[14] to be learned occur, one can calculate the number of examples from which the algorithm can **probably** generate a hypothesis that is a good **approximation** to the concept.

On this basis, a definition is proposed for a *learnable* concept:

> *a concept is learnable if and only if an algorithm can be developed, which will be able, in polynomial time, to generate a hypothesis that will correctly classify most (the error allowed is also specified) of the instances of the concept.*

What makes Valiant's model especially interesting is the calculation of an upper bound for the number of examples needed to be considered by an algorithm, in order to arrive to a correct (in the above probabilistic sense) hypothesis of a learnable concept. This upper bound depends linearly on the size of the hypothesis space and an independent parameter, which specifies the acceptable error limits.

[13]The assumption of a fixed probability distribution of examples is likely to hold for most natural learning problems and since it is not required to be known, it is not expected to be a real limitation.

[14]Valiant considers concept learning as a subset of his more general definition of learning as an automatic program acquisition process.

[Valiant, 1984a] uses this model to analyse the performance of the learning method on the following standard types of concepts:

- *disjunctive concepts* **(DNF),** containing only disjunctions of atoms[15],
- *conjunctive concepts* **(CNF),** containing only conjunctions of atoms,
- **k-DNF,** made of disjunctively connected blocks, which can contain up to k conjunctively connected atoms,
- **k-CNF** (equivalent to k-DNF) and
- *internally disjunctive concepts*, conjunctively connected atoms, each of which correspond to one attribute, but allows the assignment of more than one value to it.

Assuming boolean attributes and a finite hypothesis space, Valiant proves that the method is able to learn the concepts efficiently (i.e. in polynomial time). This assumption, however, limits substantially the applicability of this analysis and it has been relaxed in later research.

2.6.2 Using the PAC Learning Model to Measure Inductive Bias

[Haussler, 1988] presents a method of using Valiant's probabilistic framework to show that simple inductive learning algorithms can perform near-optimally. At the same time he relaxes Valiant's assumptions restricting the structure of the hypothesis space, using *inductive bias* to overcome the problems caused by this relaxation. Inductive bias, in this context, stands for the restrictions to the hypothesis space imposed by the nature of the learning algorithm (e.g. heuristics, initial starting state, etc.)[16]. In order to quantify the strength of this bias and measure the performance of the algorithms, he makes use of the '*growth function*', introduced by [Vapnik and Chervonenkis, 1971]. The outcome is:

> *a measure that relates the strength of a bias to the performance of learning algorithms that use it, so that it will be useful in analysing and comparing learning algorithms [Haussler, 1988]*

In contrast with Valiant's analysis, Haussler allows for all three types of attributes (i.e. nominal, numeric and tree-structured, he also assumes that nominal attributes can be represented by corresponding tree-structured ones). As a result, infinite hypothesis spaces are also allowed.

[15] Atom is a term used in CLT for attribute-tests.

[16] For a more detailed examination of inductive bias the reader is referred to [Utgoff, 1986].

In order to expand the model to infinite hypothesis spaces, the '*growth function*' of the hypothesis space is used, symbolised by $\Pi_H(m)$, where $1 \leq m \leq |\mathrm{X}|$, H is the hypothesis space, |X| the cardinality of the example set and m is the cardinality of the subset of |X| that is being examined (i.e. the minimum size of which is used for the determination of the sample complexity). This function measures the maximum number of dichotomies of a subset of the example set using all hypotheses from the hypothesis space. The notion of '*dichotomy*' here is similar to that used in decision trees. Examples (or instances) are labelled as '+' or '−' according to whether their attribute-values adhere to a specific hypothesis. For example, assume a numeric attribute (e.g. size) and three instances that take the values 3, 4, 5 accordingly. The hypothesis: $2 < size < 6$ will label all three instances as '+' (in decision-tree terms, the three instances would "belong" to the same node). The hypothesis $4 \leq size \leq 5$, however, labels the first instance as a '−' and the other two as '+' (in a decision tree the first instance would belong to a different node than the other three).

Having three or more distinct attribute-values in the training subset, there are a number of dichotomies which cannot be achieved by any hypothesis of the hypothesis space. In the previous example there is no way to get the second instance labelled as '−' and the other two as '+' (in decision-tree terms, one cannot have instances 1 and 3 in the same node, without having the second one too). Thus the growth function $\Pi_H(m) < 2m$ for $m \geq 3$. This leads to another definition, that of the Vapnik-Chervonenkis dimension $\mathrm{VCdim}(H)$ of a hypothesis space H, which is the cardinality of the largest subset of the example set that can be '*shattered*' (i.e. all possible dichotomies can be achieved) by H. Thus, for single-attribute, conjunctive concepts, $\mathrm{VCdim}(H) = 2$, irrespective of the type of attribute, allowing for an infinite hypothesis space. Similarly, Haussler proves that if the instance space is defined by $n \geq 1$ attributes and the concept to be induced is conjunctive then

$$n \leq \mathrm{VCdim}(H) \leq 2n$$

and

$$\Pi_H(m) < \left(\frac{1}{2}em/n\right)^{2n}, \text{ for all } m \geq 2n$$

where e is the base of the natural logarithm.
Note : Similar bounds hold for the other four types of concepts that are being considered.
Based on this result, Haussler claims the following:

> *'Because it reflects limitations on the power of discrimination and expression inherent in the hypothesis space H, the growth function $\Pi_H(m)$ is a natural way to quantify the bias of learning algorithms that use H. It is also a useful measure of the bias.' [Haussler, 1988]*

The final outcome of this work is a new measure of the maximum probability of learning a hypothesis with an approximation error greater than ε (where $0 < \varepsilon < 1$):

$$2\Pi_{H_C^L(m)}(2m)2^{-\varepsilon m/2}$$

where C is a class of target concepts
L is the learning algorithm
m is the number of examples in the random example set, $m \geq 1$
and $H_C^L(m)$ is the '*effective hypothesis space of L for target concepts in C and sample size m*', denoting the set of hypotheses generated by L for C and m.

The importance of this measure, as well as the ones derived from it, is that it holds for any learning algorithm and it incorporates a measure of its inductive bias. Thus, it limits the complexity upper bounds to more realistic and near-optimal levels.

2.6.3 Criticisms and Extensions to the PAC Learning Model

There are a few problems with the basic PAC model, which make it unusable for real problems. Some of these are the following:

1. The estimate of the sample complexity provided by the model is a *worst-case* one.
2. *Noise-free* data are assumed.
3. The model cannot be readily applied to:
   (a) *Incremental learning systems.*
   (b) Learning problems with *multi-valued* functions.
   (c) Systems that use *background knowledge.*

The realisation of these problems has led to a number of extensions to the basic algorithm and the development of new theoretical learning models. The following are a few examples of such models:

- *Distribution specific model:* The distribution on the instance space is taken into account for the calculation of the sample complexity.
- *"Probability to mistake" model:* This model specifically refers to incremental algorithms, using as a performance measure the probability that the algorithm will misclassify the n^{th} randomly selected training example.

- *"Total mistake bound" model:* This is another incremental learning model, which uses as a performance measure the total number of misclassifications that the algorithm will make in the worst-case (*worst case mistake bound*).

[Haussler, 1992] gives a more detailed account of recent research in the field, including a number of interesting new models.

2.7 Summary

This chapter has provided an overview of the work that has been done in the field of empirical concept-learning, which is the type of learning used in this project. Concept-learning has been an attractive and very active research area during the last decade, mainly due to its applicability to real-world problems. In this type of learning, the task of acquiring the description of a concept is seen as a search problem, where the search space is the set of all possible descriptions and the goal is the identification of the one which best fits the training data. In that context, the learning algorithm provides the operators for moving between states in the search space.

The problems that are being attacked with concept-learning methods (e.g. prediction, diagnosis, etc.) have been approached with statistical methods before. The field of statistics which deals with this type of tasks is called classification and its goal is the derivation of a function which fits a set of given data. This function provides a mapping between objects and the classes to which they belong. Objects in statistical classification are represented in terms of a set of features, which are also used in the definition of the classification function.

Empirical concept-learning bears a number of similarities to statistical classification, examples of which are the following:

- The use of features/attributes to describe objects and define concepts.
- The use of various statistical techniques (the heuristics used in the learning algorithms) in order to identify similarities between objects and generalise them into concept-descriptions.
- The correspondence between the search space used in learning and the feature space used for the definition of the classification function.

Thus, the two approaches can be seen as equivalent, with the exception of the types of attributes that they use. Statistical approaches have focused on numeric attributes, while nominal ones are favoured in learning. This difference is a result of the way in which classification functions and concept-descriptions are defined.

The former are mathematical expressions, while the latter tend to be represented as rules, using a logic-like notation.

Most of the research done in empirical-concept learning has concentrated on few algorithms, two of which have been described in this chapter. These two algorithms have been very popular in ML research and form the basis of the learning systems that will be described in the following chapter. In terms of classification, they both can be seen as conceptual clusterers, which divide the feature space into orthogonal hyperrectangles. Each such rectangle contains examples of one class and, together with the rest of the rectangles corresponding to the class, is used as its definition.

Concept Learning Theory is an area, which deals with the complexity of the concept-acquisition task and has recently become very popular. It attempts to formulate mathematically the learning task and examines the learnability of different types of concepts. Learnability is defined in terms of sample complexity, which determines the minimum number of instances that an algorithm will need to examine in order to induce a reliable concept-description. Ideally, such an analysis will result in a learning curve, showing the performance of the algorithm. The models that have been developed so far however are not usable in practice, for the reasons given above.

One final point that needs to be made here is that the algorithmic analysis performed by computational learning models is different from the regular computational complexity analysis. The former examines the sample, while the latter the computational complexity of an algorithm. Chapter 3 uses the latter type of complexity analysis, in order to derive a theoretical estimate of the performance of the five learning algorithms that are used.

Chapter 3

The Algorithms

3.1 Introduction

The purpose of this chapter is to introduce the five algorithms that have been examined in the project. These algorithms have been selected out of a large list of available ones[1] for several reasons. The main one is the fact that they are representative of a large portion of the work done in the field. As a result, they have been very popular among the ML researchers, who have examined and improved them substantially. Moreover, some of these algorithms (especially the ones based on decision trees) have been favoured in applications which incorporate ML components. Finally, the versions of the algorithms acquired, are well suited to the purposes of the project, both because they are all written in similar, procedural programming languages (C and Pascal), reducing thus the possibility of a language inefficiency, and more importantly because being the original versions provided by their developers, they are likely to be quite robust and complete. These two features are very important for carrying out a fair comparison of the algorithms.

The algorithms that are used in the project are all *concept-learning* ones. This means that they induce a concept description (in the form of a decision-tree, a cover or a hyperrectangle), based on a set of positive and negative examples of the concept, expressed in terms of a fixed number of features/attributes. The domain (type and value-set) of each feature (including the class/concept, which is treated as a special feature), must usually also be specified. As it will be shown in the following sections of the chapter, this later requirement is important, as it guides the algorithm to process the feature in a meaningful and efficient way.

With respect to the distinction made in earlier chapters between *generalisation-* and *specialisation-based* inductive algorithms, this project is interested in

[1] See appendix A for a brief description of the available algorithms.

both. The two algorithms which induce decision-tree concept descriptions (i.e. **NewID** [Clark and Niblett] and **C4.5** [Quinlan]) and the conceptual clustering one (**PLS1** [Rendell]) use specialisation techniques, while the ones which induce covers (i.e. **AQ15** [Michalski] and **CN2** [Clark and Niblett]) are based on generalisation. Previous comparisons (see [O'Rorke, 1982] and [Rendell, 1986]) have argued that specialisation algorithms are more efficient than generalisation ones and this is one of the issues that this work will revisit.

Finally, a major commonality of the five algorithms, which facilitates their comparison, is the coverage of the concept description that they provide. Although some of them (e.g. AQ15) use slightly more expressive schemes than others, they all achieve *orthogonal coverage* (see section 2.3.2), similar to that of a set of hyperrectangles. This is the result of assuming feature independence and not considering the relation between them[2].

Following this introductory description of the algorithms, a separate section of the chapter is devoted to each individual algorithm. These sections will adhere to the following format:

- A brief **introduction**, containing mainly historical and theoretical information about the algorithm.
- The **description of the algorithm**. This subsection will describe the algorithm in detail, focusing on its peculiarities.
- **Analysis** of the algorithm. Since the main interest of the project is time-performance, this subsection will concentrate on the computational complexity of the algorithm.
- **Concluding** remarks, about the quality of the design and the implementation of the algorithm.

3.2 NewID

3.2.1 Introduction

The first of the examined algorithms is a fairly standard version of the popular concept-learning algorithm, **ID3** [Quinlan, 1983a], (section 2.4.2) written in C. NewID [Niblett, 1989], [Boswell, 1993b] was developed at the Turing Institute, by Tim Niblett, and was incorporated in the Esprit Project MLT (Machine Learning Toolkit). Like ID3, it uses examples, in order to induce a decision-tree description of the concept. However, NewID incorporates a number of features, which are additional to the standard version of ID3 and provide solutions to problems that

[2]This is a very important limitation of the algorithms, in comparison to some of the statistical classification methods.

have been identified with the basic algorithm. This results to a more useful system for solving real-world problems.

3.2.2 Description

The basis of NewID is the same as that of ID3. Figure 3.1 describes this basic algorithm, as it appears in [Niblett, 1989] and [Boswell, 1993b]:

Input: A set of examples E, a set of attributes a_i, a class c, a termination criterion $TE(S)$ where S is a set of examples and an evaluation function $IDM(a, S)$ where a is an attribute and S a set of examples. The termination criterion is usually that all examples in S have the same class value.
Output: a decision tree.

1. Set the current examples S to E.
2. If S satisfies the termination condition ($TE(S)$) **halt** and return the tree.
3. For each attribute a_i determine the value of the function $IDM(a_i, S)$. With the attribute a_j that has the largest value of this function divide the set S into subsets by attribute values. For each such subset of examples S_k recursively re-enter at step (1) with E set to S_k. Set the subtrees of the current node to be the subtrees thus produced.

Figure 3.1: The Basic ID3 Algorithm

The Evaluation Function

Originally, it was suggested that more than one evaluation function be included in NewID [Niblett, 1989]. The user would then have to choose one of those functions to use. In practice this would mean that he/she would try more than one of them out, in order to find the one that suits his/her problem best. The idea behind this is that different heuristics perform better than others on specific types of problems.

One of the proposed choices was a *mutual information* measure, which is a

small extension of the original entropy measure[3]:

$$I(C|A) = ev(C \wedge A) - ev(A) = \sum_j p_j \log p_j - \sum_{ij} p_{ij} \log p_{ij} \qquad (entropy) \quad (3.1)$$

$$I(C : A) = I(A : C) = ev(C) + ev(A) - ev(C \wedge A) = \sum_{ij} p_{ij} \log p_{ij} - \sum_j p_j \log p_j - \sum_i p_i \log p_i \qquad (mutual\ information) \quad (3.2)$$

where $I(C|A)$ is the *conditional information content* of class attribute C, given attribute A, $I(C : A)$ is the mutual information between C and A, $ev(M)$ is the entropy value (measuring the information content) for message M, p_i is the probability of an example falling in class c_i, p_j the probability of an example having the value v_j for attribute A and p_{ij} is the combined probability, over the subset at the parent node. The practical advantage of the mutual information measure over the simple entropy one is that it solves the problem of favouring attributes with many values[4].

The second choice was *quasi-utility*, introduced by Good and Card in [Good and Card, 1971]. They propose the combination of a utility measure with the informativeness one:

$$U = max_j \sum_i q_i u(i, j)$$

where q_i is the probability of class c_i and $u(i, j)$ the utility of accepting class c_j when the correct class is c_i. In the simple case where $u(i, j) = 0$ if $i \neq j$ and $u(i, j) = 1$ if $i = j$ the goal is to maximise q_i.

However, none of these alternatives were implemented in NewID. Instead the standard entropy measure was used.

The Termination Criterion

The criterion that is usually used for terminating the growth of a branch, is that the subset of examples corresponding to the leaf-node be in a single class. One problem with this *perfect-discrimination criterion* is that it leads to overfitting of the data, especially in the presence of noise. There have been several proposed solutions to this problem (section 2.5.2) most of which fall under two categories: *pre-pruning* (or *growth-stopping*) and *post-pruning*. In general, it is accepted that post-pruning is more effective than growth-stopping [Breiman *et al.*, 1984]. Because of that, only a post-pruning method (described later in the section) is implemented in NewID, keeping the termination criterion fairly standard. The only minor addition to it, is a check as to whether a *non-trivial* split exists. A

[3]This measure appears also in [Quinlan, 1983a], [Quinlan, 1986a], [Quinlan, 1993].

[4]Refer to section 3.3.2 for a more detailed explanation.

split on an attribute A is non-trivial if by splitting on that attribute non-empty example subsets are generated[5].

Additional Features

NewID incorporates a number of features which do not appear in the standard version of ID3. The most important of these are the following:

A. **Attributes**

The standard version of ID3 uses only nominal attributes. NewID can also handle **real**, **integer** and **ordinal** attributes. The idea of using numeric attributes appears also in antecedents of NewID (e.g. ACLS [Paterson and Niblett, 1982] and CART [Breiman *et al.*, 1984]). The problem with this kind of attribute is that there is an infinite set of potential splitting points and thus the splitting process has to be changed. Ordinal attributes on the other hand allow nominal attributes, of which the values can be provided in some natural order (e.g. temperature (low, medium, high), size (small, medium, large), etc.), to be treated as integer ones. This has the advantage of providing more compact and informative splits (i.e. ranges of values instead of lists of them).

Another type of attribute that has been used in inductive learning programs is the *tree-structured* one (see section 2.2.2). In this type, the values of the attribute can be naturally structured in a hierarchy. Common examples where this is useful is the taxonomy of animals or the shape of a geometrical object. Although not directly implementing it, NewID simulates this type, using **attribute ordering**[6]. This simply means that some attributes will be evaluated before others, when deciding for a split.

Finally, the class attribute is allowed to be numeric. Although this is very useful for many classification problems, there have been very few algorithms which implement it (e.g. regression trees [Breiman *et al.*, 1984]), because of the complications it imposes on the evaluation function and the termination criterion. The first effect of that on NewID is the replacement of the standard entropy measure with the sum of **class variances** corresponding to individual values of an attribute:

$$\sum_j variance_e(\{class(e)|attval(A,e)=j\}) \tag{3.3}$$

[5]The criterion actually becomes slightly more complicated, with the use of weights on examples (explained further on in the section). In that case the subsets produced from the split have to contain at least one example of weight ≥ 1.

[6]This is not exactly the same as having tree-structured attributes, where the relationship between subtypes in the hierarchy is much stronger than in NewID.

where e is an example with class value $class(e)$ and value $attval(A, e)$ for attribute A.
Minimising this measure, one achieves maximum discrimination, similar to that achieved using entropy. In addition to the modification of the evaluation function, NewID uses a special termination criterion for numeric classes:

$$\sigma(S) \leq 1/k \ \sigma(E)$$

where E is the initial set of examples, S the subset of examples at a specific node, σ the standard deviation in each of the two sets, with respect to the class attribute, and k a user-definable parameter.

B. **Examples**

NewID provides two major extensions to the format of the examples handled by the basic ID3 algorithm. The first one is the use of the "*unknown*" and "*don't care*" attribute values[7]. If an attribute is given the value "*unknown*" in an example, it means that its value is not available. The "*don't care*" value on the other hand, denotes the irrelevance of that attribute for the classification of the specific example. Examples with "*unknown*" and "*don't care*" values are handled specially at the evaluation and the splitting stage. The existence of an "*unknown*" value causes the replacement of the example from fractional examples (one for each attribute value), based on the distribution of the examples which have known values for this attribute and belong to the same class. "*Don't care*" values on the other hand are handled by duplication of the example in question, based again on the attribute-values found in the rest of the examples.

The second idea introduced in NewID is the use of *weighted examples*. The program allows the user to provide a weight for each of the training examples, specifying thus how characteristic it is of its class. This feature can be quite useful in cases where the user can judge the quality of the examples and use that information in order to guide the learning process, acquiring thus better classification results. However, misconceptions or misunderstandings of the user can have exactly the opposite effect.

C. **Splitting**

The use of numeric and ordinal attributes has affected the splitting part of the decision-tree building process. Numeric attributes cannot be handled in the same way as nominal ones. The solution adopted in NewID is to order the examples in ascending order, based on the values of the numeric attributes, and then look for the most appropriate value to split them into two subsets. This **binary-splitting** approach has the following effects:

1. Numeric attributes can be used more than once in a branch of the decision tree.

[7] Originally introduced in ACLS [Paterson and Niblett, 1982].

2. In the presence of noise, overfitting may be facilitated by extensive use of numeric attributes, near the leaves of the tree.
3. Near the root of the tree the evaluation function will favour nominal attributes, although they may be poorer discriminators than the numeric ones, because the later may need to be split into more than two ranges, in order to give a substantial information gain.

Niblett has also mentioned the use of **multiple-splitting** for numeric attributes [Niblett, 1989], but this was not implemented.

Binary-splitting can optionally be used on nominal attributes also. In that case, the value set of the nominal attribute is split into two subsets each time the attribute is used. If one was to search exhaustively through all the subsets of values for each attribute every time, this process would be very heavy computationally [Breiman *et al.*, 1984]. In NewID, however, a heuristic search takes place, which generates the two subsets by moving a single attribute value at a time from one to the other. The attribute value selected each time is the one which causes the largest increase in the entropy score of the attribute and the process stops when there is no more gain in entropy.

D. **Pruning**

The user of NewID is given the choice to post-prune the generated tree. The pruning method is a simple but effective one:

> A subtree is pruned if the additional classification accuracy that it provides, in comparison to the classification accuracy of its root node on the majority class, is below a user-defined threshold.

More sophisticated pruning methods have been proposed by several researchers (e.g. [Kononenko *et al.*, 1984] and [Niblett and Bratko, 1987]), but they have not been implemented in NewID. Pruning in NewID is also restricted to nominal classes. This is a reasonable restriction since overfitting of numeric classes can be avoided by appropriate setting of the termination threshold.

Evaluation of trees

Another useful facility provided by NewID is the evaluation of an induced tree on a test set. Test examples are of the same format as the training ones, including where necessary "*unknown*" and "*don't care*" values. The output of the evaluation process for numeric classes is a detailed count of all negative and positive misclassifications for each value of the class. In addition to that, averages for each class value and the total classification accuracy are given. Examples with "*unknown*" and "*don't care*" values are handled in a similar way as in the training process, possibly producing fractional counts. The evaluation output for numeric classes

is less detailed than for nominal ones. It provides a list of accuracy-levels and the corresponding percentage of the test examples that were classified correctly.

3.2.3 Analysis

Although NewID uses the basic design of the ID3 algorithm unaltered, its additional elements make it substantially different from its predecessor. This section aims to incorporate these new elements, presenting an outline of the design of NewID and analysing its behaviour, with respect to its worst-case computational complexity.

The Design

Figure 3.2 presents in a schematic way the main module of the algorithm. This is the same as the basic ID3. Examining further the three functions (i.e. the termination, evaluation and splitting ones) further, some of the new elements of NewID are introduced. However, only the design of the evaluation function (figure 3.3) is substantially affected.

Figure 3.3 shows the effect that the introduction of numeric attributes and classes has on the evaluation process. The other new features affect the algorithm at a lower level (e.g. the use of "*unknown*" and "*don't care*" values in the examples affects the calculation of the entropy/class-variance measure), which does not influence its design.

Figure 3.2: The Main NewID Procedure.

Figure 3.3: The Evaluation Module.

Worst-case Computational Complexity Analysis

Several people in the past have looked at the computational complexity of ID3 ([O'Rorke, 1982], [Quinlan, 1983a], [Quinlan, 1986a], [Clark and Niblett, 1989]) and have come to the conclusion that the basic algorithm is of order $O(m|A||E|)$, where m is the number of non-leaf nodes in the final decision tree. Clark and Niblett [Clark and Niblett, 1989] have also included the use of numeric attributes, with the additional requirement of sorting the examples in a subset, increasing thus the order of complexity to $O(m|A||E|\log|E|)$. These results however are based on a number of simplifications (e.g. that m is independent of $|E|$ and $|A|$), because their purpose was to give an indication of the complexity of ID3, rather than a thorough analysis of it. This section takes a closer look to the algorithm, adding the new features of NewID.

Based on figure 3.2, one can define the complexity of NewID, in terms of the complexity of its components:

$$\begin{aligned} COMP(NewID) &= m \times COMP(Dichotomise(S)) = \\ & m \times (\ COMP(TE(S)) + COMP(BEST\text{-}SPLIT(S,A)) + \\ & COMP(SPLIT(S, a_{max}))\ ) \end{aligned} \tag{3.4}$$

where m is the number of non-leaf nodes in the tree and $COMP(P)$ gives the complexity of a process P.

Let us examine each individual component of this:

1. **COMP(TE(S)):**

   Depending on the type of the class attribute, the termination criterion varies. For *nominal* classes class membership of the examples in S is tested, which is of order $O(|S|)$. If the class is *numeric* on the other hand, the standard deviation of S is used, which can be derived from the class-variance calculations which have taken place in the evaluation stage of the previous step[8]. An additional step, which has to be done is the 'triviality-test' (section 3.2.2), which in the worst case (i.e. when only trivial splits exist) takes time $O(|S||A|)$, because for each example each attribute value has to be checked. Thus, the overall complexity of this step is $O(|S||A|)$.

2. **COMP(BEST-SPLIT(S,A)):**

   The complexity of the evaluation process, can be decomposed, according to figure 3.3, as follows:

   $$COMP(BEST\text{-}SPLIT(S,A)) = |A| \times$$

[8]This will increase the space requirements of the program, as all the variances calculated will need to be stored.

$$(max(\ (COMP(SORT(S,a_i)) + COMP(BEST\text{-}BINARY(S',a_i)))$$
$$COMP(NOM\text{-}BINARY(S,a_i))\ ) +$$
$$max(COMP(VARIANCE(S,V)), COMP(ENTROPY(S,V))))$$

Examining each component in turn:

(a) **COMP(SORT(S,$\mathbf{a}_i$)):**

Assuming an efficient sorting algorithm, the complexity of this step is $O(|S|\log|S|)$. It is possible to avoid doing this calculation all the time, by storing an ordered list of the examples, according to each numeric attribute, at the beginning of the learning process. However, this would increase the space complexity of the algorithm substantially. In the following analysis, it is assumed that the sorting process takes place every time a binary split is examined.

(b) **COMP(BEST-BINARY(S',$\mathbf{a}_i$)):**

What is done in practice at this step is that all the possible splits (worst-case $|S'|-1 \simeq |S|$) are considered at the entropy/class-variance calculation and the best one is maintained. This is an exhaustive search method, but can be done efficiently, by minor adjustment of the probability estimates, when moving from one splitting point to the other. As a result, the complexity of the whole process is only $O(|S|)$. This is still higher than the complexity of the process taking place for nominal attributes and its results are not very satisfactory. Recent work (e.g. [de Merckt, 1993]) has been considering different approaches, mainly by replacing entropy with other heuristics, which are more suitable to numeric attributes.

(c) **COMP(NOM-BINARY(S,$\mathbf{a}_i$)):**

If a_i has the value-set V_{a_i} then, in the worst-case, the complexity of this step is:

$$\frac{3|V_{a_i}|^2}{8} \times max(COMP(VARIANCE(S,V)), COMP(ENTROPY(S,V)))$$

because at most $|V_{a_i}|/2$ values have to be moved from one subset of values to the other (only the best value is moved each time) and at each iteration an average of $|V_{a_i}| - |V_{a_i}|/4$ binary splits have to be evaluated.

(d) **COMP(VARIANCE(S,V)):**

Assuming that the variance is calculated efficiently (complexity $O(|S|)$), the complexity of this step will be $O(2|S|)$, in the case of a numeric attribute and $O(|V_{a_i}||S|)$, in the case of a nominal attribute with a value-set V_{a_i}. Thus, the overall complexity is $O(|V_{a_i}||S|)$.

(e) **COMP(ENTROPY(S,V))):**
The complexity of calculating the entropy for a nominal attribute is $O(|V_{a_i}||V_c||S|)$, as a result of the calculation of the sum of the probabilities that an example will have a specific attribute value in V_{a_i} and a specific class value in V_c.

As a result of the above, entropy gives the maximum complexity:

$$max(COMP(VARIANCE(S,V)), COMP(ENTROPY(S,V))) =$$

$$O(|V_{a_i}||V_c||S|)$$

Based on the above results, the complexity of the evaluation stage is:

$$|A| \times O(|S| \log |S| + |S|)$$

for numeric attributes,

$$|A| \times O(|V_{a_i}||V_c||S|)$$

for nominal ones, in the case of multiple splitting and

$$|A| \times O(\frac{3|V_{a_i}|^3|V_c||S|}{8})$$

for binary splitting of nominal attributes. The latter case gives the worst results for this stage.

3. **COMP(SPLIT(S,a$_{max}$)):**

At this stage, the splitting of the examples and the generation of new tree-nodes is taking place. The complexity of this process is $O(|V_{a_{max}}||S|)$, where $V_{a_{max}}$ is the value-set of a_{max}. In practice, however, this process may be avoided, by incorporating it in the previous (evaluation) stage.

4. **m**

The growth of the whole tree, is done by recursive calls on the main procedure, i.e. $Dichotomise(S)$, which will stop when all the branches have stopped growing. This process is very much dependent on the learning problem, but as an indicative worst-case, one can think of a problem where numeric attributes are used, the goal is perfect discrimination and an extremely skewed tree is generated. In that case, there would be $|E|$ leaves in the tree, corresponding to $m = (|E| - 1)$ calls of the procedure $Dichotomise(S)$, and an average of $|E|/2$ examples at each node.

Let as assume for example a simple problem, with one integer attribute, which has a different value for each example in the set (e.g. discriminating between odd and even integers):

Example id.	Attribute value	Class value
1	1	+
2	2	−
3	3	+
4	4	−

This set will produce a complete tree with three non-leaf nodes. The same would happen with any number of examples, so long as each has a different attribute value and class values follow the above pattern. Generating a complete tree causes a near-worst case situation with $|E|-1$ non-leaf nodes. The number of examples examined per node ranges from $\log|E|$ in the case of a balanced tree to $|E|/2$ in the case of an extremely skewed one[9].

Thus in the extreme case the overall complexity of the NewID algorithm would be[10]:

$$|E| \times (O(|A||E|/2) \quad + \quad |A| \times O(|E|/2\log(|E|/2)) \;) \simeq$$
$$O(\; \frac{|A||E|^2}{2}\log(|E|/2) \;) \tag{3.5}$$

On the other hand, if only nominal attributes are used, the worst-case situation is different. If binary splits are used, a similar situation to the binary splits of numeric attributes could occur, leading to the following worst-case estimate:

$$O(\frac{3|A||V_{max}|^3|V_c||E|^2}{16}) \tag{3.6}$$

where V_{max} is the largest value-set.
However, if multiple splits are used, the worst-case occurs when all attributes are used at each branch of the tree. Thus, under the simplifying assumption that all attributes have $|V_{max}|$ values, the largest number of nodes is

$$m = \sum_{i=0}^{|A|-1} |V_{max}|^i = \frac{1-|V_{max}|^{|A|}}{1-|V_{max}|}$$

and the average number of examples handled at each node is

$$|E|_{avg} = \frac{|A||E|}{m}$$

In this worst-case situation, the complexity of the whole algorithm is:

$$m \times O(|A|^2|V_{max}||V_c||E|/m) = O(|A|^2|V_{max}||V_c||E|) \tag{3.7}$$

Out of the three estimates (equations 3.5, 3.6, eq:newid.compl.fin.nom2) the one for numeric attributes gives a higher order of complexity (w.r.t. the size of the training set) and thus it can be used as the overall worst-case estimate of the algorithm.

[9]The shape of the tree is determined by the evaluation function.
[10]The complexity of the splitting component is omitted from the estimate.

3.2.4 Conclusion

The above analysis has shown that the new features introduced in NewID make a substantial difference in the worst-case computational complexity estimate of the algorithm. Other things remaining constant, NewID seems to be of over-quadratic complexity with respect to the number of examples in the training set. This is very different to the near-linear performance estimate of the standard ID3 algorithm, which has appeared in a number of previous papers. The factors contributing to this increase are the following:

1. The use of **numeric attributes**, increases the complexity of the evaluation stage. This is under the assumption that an exhaustive search for a binary split is used, which is an expensive discretisation method, considering all the values of the attribute in the training set, which in the worst-case is equal to the number of examples. This estimate could be improved by the use of a different evaluation heuristic for numeric attributes.

2. Another property of numeric attributes is that they can cause overfitting to the data, increasing the size of the tree and the complexity of the algorithm. If only nominal attributes were to be used, the longest path of the tree would be bound by the size of the attribute set. This is not the case with numeric attributes, which can be used more than once in the same branch of the tree.

3. Finally, the use of numeric class attributes could also facilitate overfitting, in the case where the user-definable termination parameter is given a very high a value. In the worst-case (i.e. when the parameter is given a very high value and each example has a different class value), this would cause single-example leaf-nodes. Combining this with the use of numeric attributes, makes more likely the occurrence of a worst-case situation, similar to that described in the computational analysis above.

The rest of the new features do not seem to affect the computational complexity of the algorithm substantially.

However, one has to be very careful with the use of this worst-case estimate. It assumes situations that do not arise in a typical problem, where the algorithm would be used (e.g. classification). For example, the existence of 100% noise is unrealistic. Thus, it is expected that an average case analysis or empirical tests would give very different results.

In general, NewID is a very interesting extension of the basic ID3 algorithm, introducing new features which make the algorithm usable in a larger variety of real world problems. These features however, may have a negative effect on the performance of the algorithm. More careful design and the introduction

of efficient heuristics in some of the elements of the algorithm could solve this problem.

3.3 C4.5

3.3.1 Introduction

C4.5 is another algorithm based on ID3. It was developed by Quinlan [Quinlan, 1993] and incorporates most of the improvements that he has done to the basic algorithm. The basic structure, however, remains the same and it is similar to NewID. Because of that, this section will concentrate on the differences of the two programs, mentioning in brief the new features of C4.5. C4.5 is also written in C.

3.3.2 Description

The Evaluation Function

The first important difference of C4.5 from ID3 and NewID is the evaluation function that is used. It has been noticed that the pure entropy and the mutual information measures favour attributes with many values [Kononenko *et al.*, 1984]. Such attributes decompose the training set into many small subsets which are bound to have relatively uniform class distribution. The significance, in terms of classification, of those attributes, however, may be very small, because many of the generated subsets contain examples of the same class.

[Kononenko *et al.*, 1984] have proposed as a solution to this problem the use of binary splits, even for nominal attributes (see section 3.2.2). Quinlan, on the other hand, proposes an additional adjustment of the evaluation function, incorporating a measure of the significance of the attribute [Quinlan, 1986a], [Quinlan, 1993]. He calls the new measure *gain ratio* (as opposed to the simple *entropy gain criterion*, which is the mutual information measure, (equation 3.2)) and he defines it as follows:

$$gain\ ratio(A) = gain(A)/split\ info(A) \quad (gain\ ratio\ criterion) \tag{3.8}$$

where A is the attribute that is being evaluated, $gain(A)$ is the mutual information measure and $split\ info(A)$ is a measure of the significance of the attribute, which is nothing else but $ev(A)$ (equation 3.2), being used once again. The resulting criterion is thus:

$$\frac{ev(C) - ev(C \wedge A)}{ev(A)} + 1$$

which reverses completely the initial use of the attribute information ! (equation 3.2)
As in the case of mutual information, the attribute which maximises this criterion is selected at each stage [11].

The new criterion seems to be insensitive to the number of attribute-values, but it has been argued [Mingers, 1989] that it favours unbalanced splits.

Examples

The first new feature of C4.5, with respect to the training set is *windowing*. What this term means is that the program can be told to construct a tree on a randomly selected subset of the initial training set and then use some of the examples which are being misclassified by this tree to generate a new one. This process is repeated until all examples are correctly classified, by the current tree. The initial subset of examples is called *window* and its size can be specified. The number of misclassified examples that are used each time to generate a new tree is also user-defined, but in any case at least half of those examples will be used. An additional option, allows the generation of a number of different initial trees and the selection of the best one.

Windowing must not be confused with incremental decision-tree building, as performed for example by ID5R [Utgoff, 1989]. The tree that is being built, each time new misclassified examples are considered, is built from scratch. C4.5 does not have a facility for incrementally updating trees.

A final comment about the format of examples in C4.5 is the handling of unknown attribute-values (which are not distinguished from '*don't care*' ones). These are handled in a similar manner to NewID, producing fractional examples.

Attributes

C4.5 can handle nominal, integer and real attributes. Binary-splits are used for numeric attributes, while for the nominal ones there is an option of generating *value-groups*. These are similar to the binary-splits used in NewID, but here there can be more than 2 subsets of values. The calculation of value-groups takes place separately at each node and is computationally heavy. As a solution to this, Quinlan proposes post-grouping of values, based on the generated tree. An additional problem with pre-grouping is that unnatural subsets of examples may occur. The result of this is that more effort may have to be put in finding a sufficient discrimination function. If this problem can be avoided, which is the

[11] Due to instability of the estimate at near-trivial splits, in order for an attribute to be used, its mutual information score has to be above the average.

case for post-grouping, value-groups can enhance the comprehensibility of the tree.

C4.5 cannot handle ordinal and structured attributes, or numeric classes[12]. This is one of the major differences with NewID, which can handle ordinal attributes and numeric classes and it can simulate structured attributes.

Pruning

After some initial experimentation with pre-pruning, using the χ^2 criterion [Quinlan, 1986b], Quinlan has adopted a post-pruning method, similar to, but more sophisticated than the one used in NewID. He uses, what he calls a *pessimistic* pruning method, which estimates the additional classification error, caused by a branch of the decision-tree. This calculation involves the approximation of the probability of error at each leaf, using a user-defined *confidence level* CF. What is interesting about this method is that it calculates the error probability based on the training set itself, rather than by using additional test cases (e.g. [Breiman *et al.*, 1984]).

An additional feature of the pruning method employed by C4.5 is that a subtree can be replaced by one of its subtrees, rather than by a single leaf-node. This allows for removal of incorrectly selected tests and flexible simplification of the tree.

Rule Generation

Perhaps the most interesting feature of C4.5, however, is its capability of turning decision-trees into production rules, very much like the ones produced by the CN2 algorithm (section 3.5). The advantage of this is that rules are more comprehensible by humans. This, however, is not the case when each path through the tree is used as a separate rule. Therefore, the initially generated rules have to pass through a number of simplification stages, including removal of non-significant conditions, restructuring of the rules and grouping them together according to the class that they predict. The final product of C4.5 is thus an **ordered** set of rules, each of which describes instances of one class. The reason why the rules have to be ordered is that, due to the simplifications, clashes between the rules are possible[13].

[12]For the latter, Quinlan has developed a different algorithm [Quinlan, 1992], which he has not incorporated in C4.5.

[13]See section 3.5 for a discussion on the comprehensibility of ordered and unordered rules.

Evaluation Module

Usage

Finally, the classification module of C4.5 incorporates some useful features:

- One can *interactively* provide test cases to the classifier, potentially including unknown attribute values, as well as probabilistic input, of the form: $v1 : p1, v2 : p2, ..., vn : pn$, where $v1$ ... vn are some of the values of an attribute and $p1$ ... pn their corresponding probabilities.
- C4.5 produces also *probabilistic output*, using certainty factors, similar to the ones used by some expert systems (e.g. MYCIN [Shortliffe, 1976]).
- *Soft thresholds* can be used, in addition to the "hard" ones for numeric attributes. This feature allows for probabilistic classification near the threshold value, avoiding thus misclassifications caused by slightly erroneous input.

3.3.3 Analysis

Design

The basic decision-tree building module of C4.5 is very similar to NewID. In terms of design the main difference of the two is that C4.5 cannot handle numeric classes. Thus, if one excludes the variance calculation from figure 3.3 the two flow-charts used to describe NewID can be used for C4.5 also.

Worst-Case Computational Complexity analysis

In computational terms, however, the algorithms have a few more differences:

1. The *gain ratio* estimate involves more calculations than the simple entropy one, which is employed by NewID. This however does not affect the order of complexity of the evaluation process.
2. The optional splitting of nominal attributes into *value groups* is also much more expensive than the binary-splitting used in NewID. Although a heuristic is used, which looks at pairs of attribute-values rather than exhaustively examining each combination of subsets of the attribute's value-set

$V \leftarrow \{v_1, ..., v_n\}$
$continue \leftarrow T$
$max\text{-}score \leftarrow gain\ ratio(V)$
While $|V| > 2$ & $continue$ **do**
$continue \leftarrow F$
For $i = 1$ **to** $|V|$ **do**
For $j = i + 1$ **to** $|V|$ **do**
$v_{ij} \leftarrow \{v_i, v_j\}$
$V' \leftarrow \{V - \{v_i\} - \{v_j\}\} + \{v_{ij}\}$
If $gain\ ratio(V') > max\text{-}score$
$max\text{-}score \leftarrow gain\ ratio(V')$
$V \leftarrow V'$
$continue \leftarrow T$
return V

where $\{v_1, ..., v_n\}$ is the value set of the attribute.

Figure 3.4: The value-grouping algorithm.

(figure 3.4), the worst-case computational complexity of the process is still of order[14]:

$$O((|V_a| - 1 + \sum_{i=2}^{|V_a|} \binom{i}{2}) \times |V_c||S|) =$$
$$O((|V_a| - 1 + \binom{|V_a| + 1}{3}) \times |V_c||S|) =$$
$$O((|V_a| - 1 + \frac{(|V_a| + 1)!}{(|V_a| - 2)!\ 3!}) \times |V_c||S|) \simeq$$
$$O(\frac{1}{6}|V_c||V_a|^3|S|)$$

where $|V_a|$ is the number of values of attribute a, $|V_c|$ the number of classes and $|S|$ the number of examples at the current node[15].

3. The final difference lies in the *maximum number of non-leaf nodes* that the final decision tree can have. In NewID the conclusion was that in the worst case the number of non-leaf nodes is $|E| - 1$ with $|E|/2$ examples per node. The same applies to C4.5, since the only requirement is the use of numeric attributes.

[14]Note that for each subset of values only the entropy of the new pair has to be calculated.
[15]Refer to the calculation of the complexity of the entropy measure in section 3.2.3.

Summarising the above, the complexity of the main element of C4.5, in the case where numeric attributes are use, is of similar order to NewID (eq. 3.5):

$$O(\ \frac{|A||E|^2}{2}\log(|E|/2)\ ) \tag{3.9}$$

In the case of nominal attributes, if value grouping is used, the complexity estimate is:

$$O(\ |A||V_{max}||V_c||E| \times (\frac{1}{6}|V_c||V_{max}|^3|E|)\ ) =$$
$$O(\ \frac{1}{6}|A||V_{max}|^4|V_c|^2|E|^2\ ) \tag{3.10}$$

On the other hand, if the standard splitting method is used, the estimate will be:

$$O(|A|^2|V_{max}||V_c||E|) \tag{3.11}$$

As for NewID, the worst estimate is the one using numeric attributes. However, using value-grouping is also very expensive and in the average case, it is expected to cost more than binary-splitting of numeric attributes.

3.3.4 Conclusion

C4.5 is a very interesting program, which incorporates much of the research done on decision-tree induction. What is especially interesting is the ability to transform decision-trees into meaningful and compact sets of rules, which is a very innovative idea.

However, the main structure of the program does not differ substantially from its predecessors and, as shown above (equation 3.9), its worst-case computational complexity is of the same order as for NewID. According to that, the complexity of the algorithm is of over-quadratic relation to the number of examples in the training set. This result has been based, however, on a very extreme and atypical situation.

Apart from that, there are a few features which are either not well founded or could be improved. One of them is the new evaluation criterion, which seems like a patch on the old one, rather than a proper solution. Another problem is the heavy calculations involved in the process of value-grouping, the worth of which is actually questionable. Post-grouping may prove to be a solution to this problem.

In addition to those problems, there are a few additions to C4.5, which would make it applicable to a larger range of problems. Namely the ability to handle numeric, as well as nominal classes and the incremental updating of decision-trees, are two of those desirable features.

3.4 PLS1

3.4.1 Introduction

PLS1 (Probabilistic Learning System 1) [Rendell, 1983b], [Rendell, 1987] is a conceptual clustering algorithm, developed at the same time as ID3. Although it uses a more statistical representation scheme than the other algorithms, examined in the project, its behaviour is very similar to that of ID3. Their major difference is in the way that they handle different types of features and this results to a different computational complexity.

3.4.2 Description

The representation used by PLS1 is that of a *hyperspace*, defined by the features used in the problem. Each feature thus corresponds to an axis of the space, along which its values vary. Based on that idea, the main process taking place is an *iterative dichotomisation* of the space into smaller *regions* (hyper-rectangles), by the use of hyperplanes, which are parallel to the axes and are selected according to a heuristic evaluation function, similar to ID3's entropy measure. Figure 3.5 describes the main PLS1 algorithm.

Examining this description, one can see the similarity of the learning process to the basic ID3 algorithm. Regions correspond to nodes in a binary decision tree and hyperplanes to the feature values.

The Evaluation Function

One important difference between PLS1 and ID3 is that the splitting process is taking place individually for each class, producing a separate set of regions. This results to a simpler evaluation function, which considers the training set as positive and negative examples of each class. The function is based on a **purity measure** for each region (its *utility*), provided by the proportion of positive examples covered by the region. Thus the utility of a region r_i is defined as:

$$u(r_i) = p_i/t_i \tag{3.12}$$

where p_i is the number of positive and t_i the total number of examples in r_i.

In order to evaluate a split, the *distance* of the two generated regions r_1 and r_2 is calculated. This measure is defined as:

$$d(r_1, r_2) = |\log(u(r_1)) - \log(u(r_2))| - c \times \log(e(r_1)e(r_2)) \tag{3.13}$$

Input: A set of examples E, a set of attributes A, a set of class values C, user-defined termination thresholds and confidence factor.
Output: One set of regions, for each class in C.

For each class c_i in C do:

Separate E to E_+ (positive) and E_- (negative) examples of c_i.

Define R_i as the region covering the whole of E.

While "non-terminal regions"[a] remain **do**:

Select one of those regions r.

Let d_{best} be 0.

Build the set H of hyperplanes, horizontal to the axes, which split r into subregions.

Initialise h_{best}.

For each hyperplane h in H **do**:

Split r into r_1 and r_2, using h.

Compute their dissimilarity $d(r_1, r_2)$.

If $d(r_1, r_2) > d_{best}$
Let d_{best} be $d(r_1, r_2)$.
Let h_{best} be h.

If $d_{best} > 0$
Replace r by the corresponding r_1, r_2 in R_i.

Return R_i.

[a] Regions which can be split further.

Figure 3.5: Design of the basic PLS1 algorithm

where $e(r_i)$ is an estimate of the error of the probability calculations. Usually, the main source of error is assumed to be the finite number t_i of examples covered by r_i and thus the error is estimated as $e(r_i) = 1 + 1/\sqrt{t_i}$. c is a user-defined confidence factor, which determines the significance of the split.

The distance function is an estimate of the dissimilarity between the examined regions. The higher the result the more dissimilar the regions are said to be and the better the split. This has a similar effect to the entropy measure of a single attribute-value, as used in ID3.

The Termination Criterion

The splitting process for a region terminates when none of the candidate splitting hyperplanes generates two subregions which are dissimilar enough (i.e. have a distance > 0). In this context, the confidence factor c, can be thought of as a stopping criterion, similar to the χ^2, used at some versions of ID3 [Quinlan, 1986b]. Apart from the confidence factor, however, there are two more parameters, which can cause the splitting process to terminate. These define the minimum number of positive and the total number of examples a region must contain, in order to be split further into subregions.

The effect of all those stopping parameters is that the algorithm can be set to exhibit different levels of noise resistance. The higher the values of the parameters, the less complete discrimination, the algorithm is aiming at.

Attributes

The initial version of the algorithm could handle only "*ordered*" (integer) attributes. This restriction is imposed by the hyperspace representation that is used. The values of nominal attributes cannot be ordered along an axis of the space and cannot be binary-split in a meaningful way. Later versions of PLS1 allow the use of nominal attributes, the values of which are then ordered, in terms of the proportion to which they appear in the training set. Thus, attribute values that appear in many examples are placed near each other, aiming at producing pure regions, by discriminating between these "important" values. This method of handling nominal attributes is obviously unsatisfactory, as important information contained by the attribute values is being lost. Moreover, PLS1 cannot handle continuous numeric features. The reason for that is not clear, as it seems possible that only a minor extension to the algorithm is sufficient for handling this type of attribute.

Another interesting feature of PLS1, is that it does not require an initial definition of the attributes and their domain. The reason for this is that all the

attributes are treated in a similar manner and thus their values can be collected by the training set itself. Only in the case where nominal attributes are used, a parameter has to be set to signal this, in order for the initial ordering of their values to take place.

One final point to be made about the way in which PLS1 handles the attributes, is a user-defined stepping value, which determines the values, examined as potential splitting points. In other words, the user defines how many values, along each axis will be examined and at what interval. The result of this is that not all the values of an integer attribute, appearing in the training set, can be used as splitting points (which is the case for NewID and C4.5). Thus, low values of the stepping factor can reduce classification accuracy significantly, while they can also increase the resistance of the algorithm to noise.

Other Programs

Concluding, one has to mention, that the clustering algorithm of PLS1 has been used in many programs, produced by Rendell, which perform more complicated induction tasks[16]:

- There is an *incremental* version of PLS1, which, in addition to the clustering algorithm, uses a normalisation and a regression procedure, producing non-linear approximations of the discrimination function.
- There is a *parallel* algorithm, called PLS2 [Rendell, 1983a], which uses a Genetic Algorithm to chose between a population of region sets.
- There is a program, called PLS0 [Rendell, 1985], which performs *feature construction* from elementary data, achieving thus more substantial data compression.

3.4.3 Analysis

Design

The core of the PLS1 algorithm is the iterative dichotomisation process which takes place for each class separately, generating a set of regions for it. Figure 3.6 describes this process.

[16] A comprehensive description of those programs is given in [Rendell, 1987].

Figure 3.6: The PLS1 clustering Procedure.

Worst-Case Computational Complexity Analysis

The process described in figure 3.6 is the main element and the main contributor to the complexity of the PLS1 algorithm. According to [Rendell *et al.*, 1989] this process has a linear relationship to the number of training instances and is of order $O(m|E||A|)$, where $|E|$ is the number of training examples, $|A|$ is the number of attributes and m the number of times the dichotomisation process takes place. The following analysis will examine the validity of this estimate.

Based on figure 3.6 the overall complexity of the algorithm is:

$$COMP(PLS1) = |C| \times (m \times COMP(Dichotomise(r)) + 2|E|) \qquad (3.14)$$

where $2|E|$ corresponds to the complexity of the initialisation process (defining R_i and separating E into E_+ and E_-).

The complexity of the dichotomisation process (figure 3.6), however, can be further decomposed as follows:

$$\begin{aligned} COMP(Dichotomise(r)) = COMP(TE(r)) + COMP(Build\text{-}planes(r) \quad + \\ |H| \times (\ COMP(Split(r, h_j)) + COMP(d(r_1, r_2))\ ) \end{aligned} \qquad (3.15)$$

Let us examine each component in turn:

1. **COMP(TE(r)):**
   As described in section 3.4.2, this process involves calculating the number of positive and the total number of examples in the region and comparing them to user-defined thresholds. Thus, if S is the subset of the training set covered by r, the complexity of this process is of order $O(|S|)$.

2. **COMP(Build-planes(r)):**
   This process generates the set of hyperplanes, which subdivide r into two smaller regions. The hyperplanes are generated at specific interval values, determined by the user-defined step factor. Assuming that, irrespective of the step factor, only one hyperplane is generated between two successive values of an attribute in S, in the worst case $|H| = |A| \times (|S| - 1)$ hyperplanes will be generated. Thus the complexity of this process is of order $O(|A||S|)$.

3. **COMP(Split(r,$\mathbf{h}_j$)):**
   This procedure compares the attribute value of each member of S with the value corresponding to the hyperplane h_j, forming thus two subsets of r. Thus the complexity of the process is $O(|S|)$.

4. **COMP(d($\mathbf{r}_1$,$\mathbf{r}_2$)):**
   The calculation of the distance between the two regions (equations 3.12,

3.13), includes the calculation of probabilities for the positive and total number of instances in each region, as well as the error estimate for each one. Assuming that the necessary totals are calculated at the previous stage, the contribution of this process to the overall complexity is insignificant.

Thus, according to equation 3.15 the complexity of the dichotomisation process is:

$$COMP(Dichotomise(r)) = O(|S|)+O(|A||S|)+O(|A||S|^2) \simeq O(|A||S|^2) \quad (3.16)$$

This estimate could be improved by ordering the instances in S, before examining the hyperplanes along each axis. The complexity of the process would then be $O(|A||S|\log|S|)$. Since this is not included in the description of the algorithm, however, one cannot assume that this is the case.

The final "piece" missing from the complexity analysis of the algorithm is the number of times (m) the dichotomisation procedure has to be used. Since each region-dichotomisation corresponds to a node of a binary decision tree, the worst case situation is similar to that described for C4.5 (section 3.3.3). Thus, in the worst case, $m = |E| - 1$ and $|S| = |E|/2$. Using this result, the overall worst-case complexity of the algorithm is:

$$\begin{aligned} COMP(PLS1) &= |C| \times (\ (|E|-1) \times O(\frac{|A||E|^2}{4}) + 2|E| \ ) = \\ & O(|C||A||E|^3) \end{aligned} \quad (3.17)$$

which is almost an order of magnitude higher than the estimate for the ID3-based programs (C4.5, equation 3.9 and NewID, equation 3.5), but becomes almost identical to it if it is reduced to $O(|C||A||E|^2\log|E|)$.

3.4.4 Conclusion

The result of the above analysis is significantly different from the complexity estimate, which appears in [Rendell *et al.*, 1989]. As for the other algorithms as well, the reason for this is that highly atypical situations are assumed, in order to obtain the extreme case. Moreover, the number of dichotomisations (m) that take place is shown to be bound by the number examples, something which has not been attempted in previous estimates.

In general the behaviour of the algorithm is very similar to that of the ID3-based ones, especially in the case where only integer attributes are used, resulting to a binary decision tree. The algorithm searches exhaustively the space of binary splits, applying the evaluation function at each candidate splitting point[17].

[17]This is true under the assumption that the step factor does not limit the number of hyperplanes.

Concluding, PLS1 is an interesting algorithm, which handles the problem of concept-learning in a similar manner to ID3, taking a more statistical approach. In doing that it seems to uncover the principles underlying the iterative dichotomisation process. The major limitation of the algorithm is the way in which it handles nominal attributes, which cannot be expressed sensibly, using the hyperspace representation. The solution to this problem seems to be the use of a different representation scheme for those attributes.

3.5 CN2

3.5.1 Introduction

CN2 [Clark and Niblett, 1989] [Clark, 1989] [Clark and Boswell, 1991] [Boswell, 1993a] is another algorithm, developed by the Turing Institute as part of the MLT project. It is an improved version of the basic AQ [Michalski, 1983] algorithm, incorporating ideas from ID3 [Quinlan, 1983a]. The program is written in C and has a similar interface to the one used by the NewID algorithm (section 3.2).

3.5.2 Description

CN2 is based on AQ and has inherited its basic structure. AQ uses a *beam search*, in order to induce a set of covers (see section 2.4.1), which can then be used as discrimination functions between a number of concepts/classes, figure 2.9, 2.10. Beam search combines a generalisation and a specialisation stage. Generalisation takes place in the main algorithm (figure 2.9), as one moves from a specific positive seed example to a general discrimination function, the cover for the class. However, this generalisation is achieved through an iterative specialisation process, taking place in the 'star' algorithm (figure 2.10). This process of specialising the complexes of a star, can be thought of as a set of m (maximum star size) *hill-climbing* searches, taking place in parallel.

The main problem with the AQ algorithm is that it is very dependent on individual training examples (i.e. the selected seeds [Clark and Niblett, 1989]). As a result it is difficult to adjust the algorithm in order to handle noise in the training set, in a similar manner to tree-pruning in the ID-family of algorithms (e.g. [Kononenko *et al.*, 1984]). Thus, [Clark and Niblett, 1989] suggested substantial changes to the algorithm, leading to the development of CN2, which is aimed to provide the flexible design of the AQ algorithm, combined with noise handling mechanisms.

Let E be a set of classified examples.
Let *SELECTORS* be the set of all possible selectors.

Procedure CN2(E)
  Let *RULE-LIST* be the empty list.
  Repeat until *BEST-CPX* is nil or E is empty:
    Let *BEST-CPX* be *Find-Best-Complex(E)*.
    If *BEST-CPX* is not nil,
    Then let E' be the examples covered by *BEST-CPX*.
      Remove E' from E.
      Let C be the most common class of examples in E'.
      Add the rule 'If *BEST-CPX* then the class is C'
      to the end of the *RULE-LIST*.
  Return *RULE-LIST*.

Procedure *Find-Best-Complex(E)*
  Let $STAR$ be the set containing the empty complex.
  Let *BEST-CPX* be nil.
  While $STAR$ is not empty,
   Specialise all complexes in $STAR$ as follows:
    Let *NEWSTAR* be the set:
      $\{x \wedge y | x \in STAR, y \in SELECTORS\}$.
    Remove all complexes in *NEWSTAR* that are either in $STAR$
    (i.e., the unspecialised ones) or null.
    For every complex C_i in *NEWSTAR*:
      If C_i is statistically significant and better than
      BEST-CPX by user-defined criteria when tested on E,
      Then replace the current value of *BEST-CPX* by C_i.
    Repeat until size of *NEWSTAR* $\leq$ user-defined maximum:
      Remove the worst complex from *NEWSTAR*.
    Let $STAR$ be *NEWSTAR*.
  Return *BEST-CPX*.

Figure 3.7: The CN2 algorithm

Thus, although it maintains the basic ideas, CN2 is different from AQ. Figure 3.7 describes the algorithm as it appears in [Clark and Niblett, 1989].

The first change is the form of the output. Instead of inducing a set of covers, CN2 generates an *IF ... THEN ... ELSEIF ... THEN ... ... ELSE ...* construct, which was named *rule list* [Rivest, 1987]. The main feature of this is that the semantics of each individual rule, depend on the previous ones. In other words, in order for a rule to fire, all the previous rules must have failed. This feature of the rule list excludes the possibility of a clash during the classification process, but makes it difficult to be interpreted by humans.

Rule lists were the result of a more important change in the algorithm: the evaluation of complexes is no longer restricted to the use of the set of negative examples of a specific class. The selection of a complex at each iteration of the *Find-Best-Complex(E)* procedure, is no longer dependent on seed positive and negative examples but on its classification performance on the whole training set, as estimated by a user-defined heuristic. This new feature makes the algorithm more tolerant to noise, since each *BEST-CPX* selected does not have to be complete and consistent with the training set, but just "predictive and reliable" [Clark and Niblett, 1989].

The issue of reliability introduces the third major change in the algorithm, the pruning (or 'search stopping') mechanism. This is achieved by the use of a heuristic, which measures the significance of each newly generated complex:

$$2\sum_{i=1}^{n} f_i \log(f_i/e_i) \quad (\textit{Likelihood Ratio Statistic}) \tag{3.18}$$

where n is the number of classes, f_i is the number of examples belonging to the i^{th} class, in the set of examples covered by the complex and e_i is the total number of examples belonging to the class, scaled to the coverage of the complex (i.e. $\sum_{i=1}^{n} f_i$).
The likelihood ratio [Kalbfleish, 1979], measures the significance, in terms of classification, of the complex, based on the distance between the resulting class distribution and the default one. In order for the rule generation process to continue the result of this measure has to be above a user-defined threshold, similar to the cut-off threshold in some ID-based systems [Kononenko *et al.*, 1984].

The final important difference between AQ and CN2 is the criteria used in the evaluation of complexes. The most common criterion, used in AQ-based systems is the coverage of the complex (i.e. how many examples it covers). CN2 however uses entropy to measure the quality of the complex:

$$-\sum_{i=1}^{n} p_i \log(p_i) \quad (\textit{Entropy}) \tag{3.19}$$

where n is the number of classes and p_i is the probability of the i^{th} class, in the set of examples covered by the complex.

The main argument for using entropy (applying similarly to ID-based systems) is that it gives an informative estimate of the classification accuracy of the complex, based on its overall performance, rather than its performance on a single class. Entropy however has not proved the ideal choice and it was replaced by the '*Laplacian error estimate* [Clark and Boswell, 1991], described later in this section.

Based on some experiments done by the developers of CN2
[Clark and Niblett, 1989], the algorithm seemed to provide similar classification accuracy to that of a simple implementation of AQ and an ID-based algorithm (ASSISTANT [Kononenko *et al.*, 1984]) in relatively "noiseless" domains. The advantages of the algorithm, however, become apparent with the introduction of noise. In those cases CN2 did substantially better than the AQ algorithm.

These experiments, however, showed that the algorithm could be improved further. The main problem was that in some cases insignificant rules would gain a high enough score to pass the significance threshold. The reason for that is that entropy does not take into account the overall coverage of the rules, favouring thus rules which achieve a high accuracy on a small number of examples. The significance test would eliminate some of those rules, but those which just "survive", would be preferred over more significant rules which are less accurate.

The solution to this problem [Clark and Boswell, 1991] was to use the 'Laplacian error estimate' to evaluate the quality of the complexes:

$$1 - \frac{(f_i + 1)}{(\sum_{j=1}^{n} f_i + n)} \qquad (Laplace\ Error\ Estimate) \tag{3.20}$$

where f_i and n are defined as in equation 3.18.
Using this function, the quality of a complex is estimated under the assumption that the complex predicts the class with the highest frequency f_i in the set of examples that it covers. The complex with the lowest error estimate is then said to be the best one.

The main advantage of this evaluation function over the entropy one is the it takes into account the total coverage of the complex[18], rather than its performance on individual classes. In other words, it does not pay attention to the discrimination power of the complex, concentrating on its overall significance. In this respect, the significance test is incorporated in the evaluation function. The likelihood ratio statistic was maintained, however, acting now as a pure search stopping criterion[19].

[18]Note the importance of the constant elements of the function which provide an indication of the scale at which the complex covers the training set. The choice of these constants, however, has not been justified in any of the cited papers.

[19]Alternatively, the result of the evaluation function could have been used as a termination criterion.

In order to illustrate the behaviour of the two heuristics, assume the following simplified cases:

CASE 1: [Clark and Boswell, 1991]
Assume a binary classification task and the complexes A[1000,1] and B[5,0], covering the former 1000 examples of the first and 1 example of the second class and the later 5 and 0 respectively. In that case the entropy measure would select complex B, because it achieves perfect discrimination, ignoring the fact that A covers a lot more examples and is thus more reliable. The new evaluation function resolves that problem, providing a much better score for complex A than for B.

CASE 2: Assume a similar situation to the previous case, but this time complexes A[1000,100] and B[100,10]. Entropy would give the same score to both those complexes, while the Laplace error estimate would favour the former.

CASE 3: Assume a problem with three classes and complexes A[1000,1000,0] and B[1000,500,500]. This is a situation where the new evaluation function cannot choose between the two complexes, while entropy would chose A, because the examples are concentrated in two of the three classes.

Empirical results [Clark and Boswell, 1991] have shown that the Laplacian Error Estimate results to substantially higher accuracy than the Entropy evaluation function, especially in noisy domains.

Another addition that was done to the CN2 algorithm, was the ability to generate unordered rules [Clark and Boswell, 1991], similar to the covers produced by the basic AQ algorithm. This modification resulted to an algorithm very similar to AQ (i.e. carrying out a beam search individually for each class), with two major differences, inherited from the "ordered" version of the algorithm:

- There are no seed positive and negative examples selected. The algorithm searches for "reliable and significant" complexes, rather than "complete and consistent" ones.
- The 'Laplace Error Estimate' heuristic is used for the evaluation of complexes.

The main advantage of producing unordered rule lists is that they are more comprehensible than the ordered ones. However, contradictions between the rules are now possible. This problem was solved, by tagging each rule with its correspondent class-frequency distribution in the training set. Using that, an unseen

example which causes a clash will be classified according to the sum of the distributions of the rules that fire. For example, assume the following rules:

$$IF\ cond_1\ THEN\ C_1\ [10,0]\ \ (Rule1)$$
$$IF\ cond_2\ THEN\ C_2\ [5,10]\ \ (Rule2)$$

where $[10,0]$ is the class-frequency distribution for rule 1. Assume also an example which satisfies both $cond_1$ and $cond_2$. In that case, the total frequency distribution would be $[15,10]$ and the example will be classified as C_1. Experimental results [Clark and Boswell, 1991] showed that unordered rules provide also slightly higher prediction accuracy, than ordered ones, which may be the result of tolerance to noise, due to the clash resolving mechanism. An additional feature of unordered rule lists is that they tend to be much larger than the ordered ones.

Finally, in terms of its interface, CN2 is very similar to NewID. The only differences are the following:

1. Only *nominal class* values can be used.
2. An example which includes a '*don't care*' value for some attribute is not duplicated as in NewID.
3. In the case of an unordered rule list, single rule can be evaluated as well as the whole list.
4. *Nominal attributes* cannot be handled as binary.

Another interesting feature of the unordered list produced by CN2 is that the class predicted by a rule does not have to be the one with the highest value in the frequency distribution. This is usually the case for classes which are not represented with many examples in the training test.

3.5.3 Analysis

Design

As described in the previous section, there are two versions of CN2, producing ordered and unordered rules. Although, they have the same underlying structure, the design of the controlling element is different for each of them (figures 3.8, 3.9).

The search procedure, however is the same for both versions, with the exception of the calculation of the significance and evaluation heuristics (figure 3.10). These are class specific for the unordered version of CN2. Moreover, the 'Laplace Error Estimate' is used, instead of the 'Entropy'.

Figure 3.8: The Control Module (Ordered).

Figure 3.9: The Control Module (Unordered).

Figure 3.10: Find Best Complex (CN2).

Worst-Case Computational Complexity Analysis

According to [Clark and Niblett, 1989], the computational complexity of "ordered" CN2 is $O(|A|MAXSTAR(|E| + \log(|A|MAXSTAR)))$, where A, E correspond to the attribute and the training set and $MAXSTAR$ is the maximum size of a star (user-definable). This estimate however corresponds only to the search element of the algorithm and is done under the assumption of binary class and attributes, which affects significantly the calculation. The analysis done here examines the algorithms in more detail, aiming at a **worst-case estimate**.

According to figure 3.8 the complexity of the "ordered" version of CN2 can be broken down as follows:

$$\begin{aligned} &COMP(\textit{ORD-CN2}(E)) = \\ &\quad n_1 \times (\ COMP(\textit{Find-Best-Cpx}(E)) + \\ &\quad COMP(\textit{Covered}(\textit{BEST-CPX}, E)) + \\ &\quad COMP(E - E') + COMP(\textit{Most-Common}(E'))\ ) \end{aligned} \tag{3.21}$$

where n_1 is the number of iterations before the final rule list is constructed. Similarly for the "unordered" (fig. 3.9) version of the algorithm:

$$\begin{aligned} &COMP(\textit{UNORD-CN2}(E)) = \\ &\quad |C| \times n_2 \times (\ COMP(\textit{Find-Best-Cpx}(E, C_i)) + \\ &\quad COMP(\textit{Covered}(\textit{BEST-CPX}, E, C_i)) + COMP(E - E')\ ) \end{aligned} \tag{3.22}$$

where n_2 is similar to n_1 but for an individual class.

Let us examine each individual component:

1. **COMP(Find-Best-Cpx(E[,C_i])):**
   This can be decomposed further according to figure 3.10:

$$\begin{aligned} &COMP(\textit{Find-Best-Cpx}(E[, C_i])) = \\ &\quad m \ \times \ (\ COMP(\textit{Multiply}(STAR, SELECTORS)) + \\ &\qquad COMP(\textit{Reduce}(NEWSTAR, STAR)) + \\ &\qquad |NEW| \times (\ COMP(\textit{Significant}(CPX_j[, C_i])) + \\ &\qquad\qquad COMP(\textit{Evaluate}(CPX_j)[, C_i]))\ ) + \\ &\qquad COMP(Trim(NEW)\ ) \end{aligned} \tag{3.23}$$

   where m stands for the number of iterations to be done during the process.

   The complexity of each of those components is:

   (a) **COMP(Multiply(STAR,SELECTORS)):**
   The complexity of this process is clearly $O(|STAR||SELECTORS|)$,

since each possible selector is added to each complex of the star. $|STAR| = MAXSTAR$, while in the worst case, where all the attributes are numeric and have different values for each example, $|SELECTORS| = |A| \times |E|/2$, assuming that their values are represented in ranges. Thus, the total complexity of this process is: $O(MAXSTAR \times |A| \times |E|/2)$. In the equivalent worst-case for nominal attributes, the complexity of the process is $O(MAXSTAR|A||V_{max}|)$, where V_{max} is the largest value-set. The later case is clearly less expensive than the former[20].

(b) **COMP(Reduce(NEWSTAR,STAR)):**
For this process one needs to compare each of the newly added selectors with the ones already existing in each complex of STAR and discard the new complex if it is subsumed by the old, or if it contains a contradiction. This will usually be done in parallel to the multiplication and costs $(MAXSTAR \times |A| \times |E|/2) \times$ *CPX-LENGTH*, where *CPX-LENGTH* is the length of the old complex, which in the worst case could contain an average of $|A|/2$ selectors. One could argue at this point that *NEWSTAR* should be reduced further, by checking whether any of the newly generated complexes is subsumed by other new ones. Although this case is not mentioned in the literature about the algorithm, it seems unlikely to take place at this stage, because it would result in a combinatorial explosion. It can be done more efficiently after the trimming of the new star. Thus the overall complexity of this step is: $O(MAXSTAR \times |A|^2|E|/4)$. Similarly for nominal attributes the worst-case is $O(MAXSTAR|A|^2|V_{max}|/2)$.

(c) $|\boldsymbol{NEW}| \times \boldsymbol{COMP}(\mathit{Significant}(\boldsymbol{CPX}_j[,\boldsymbol{C}_i]))$:
The main requirement, in terms of computations, at this stage is the calculation of the class frequencies (equation 3.18) for each complex (and their summation in the "ordered" version). Since we have accepted $|A|/2$ as the maximum number of selectors per complex, in the worst case, this process takes time $O(|A|/2 \times |E|)$. The maximum size of the star ($|NEW|$), after the reduction stage, is $|NEW| = |A| \times |E|/2$, because that is the maximum number of distinct complexes that can be produced. Multiplying that to the complexity of the significance calculation, gives an overall complexity of $O(\frac{|A|^2|E|^2}{4})$ for the "unordered" and $O(|C| \times \frac{|A|^2|E|^2}{4})$ for the "ordered" one. The corresponding complexities for nominal attributes are $O(|A|^2|V_{max}||E|/2)$ and $O(|C||A|^2|V_{max}||E|/2)$.

(d) **COMP(Evaluate(CPX$_j$[,C$_i$]))):**
Both the evaluation functions used, involve the calculation of class

[20]The assumption that the number of distinct values for a nominal attribute is $\leq |E|/2$ is reasonable for large training sets where the complexity of the algorithm becomes important.

frequencies and probabilities (equations 3.19, 3.20, with the difference that entropy includes a summation of the results for each different class[21] Thus, the complexity for each version is exactly the same as in the previous step.

(e) **COMP(Trim(NEW)):**
This final step requires the complexes to be sorted according to their scores and only the MAXSTAR best ones to be kept. The cost of sorting, assuming an efficient sorting routine, is $O(|A||E|/2 \times log(|A||E|/2))$.

(f) *m:* Since the complexes in STAR get more specialised each time, m is upper bound by the maximum size of a complex. Thus the maximum number of iterations that can take place is $|A|$.

Thus, according to equation 3.24, the overall complexity of the search process for the "unordered" version is:

$$\begin{aligned} &COMP(Find-Best-Cpx(E)) = \\ &|A| \times \\ &\quad (O(MAXSTAR \times |A|^2 \times |E|/2) + \\ &\quad\quad O(\frac{|A|^2|E|^2}{4}) + \\ &\quad\quad O(\times |A| \times |E|/2 \times \log(|A| \times |E|/2))) \simeq \\ &|A|^3|E|^2 \end{aligned} \tag{3.24}$$

and for the "ordered" one:

$$COMP(\textit{Find-Best-Cpx}(E, C_i)) = |C||A|^3|E|^2 \tag{3.25}$$

Similarly, the worst-case complexity for nominal attributes is:

$$O(|A|^3|V_{max}||E|)$$

for the "unordered and

$$O(|C||A|^3|V_{max}||E|)$$

for the ordered version of the algorithm.

One has to note at this point that due to the $MAXSTAR$ restriction not all of the $2^{(|A||E|/2)} - 1$ potential complexes are examined. MAXSTAR, thus determines the *exhaustiveness* of the search.

[21] Entropy calculations are significantly more expensive in terms of machine time due to the calculation of logarithms.

2. **COMP(Covered(BEST-CPX,E[,C_i])):**
   This process[22] requires that each selector of the complex is compared to the corresponding selector of each example. Assuming that the selectors are ordered according to the attributes and attribute values, this process takes time $O(|A||E|)$.

3. ***COMP*($E-E'$)**: This step does not add to the computational complexity of the process, because it can be carried out during the previous step.

4. **COMP(Most-Common(E')):**
   In the worst case $E' = E$ and the computational complexity of this step is $O(|C||E|)$.

5. **n_1**:
   In the worst-case of extreme overfitting to the data, $|E|$ complexes will be generated. Thus $n_1 = |E|$[23]. If nominal attributes are used, the corresponding maximum value of the parameter is $n_1 = |V_{max}|^{|A|}$.

6. **n_2**:
   In average, the number of examples belonging to each class is $|E|/|C|$. In the worst case of overfitting, described above, this number corresponds also to the number of complexes generated at each iteration. Thus, in a similar way, we assume that $n_2 = |E|/|C|$. For nominal attributes $n_2 = n_1$.

Based on the above analysis, the worst-case complexity of the "unordered" version of the algorithm is:

$$\begin{aligned} &COMP(\textit{UNORD-CN2}(E)) = \\ &\quad |C| \times (\ |A|^3|E|^3 + |A||E|^2 \ ) = \\ &\quad O(|C||A|^3|E|^3) \end{aligned} \tag{3.26}$$

while for the "ordered" version the complexity is:

$$\begin{aligned} &COMP(\textit{ORD-CN2}(E)) = \\ &\quad |C||A|^3|E|^3 + |A||E|^2 + |C||E| = \\ &\quad O(|C||A|^3|E|^3) \end{aligned} \tag{3.27}$$

In other words, both versions are of the same order of complexity.

Similarly for nominal attributes the complexity of the two versions of the algorithm is:

$$O(|C||A|^3|V_{max}|^{|A|+1}|E|) \tag{3.28}$$

[22]Note that this process is implied also in the calculation of the class frequencies.

[23]This is similar to the situation where 1 large complex with $|E|$ selectors is produced.

3.5.4 Conclusion

The results of the worst-case analysis are very different from the previously reported estimate of the complexity of CN2 [Clark and Niblett, 1989]. The reason for that is the extremity of the worst-case assumptions, which are very atypical of the situations in which the algorithm is expected to be used. However, these results could be used as means of comparing the expected performance of the examined algorithms. In that respect, CN2 seems to be doing better than AQ15 and worse than the decision-tree based algorithms both with respect to the size of the training set and the size of the search space.

The factors which determine the worst-case scenario for CN2 are the following:

1. The maximum **number of complexes** that can be produced.
2. The number of **iterations of the complex-generating procedure**, needed to produce the maximum number of complexes.
3. The **total number of selectors** that exist.
4. The **maximum number of selectors each complex** can have.

The estimation of the first two, of the above listed, parameters is decisive for the complexity of the algorithm. Previous work on the topic has assumed that these parameters are only dependent on the nature of the problem, avoiding the estimation of upper bounds that was attempted in the above analysis.

Concluding, one could argue that CN2 provides a substantial improvement to the basic AQ algorithm. It removes the dependence of the algorithm from specific examples, providing unconstrained specialisation of complexes (i.e. the search is not constrained to consistent complexes). This results to better performance in the presence of noise, further enhanced by the use of the significance criterion, which provides a "search-stopping" mechanism.

3.6 AQ15

3.6.1 Introduction

As its name reveals, AQ15 [Michalski *et al.*, 1986], [Hong *et al.*, 1986], is based on the AQ algorithm (section 2.4.1). It is one of the latest versions of the algorithm, incorporating a number of new features (e.g. incremental and constructive induction), although its basic inductive process is very similar to its antecedents. The program has been implemented in Pascal.

3.6.2 Description

Due to the similarity of AQ15 to the basic AQ algorithm, this section will focus on its new features.

Representation scheme

The representation scheme used in the AQ-family of learning algorithms is the '*Annotated Predicate Calculus (APC)*' [Michalski, 1983]. This scheme is more expressive than the schemes used in other algorithms and allows the construction of decision-rules, which are comprehensive to humans. AQ15 introduces an extension to the basic scheme, as a by-product of the constructive induction process. Selectors (or *extended selectors* as they are now called) can now refer to a combination of attributes, rather than a single one. If one assumes for example that attributes a_1 and a_2 are numeric the following is a valid selector:

$$5 \times a_1 + a_2 = 30..40$$

Attributes

A further distinguishing feature of the AQ series of algorithms is their ability to handle tree-structured attributes and their inability to handle continuous ones. Thus, AQ15 accepts *nominal*, *linear* (integers), "*interval*" (ordinal), *cyclic*[24] (integers in circular order) and *structured* attributes. A further addition to AQ15 is the use of structured classes. A class can be subdivided into *children* classes, which are associated with a subset of the training examples belonging to their parent class. This subdivision could potentially result to more accurate and meaningful classification within the subclasses.

Examples

The training examples for AQ15 have a similar format to the other algorithms, examined in the project. However, apart from being assigned a single value, an attribute can be associated with a list or a range of its legal values. Undefined attribute values are also allowed but they are all handled as "*don't care*" ones (ref. section 3.2.2).

[24]These are also a by-product of constructive induction. An example would be an attribute which is defined as the modulo of an integer attribute with some arithmetic constant.

Incremental Learning

In addition to the training examples, AQ15 accepts decision rules as input. These can either be provided by the user as background knowledge or by a previous run of the algorithm. In the case where such rules are provided, they are being refined according to the new training examples. The result is a new set of rules which cover in a "*consistent* and *complete*" way the input rules and the new examples. This feature makes the system usable in situations where continuous update of the rules, in the light of new information, is required.

Constructive Learning

Apart from decision rules, AQ15 accepts an additional form of background information. It can be given rules which associate attributes in some way and can be used to generate new attributes. There are two types of those rules: *arithmetic* (a-rules) and *logical* (l-rules). The former are applied to numeric and the later to all types of attributes. The following are examples of such rules (assume that a, b, c are the three sides of a triangle):
A-rule:

$$\text{area} := 0.5 \times \text{a} \times \text{b} \textbf{ if } \text{type=right-angle}$$

L-rule:

$$\text{area} := \text{large} \textbf{ if } \text{a} \ \& \ \text{b} \ \& \ \text{c} > 10$$

The algorithm uses those rules constructing new features, which are expected to be more powerful in terms of prediction.

Noise Handling

AQ15 lacks the search-stopping criterion used in CN2 (ref. 3.5.2), but it incorporates pre- and post-processing facilities, in order to eliminate the effects of noise. Pre-processing involves the resolution of clashes between training examples before the induction process commences. Clashing examples can be considered positive or negative for their class or they can be ignored altogether. The way in which they are to be handled is specified by the user. Post-processing involves the simplification of the generated decision-rules, by eliminating insignificant complexes[25] (ref. [Michalski *et al.*, 1986]). The significance of complexes is determined by the number of examples that they cover.

[25]This process differs from the tree-pruning used in NewID and C4.5, which would correspond to the removal of selectors from within the complexes.

Lexicographic Functionals (LEFs)

The learning process of the AQ algorithm involves a search for complexes which do well in discriminating between positive and negative examples of a class. During this search, complexes are evaluated based on some user-defined criteria (LEFs). The most commonly used LEF is the coverage of the complex. AQ15 introduces some new LEFs, which can be used for complex-evaluation. Examples of those are the *ratio between positive and negative events* covered by the complex and the *cost of variables used*. The existence of more LEFs allow the user to tailor the behaviour of the system to his/her specific problem.

Evaluation of Covers

AQ15 allows for two modes of evaluation of the generated decision-rules: *strict* and *analogical*. In the former mode an instance must be covered completely by the complexes of a rule, in order to be used for its evaluation. In the analogical mode, however, the *distance* between the rule and the instance is calculated. The distance of a rule from an instance is defined as the probabilistic sum of the distance of each of the rule's complexes from the instance, which in turn is defined in terms of the number of selectors of the complex, which cover the corresponding selectors of the instance (ref. [Michalski *et al.*, 1986]). Having calculated the distance of all the rules, the one with the highest score is being used for the classification of the instance.

Usage

Finally, there are two points to be made about the way in which the program is being used:

- The program can be run in one of the three following modes:

  ic (intersecting covers) In this mode covers of different classes are allowed to intersect.

  dc (disjoint covers) In this mode the algorithm uses as negative examples for a class the covers generated for the preceding classes.

  vl (variable-valued logic) This mode produces an ordered list of rules[26], by ignoring the examples of preceding classes.

- All input to the program is done using specially named relational tables. This enables the use of a spreadsheet as a front-end to the system.

[26]This is different from the rule list generated by the ordered version of CN2, because only one rule is generated for each class.

3.6.3 Analysis

Most of the new features introduced in AQ15 are aimed at extending the usability of the system, but do not affect significantly the basic induction structure. Thus, if one ignores the pre- and post-processing facilities which have been added[27], the core element is similar to the basic version of the algorithm (figures 2.9, 2.10). Thus, this section will focus on a few differences which exist.

Design

There are two new features of AQ15 which could have an effect on the design of the control module:

- The use of decision-rules as input to the induction process.
- The use of structured classes.

The former involves some extra pre-processing for each individual class. The input rules are being translated into positive and negative examples for the class. The later change is handled by considering each sub-class individually, inheriting the negative examples of its parent and adding to them the examples of its brother-classes.

Thus, the design of the control module is not affected significantly and it is similar to the one used for the unordered version of CN2 (figure 3.9). Their major difference, however, is that AQ15 bases its search on individual positive seed examples. Figure 3.11 describes the control element of AQ15.

As seen, when examining CN2 (figure 3.7), the search (for best complex) procedure of the algorithm is also different from the one in the basic AQ algorithm. The difference lies in that AQ (and AQ15) considers only those complexes which do not cover the negative examples for the class, rather than evaluating all possible complexes. Moreover, AQ15 differs from the basic AQ algorithm in that it orders the set of negative examples, rather than selecting one in random each time. Figure 3.12 describes the process.

Worst-Case Computational Complexity Analysis

Despite their similar structure, the complexity of CN2 and AQ15 differ significantly, mainly due to the inability of AQ15 to handle numeric attributes of unlimited range, which increase substantially the worst-case complexity of the CN2 algorithm.

[27] A thorough description of those is given in [Hong *et al.*, 1986].

Figure 3.11: The Control Module.

Figure 3.12: The Search Procedure.

The complexity of the AQ15 algorithm can be calculated as follows:

$$\begin{aligned} &COMP(AQ15(E)) = \\ &\quad |C| \times n \times (\; COMP(\textit{Find-Best-Cpx}(E_-, e^+)) + \\ &\quad COMP(Covered(BEST-CPX, E_+)) + COMP(E_+ - E'_+) \;) \end{aligned} \tag{3.29}$$

where n is the number of iterations before the final rule is constructed.
Although this calculation looks very similar to the one done for CN2 (equation 3.22), the worst-case complexity of its components is significantly different.

Let us examine each component in turn:

1. $\boldsymbol{COMP}(\textit{Find-Best-Cpx}(\boldsymbol{E_-, e^+}))$:
   This can be decomposed further according to figure 3.12:

$$\begin{aligned} &COMP(\textit{Find-Best-Cpx}(E_-, e^+)) = \\ &\quad COMP(\quad \textit{Dist-Sort}(E_-, e^+) \quad) + \\ &\quad |E_-| \times \quad (\; COMP(Covers(STAR, e_j^-)) + \\ &\qquad\qquad COMP(\textit{Cover-against}(e^+, e_j^-)) + \\ &\qquad\qquad COMP(Multiply(STAR, MGC)) + \\ &\qquad\qquad COMP(\textit{Value-Order}(STAR, LEF)) + \\ &\qquad\qquad COMP(Trim(STAR)) \;) \end{aligned} \tag{3.30}$$

   The complexity of each of those components is:

   (a) **COMP(Dist-Sort(E_-, e^+)):**
   This process involves the calculation of the distance between each negative example in E_- and the positive seed e^+ and the sorting of the negative examples in descending order of distance. The distance between two complexes (the examples) is defined as the number of disjoint selectors that they contain. Thus the complexity of the distance calculation is of order $O(|E_-||A|)$, while an efficient sorting routine will take time $O(|E_-|\log|E_-|)$. Assuming that $|A| < \log|E_-|$, for large data sets, the overall complexity of the process is $O(|E_-|\log|E_-|)$.

   (b) **COMP(Covers(STAR,e_j^-)):**
   During this process, each complex in $STAR$ is examined as to whether it covers e_j^-. Thus the complexity of the process is of order $O(MAXSTAR|A|)$.

   (c) **COMP(Cover-against(e^+, e_j^-)):**
   At this stage a disjunction of selectors is produced, which provide the *Most General Cover*[28] of e^+ that excludes e_j^-. The generalisation

[28] A cover of e^+, containing single-selector complexes, that excludes e_j^-.

of each selector is based on the domain definition of the attribute involved, which specifies the range of values the attribute can take. Since AQ15 cannot handle numeric attributes without an upper bound, the value-set for each attribute is finite. Thus, if V_{max} is the largest defined value set, the complexity of this process is of order $O(|A||V_{max}|)$.

(d) **COMP(Multiply(STAR,MGC)):**
During this process, the conjunction of each single-selector complex of MGC and each complex in $STAR$ is produced. This process is of order $O(|MGC||STAR|) = O(MAXSTAR|A||V_{max}|)$ and its output is a star containing in the worst case $MAXSTAR|A||V_{max}|$ complexes.

(e) **COMP(Value-Order(STAR,LEF)):**
The complexity of this process depends on the evaluation function (LEF) which is selected. In the worst-case, the set of examples covered by each complex will have to be found and the complexity of the process will be $O(|STAR||E_-||A|) = O(MAXSTAR|E_-||A|^2|V_{max}|)$. This is the most expensive process carried out during the search for the best complex and it would have been more expensive if infinite value sets were allowed.

(f) **COMP(Trim(STAR)):**
At this stage complexes which are covered by a disjunction of the other complexes in the cover are eliminated and then the $MAXSTAR$ best ones are kept. The former process is quite expensive, requiring, in the worst-case, time $O(|STAR|^2|A|) = O(MAXSTAR^2|A|^3|V_{max}|^2)$.

Thus the overall complexity, according to equation 3.29 of the search process is:

$$\begin{aligned} COMP(\textit{Find-Best-Cpx}(E_-, e^+)) &= |E_-|\log|E_-| + \\ |E_-| \times \; & (MAXSTAR|A| + |A||V_{max}| + \\ & MAXSTAR|A||V_{max}| + \\ & MAXSTAR|E_-||A|^2|V_{max}| + \\ & MAXSTAR^2|A|^3|V_{max}|^2) = \\ & O(MAXSTAR|A|^2|V_{max}||E_-|^2) \end{aligned} \tag{3.31}$$

2. $\boldsymbol{COMP(Covered(BEST-CPX, E_+))}$:
During this process each positive example is examined as to whether it is covered by $BEST\text{-}CPX$. As mentioned above, this process costs time $O(|E_+||A||V_{max}|)$.

3. $\boldsymbol{COMP(E_+ - E'_+)}$:
This process does not add to the complexity of the algorithm because it can be done at the previous stage.

n : Clearly, in the worst case, only one positive example will be subtracted from the set of positive examples at each iteration. Then $n = |E_+|$.

Based on the above analysis and equation 3.29, the complexity of the AQ15 algorithm is[29]:

$$|C| \times |E_+| \times (MAXSTAR|A|^2|V_{max}||E_-|^2)$$

For a large number of classes one can assume that $|E_-| \simeq |E|$. Thus, if no children-classes exist[30] the overall complexity of the algorithm is[31]:

$$O(MAXSTAR|A|^2|V_{max}||E|^3) \tag{3.32}$$

According to this estimate, the complexity of AQ15 has a cubic relation to the number of training examples. This is one order of magnitude less than the estimate for CN2 and that is the result of the upper bound on the size of the value sets.

3.6.4 Conclusion

By setting a restriction to the number of values each attribute can take one would expect the complexity of the AQ15 to be significantly smaller than the other examined algorithms. However, this is not the case and the algorithm is found to have a high order of complexity, not only relative to the number of training examples, but also to other parameters. Thus, for example, if one was to increase the maximum size of the value set at the same time as increasing the number of examples, the effect could be dramatic.

Apart from the effect that it has on the complexity of the algorithm, having to specify the range of values for numeric attributes, reduces the applicability of the system to many real-world problems. This, however, may be offset by the plethora of other useful facilities provided by AQ15. Most importantly, the use of incremental learning and the ability to construct new features, based on background knowledge can be advantageous to a number of applications.

Concluding, AQ15 is an interesting learning system, with numerous useful facilities. The complexity of the underlying search processes, however, is quite large, increasing the computational complexity of the algorithm. In order to solve

[29]Only the complexity of the search process is included, because it is the one which determines the complexity of the whole process.

[30]If this is not the case the complexity estimate must be multiplied by the proportion of sub-classes to the classes at the highest level of the hierarchy.

[31]Notice that due to the upper bound to the number of values of integer attributes, the complexity of the algorithm is the same for both nominal and numeric attributes.

this problem, the search space is being limited, with the use of upper bounds to the size of the value sets. Thus, an improvement of the search method could make possible the removal of the restriction in the search space. This is succeeded to a certain extend in the CN2 algorithm (section 3.5).

3.7 Summary

This chapter has provided a description of the algorithms that were used in the project. In addition to that, a detailed worst-case computational complexity analysis of the algorithms was carried out, focusing on those design decisions, which affect mostly their performance.

Algorithm	*Heuristic*	*Purpose*	*Problems*
NewID	Simple Entropy (eq. 3.1)	Evaluation	Favours attributes with many values
	Sum of Class Variances (eq. 3.3)	Evaluation	
	"Binary nominal"	Splitting	
C4.5	Mutual Information (entropy gain) (eq. 3.2)	Evaluation	Unbalanced Splits
	Value-grouping	Splitting	
PLS1	Region Distance (eq. 3.13)	Evaluation and Growth-stopping	
CN2	Likelihood Ratio Statistic (eq. 3.18)	Growth-stopping	
	Entropy (eq. 3.19)	Evaluation	Rules with small coverage
	Laplace Error Estimate (eq. 3.20)	Evaluation	
AQ15	LEFs (variety of heuristics)	Evaluation	

Table 3.1: Heuristics used in each algorithm.

During the description of the algorithms, special attention was paid to the heuristics used and the types of attributes that each algorithm can handle. This is due to the importance of these two features to the computational complexity of the algorithms. Table 3.1 and 3.2 summarise these information, provided throughout the chapter.

In addition to these, the following are important features of the algorithms:

Algorithms	*Attribute Types*				
	nominal	ordinal	integer	continuous	structured
NewID	√	√	√	√	—
C4.5	√	—	√	√	—
PLS1	—	—	√	—	—
CN2	√	√	√	√	—
AQ15	√	√	(bound)	—	√

Table 3.2: Attribute types that each algorithm can handle.

- **Coverage**
  Irrespective of the representation used, the end-result of all the examined algorithms can be seen as a set of orthogonal hyperrectangles in the hyperspace defined by the feature set. This is a feature dominating the design of the algorithms, which incrementally build this set of rectangles.

- **Expressiveness of Representation**
  Although they all generate a set of hyperrectangles, the expressiveness of the representation scheme of the examined algorithms varies substantially. For example, the scheme used by PLS1 is very restrictive allowing only for ordered attributes, while AQ15 can handle structured concepts and attributes and generate new attributes, combining old ones in a logical or an arithmetical way.

- **Search Type**
  The "*specialisation*" algorithms (NewID, C4.5, PLS1) use *hill-climbing*, while the "*generalisation*" ones use *beam search*.

- **Incremental and Constructive Learning**
  Only AQ15 is able to perform these types of learning.

In terms of their computational complexity, the algorithms which use *decision-trees* were found to be over-quadratic, while the other three are of even higher order. These results are very different from previously reported work, where similar algorithms to the ones examined here are shown to be of a near-linear worst-case computational complexity (e.g. [Rendell *et al.*, 1989], [O'Rorke, 1982], etc.). The main reason for this is that researchers, who have dealt with the problem in the past, assumed that only nominal features would be used by the algorithms. An assumption that was valid for early versions of the algorithms. The most recent versions, however, incorporate mechanisms for dealing with numeric features, which cause the search space to become of an infinite size and push the upper bounds, considered in worst-case analysis to the size of the training set.

Algorithms	*Attribute Types*			
	nominal	special	integer	continuous
NewID	$O(ca^2ve)$	$O(\frac{3}{16}cav^3e^2)$	$O(\frac{1}{2}ae^2\log(e/2))$	$O(\frac{1}{2}ae^2\log(e/2))$
C4.5	$O(ca^2ve)$	$O(\frac{1}{6}c^2av^4e^2)$	$O(\frac{1}{2}ae^2\log(e/2))$	$O(\frac{1}{2}ae^2\log(e/2))$
PLS1	—	—	$O(cae^3)$	—
CN2	$O(ca^3v^{a+1}e)$	—	$O(ca^3e^3)$	$O(ca^3e^3)$
AQ15	$O(sa^2ve^3)$	—	$O(sa^2ve^3)$	—

Notes:
$s = MAXSTAR$
$c = |C| = |V_c|$
$a = |A|$
$v = |V_{max}|$
$e = |E|$

Table 3.3: Summary of complexity estimates.

Table 3.3 summarises the worst-case computational complexity estimates produced in this chapter, grouping them according to the different types of attributes each algorithm can handle. The following need to be noted about the contents of this table:

- *Structured attributes* are not included, since only AQ15 can handle them.
- *Ordinal attributes* are handled in the same way as integer ones and are thus omitted as well.
- C4.5 and NewID allow for "*special treatment*" of nominal attributes.
- The cubic estimate is presented for PLS1, because the over-quadratic one cannot be justified by its description.

In general the following comments can be made about the complexity results presented in the chapter:

1. Most of the algorithms can handle *nominal attributes* quite efficiently, with respect to the size of the training set, but they do worse in terms of the size of the attribute set and the value-sets. This is because the search space is limited by the domain-definition of the nominal attributes.
2. The use of *value-groups* (binary split in the case of NewID) for nominal attributes, increases even more the order of complexity with respect to the size of the value-sets. At the same time the complexity with respect to

the number of examples increases, since the restrictions set by nominal attributes to the size of the search space are removed.

3. The complexity of the algorithms which can handle *integer* and *continuous* attributes is the same for both these types.

4. AQ15 imposes an *upper bound to the number of values* of an integer attribute, thus decreasing the order of complexity of the algorithm.

5. The value of $MAXSTAR$ does not seem to affect the worst-case complexity of the CN2 algorithm, although this is not expected to hold in the average case.

Nevertheless, one has to be very careful with the interpretation of the results of a worst-case analysis. The situations which were assumed, in order to obtain those results are extreme and very atypical of the problems that a concept-learning system will usually be required to solve. Thus, it is expected that the results of an average-case analysis would be very different. The following chapter examines the scalability of the algorithms in an experimental way, using artificial and natural data, in order to confirm the validity of the results reported in this chapter and examine how these compare to the performance of the algorithms in typical situations.

Chapter 4

Scaling up on Real and Artificial Data

4.1 Introduction

The claim made in chapter 3 was that the worst-case computational complexity of the examined algorithms is over-quadratic and in some cases cubic. This claim however is based on the occurrence of extreme situations which are atypical for normal classification tasks. The first objective of this chapter is to compare these worst-case results with the results of an experimental analysis of the algorithms, using two real-world data sets. These data sets describe typical classification tasks aiming to provide an average-case indication of the performance of the algorithms. In addition to that, an artificial problem is used, in order to examine the validity of the theoretical results. This task describes a typical situation where the learning methods used in the examined algorithms are inadequate, resulting to their near-worst case performance.

There have been numerous experimental analyses and comparisons of Machine Learning algorithms in the past (e.g. [O'Rorke, 1982], [Rendell, 1986], [Gams and Lavrac, 1987], [Fisher and McKusick, 1991], [Mooney *et al.*, 1991], [Weiss and Kapouleas, 1991], [Nakhaeizadeh *et al.*, 1993], etc.), a few of which have examined the computational performance of the algorithms. The ones which are relevant to the work presented here are [O'Rorke, 1982] and [Rendell, 1986], which however suffer from the following problems:

1. They use **old versions of the algorithms**, most of which cannot handle numeric attributes.
2. The **data sets** that are used are **too small** (i.e. < 1000 data points) for an adequate computational performance evaluation of the algorithms.

3. They perform a general comparison of the algorithms, of which the performance evaluation is only a small part. As a result, the **evaluation is rather shallow**.

On that basis, the analysis presented in this chapter is a specialised one, examining in depth the scaling behaviour of the five concept-learning algorithms on real-world classification tasks. Most of the algorithms are state-of-the-art ones[1] and the data sets are substantially larger than the ones used in the work cited above.

4.2 Description of Data Sets

4.2.1 Selection Criteria

The main criterion for the selection of the real data sets is their size. At this point, an important distinction has to be made between the *size of the data set* and the *size of the search space*, both of which are relevant to the problem of scalability. The size of the data set is defined by the number of instances that the set contains. The search space on the other hand is defined by the *type* and the *domain* of the features used to describe the concept.

Although the size of the training set is the main variable which is used to evaluate the scalability of the algorithms, the type of attributes that are used is also a very important factor for the selection of the data sets. This is due to the observation that the computational complexity estimate of the algorithms is usually much worse for numeric than it is for nominal attributes. Thus problems which use numeric attributes have been favoured for this analysis.

Other criteria, which have affected to a smaller extend the selection of the data sets are the following:

- The amount of *noise* in the training set. Both the computational and the classification performance of the algorithms tend to degrade as the amount of noise in the data set increases. Seen from a different point of view, the effects of noise with respect to an increase in the size of the training set is a very interesting issue, affected also by the distribution of the noise.

- The *type* of the problem. It is a common practice of the developers of ML algorithms to test them on toy-problems that they construct. This allows them to test specific aspects of the algorithm. However, very rarely the

[1]Except from PLS1 which has not been improved substantially in the recent past.

training sets that they use combine all the problems that appear in real-world domains. This is usually true about the size of the data set. Toy-problem data sets are seldom as big as real-world ones. Thus real-world problems are favoured for the scalability analysis of the algorithms.

- The *structure* of the problem is also important since some problems require special representation schemes which are not supported by some or all of the algorithms used in this project.

The main source of data sets that has been considered is the *UCI Repository of Machine Learning Databases* [Murphy and Aha, 1993], which contains a number of data sets used in Machine Learning research in the past. Appendix B examines some of those data sets, classifying them according to the criteria listed above.

4.2.2 Recognising Typed Letters

The first real data set used in the experimental analysis is the '*Letter Recognition*' one, which was donated to the UCI Repository by D.J. Slate. The data set has originally been used by its author as an application domain for Holland-style genetic classifier systems[Frey and Slate, 1991]. More recently the data set has also been used in the StatLog project[Nakhaeizadeh *et al.*, 1993].

The task in this set is to classify typed uppercase letters of the Latin alphabet, based on a number of statistical properties of their pixel images. The initial characters that are used belong to 20 different fonts, including normal Latin ones, script, italics and Gothic. These characters are randomly distorted in four different ways:

- Horizontal magnification.
- Vertical magnification.
- Horizontal warp.
- Vertical warp.

introducing thus noise to the data. The resulting images are then encoded in terms of 16 of their first and second-order statistical properties, which are further scaled to the common integer range 0-15 [2]. The final scaled values constitute the features used in the classification problem.

[2]This scaling provides a convenient representation for the genetic classifiers used by the author of the data set.

The data set contains 20,000 instances, which are generated in the manner described above. Examples of the features used to describe each instance are the following[3]:

1. The width and the height of the smallest rectangular box containing the character, which measures the size of the character.
2. The horizontal and vertical mean positions of "on" pixels, which measures the distribution of "on" pixels in the pixel image.
3. The horizontal and vertical variance, based on the corresponding mean positions.

The class feature is simply the uppercase letter corresponding to the instance. Finally, the data set does not contain any missing values.

The main reason why this data set was selected is the number of instances that it contains. Moreover the use of bound integer attributes makes it very interesting, since one of the algorithms (AQ15) cannot handle unbound ranges of integers. Thus, this type of attribute allows the investigation of all five algorithms, in a numeric feature space. Finally, the fact that the set has been previously used allows for comparison of the results presented here with the ones previously presented.

4.2.3 Classifying Chromosomes

The second real data set used in the experiments describes a chromosome analysis task. It is the Copenhagen data set, used also in [Errington and Graham, 1993a] and [Errington and Graham, 1993b], where an artificial Neural Network system is used for the classification of chromosomes. The Neural Network is shown to achieve higher classification performance than other statistical classifiers which have been used in the past on the same task. The data set was provided by the Department of Medical Biophysics, University of Manchester.

Each instance of the set corresponds to a chromosome, which is described in terms of 15 features, extracted in the following way:

1. Chromosomes in cells which are in the metaphase stage, are stained, producing a series of bands along their length (G-banding).
2. A Grey-level profile is produced for each chromosome, giving an indication of the banding pattern.

[3]Refer to [Frey and Slate, 1991] for a complete description of the attribute set.

3. The profiles are scaled in order to eliminate length and grey-level variations between the chromosomes.

4. The scaled profiles are discretised at 15 positions along the length of the chromosome, using local averaging.

The result of this process is a vector of 15 real-valued attributes, which correspond to the 15 average grey level values.

The data set contains 8,106 examples, described in terms of the 15 extracted features. In [Errington and Graham, 1993a] and [Errington and Graham, 1993b] two more attributes are being used (the length of the chromosome and its *centromere index*[4]), which have not been used in this analysis. The class attribute can take 24 independent values, corresponding to the 24 different types of chromosomes (22 "autosomes" and 2 "sex chromosomes"). Finally, due to the sampling method used, the quality of the data set is considered very good, minimising the possibility of noise.

The most interesting characteristic of this data set is the fact that it uses continuous attributes. Although only three of the examined algorithms can be tested on this set (AQ15 and PLS1 cannot handle continuous attributes) it is interesting to examine the effect that real, instead of integer-valued, attributes may have on the computational performance of the algorithms. In addition to that, the set is fairly large and it has been used in the past, allowing for a comparison of the classification performance of the symbolic algorithms used in this project to the numeric one used in [Errington and Graham, 1993a] and [Errington and Graham, 1993b].

4.2.4 Learning Even Numbers

This is a simple artificial problem, whose purpose is to examine the validity of the worst-case estimates produced in the previous chapter. The task is the discrimination between even and odd integers, provided no more information but the integers themselves. The size of the data set is obviously unlimited, while the size of the training sets used in the experiment is externally defined.

Each set contains the first n integers, where n is the set's size. There is only one integer attribute used for the problem and the class attribute is a binary one, taking the values *true* and *false* when the attribute value is an even or an odd number respectively.

The key feature of this problem is that there is no knowledge encoded in the feature set. Instead of providing information about properties of the instances

[4]This is an indicator of the size of the small arm of the chromosome as a proportion of its whole length.

(integers), raw data (i.e. the integers themselves) are provided. As a result there are no similarities between the instances, on which induction can be based.

Also very important is the pattern that the data follow, which is one that cannot be detected by the examined algorithms[5]. When provided with numeric attributes, the algorithms look for ranges of those attributes, sharing the same class value. These ranges serve as sides of rectangular regions in the hyperspace. In this case, the outcome of that method is a large number of ranges, each of which contains only one instance. Clearly, this discrimination method is not suitable for the problem, because there are no adjacent attribute values sharing the same class value.

In addition, the problem has a number of characteristics, which make it suitable for the experimental worst-case analysis of the algorithms:

1. The **attribute is numeric**, defining an infinite search space.
2. The fact that the **class is binary** and there is **only one attribute** minimises the effect that other factors apart from the number of examples have on the computational performance of the algorithms.
3. The entropy heuristic, used in a number of algorithms, favours splits near the ends of the range of integers, used as instances. The result of this is that decision trees are not only **complete** but **highly skewed** as well.

Given the above features, this problem is expected to force the examined algorithms to near worst-case behaviour, similar to that assumed in the theoretical analysis.

4.3 Test Organisation

4.3.1 Measuring Scalability

The problem of measuring the scalability of an algorithm, using a specific data set can be defined in terms of an independent variable, which is the size of the training set, and the quantity being measured, which is the rate of change in the computational performance of the algorithm. The desired outcome is a function of the form:

$$P_a = f(S_t)$$

[5]Other types of ML algorithms, notably the ones used in Inductive Logic Programming, can easily cope with this problem.

defining a relationship between the performance of the algorithm P_a and the size of the training set S_t, which can be translated to a computational complexity estimate.

In order to derive the above relationship, the CPU-time consumption of the learning process is measured at different '*size-steps*'. *CPU-time consumption* is defined in this case as the '*user time*' component of the overall time consumption of the process, excluding things like the *swapping time*, which depends on the memory capacity of the machine and can have an undesirable effect on the final measurement. Built-in C and Pascal functions (**getrusage** and **clock** respectively) are used for the measurement of time consumption.

The size-steps that are used are determined in an exponential way, starting from a small power of 2, usually $2^6 = 64$ instances, and multiplying by 2 each time, up to the step closest to the 3/4 of the size of the whole data set[6]. The reason why the size of the training set S_t is varied exponentially is because the rate of change, rather than the absolute value, of the computational performance of the algorithm is examined. Moreover, in order to reduce the possibility of erroneous and coincidental measurements, at each size-step three random subsets of the data set are used and the results of the three tests are averaged.

The results of this analysis are plotted on a logarithmic scale for both axes and the slope of different parts of the resulting line are examined. In most cases, an additional fitting of several standard curves is attempted, approximating the order of increase of the learning time. The curve that best fits the generated line can finally be compared to the theoretical estimate of the complexity of the algorithm.

4.3.2 Measuring Classification Accuracy

In addition to the computational performance, for real data sets, the classification accuracy of the decision function, generated by each algorithm, is measured on unseen test cases. The objective of this measurement is to verify that algorithms which do well in computational terms, do not do so at the expense of low classification performance.

There are a number of statistical methods for measuring the classification performance of an algorithm[7]. One of the most reliable ones that has been used in the past is *cross-validation*, where the data set is divided into k subsets, each of which is used as the training set and the rest as a test set. The results of the k runs are averaged to produce the classification accuracy of the algorithm for a training set of size equal to each of the k subsets.

[6]The remaining 1/4 is used as a test set of unseen cases.

[7][Nakhaeizadeh *et al.*, 1993] gives an interesting account of them.

The main problem with this approach is that it is computationally heavy, especially for large size-steps. An alternative to that, which has also been very popular, is to select at random 3/4 of the data set to be used as the training set and use the remaining 1/4 as a test set. A variation of this method is used here as well. A test set, which is 1/4 of the whole data set, is randomly selected for testing and from the remaining 3/4 a subset of the required size is selected for training. Again three tests are done at each size-step and the results are averaged to give the classification accuracy of the algorithm for that size.

As in most similar analyses, the classification accuracy of an algorithm is measured as the percentage of correctly classified instances of the test set. The results are plotted on a logarithmic scale for the horizontal axis (size of training set) and a linear one for the vertical axis (accuracy). The outcome of that plot is known as the '*learning curve*' of the algorithm.

4.3.3 Hardware Specification

All the experiments presented in this chapter were carried out on a 'Sun-SPARCsystem-400' machine[8], which contains 32 MBytes of fixed memory and 100 MBytes of swap memory. The system runs the 'SUNOS 4.1.2' operating system.

4.4 Experimental Results

4.4.1 Letter Recognition

Set-up

According to the '*size-step*' scheme introduced above, this data set can be used to produce 9 steps. Set sizes vary from 64 to 16,000 training instances. The size of the test set is always around 4,000 instances, which are different from the ones used in the training set. At each step three tests are done and the results are averaged to produce an estimate of the computational and the classification performance of the algorithms. The maximum deviation from the mean value is also reported for each step.

In this experiment, all five algorithms can be used, since all of them can handle bound integer attributes. The following are the parameter settings for each of the algorithms:

[8] A server machine similar to a SPARC2.

C4.5: *Default settings.*

NewID: *Default settings.*

PLS1:
1. The parameter *mintotalcount*, which determines how many instances a region must contain, in order to be considered for splitting, is set to 2, aiming at perfect discrimination.
2. Also *maxspansteps* is set to 15, because all of the integer attributes used have a value range 0..15. Thus 15 steps allow the examination of all possible splitting points, between the 16 feature values.

CN2:
1. The *ordered* version of the algorithm is used (described in section 3.5).
2. The *Laplacian* error estimate is used, because it has proved [Clark and Boswell, 1991] to be better than the *Entropy* one.
3. The *maximum star size* parameter has been set to 7, which has proved, during some initial experimentation, to be large enough for the generated decision rules.

AQ15:
1. *Disjoint covers* are generated, similar to the other examined algorithms.
2. The *maximum star size* parameter has been set to 15, which is the size of the feature-set[9]. The setting of this parameter however is suspected for the low classification performance of the algorithm in the experiment. Some experimentation was also done, using a value of 7 (similar to that used for CN2), which has provided worse classification results.

From the above list, it is clear that wherever possible, the default values were used. The setting of the *maximum star size* parameter, for the CN2 and AQ15 algorithms, has proved to be a difficult task, since the value of this parameter affects substantially the performance of the algorithms.

Scalability Results

The results of the computational performance part of this experiment are summarised in figure 4.1 [10].

The first thing to note about these results is that they do not agree with the theoretical estimates presented in the previous chapter. The rate at which the CPU-time consumption increases, as the size of the training set is doubled at

[9] This is a guideline appearing in [Hong *et al.*, 1986].

[10] Appendix C contains the tables corresponding to the graphs that are presented in this section.

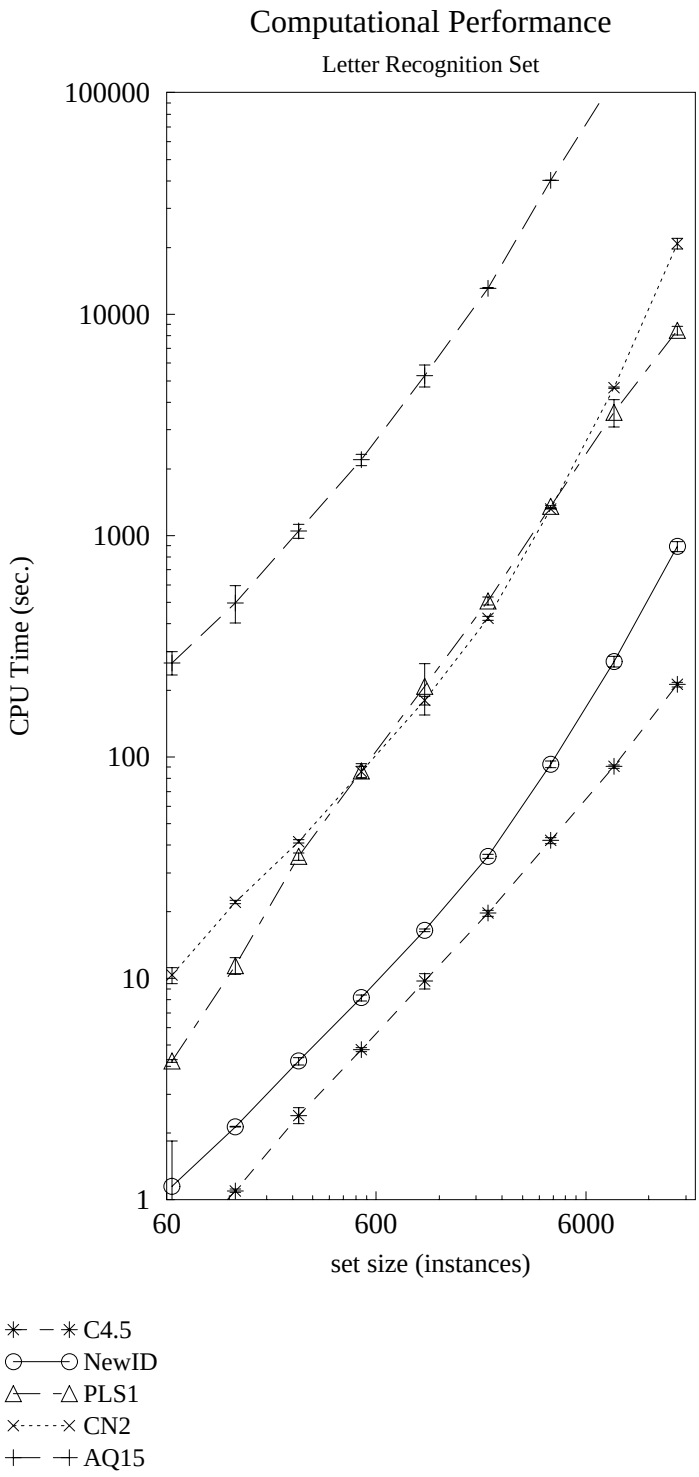

Figure 4.1: Scalability Results, using the *Letter Recognition* data set. (corresponds to table C.1)

each step, is slightly above linear. This is illustrated more clearly in figure 4.2, where the rate at which the CPU-time consumption increases at each size-step is examined for each algorithm. This rate is calculated by the following formula[11]:

$$r_s = \frac{c_s}{c_{s-1}}$$

where r_s is the calculated ratio at size-step s, while c_s and c_{s-1} correspond to the CPU-time consumption of the algorithm at steps s and $s-1$ respectively.
If the behaviour of the algorithm was linear, the outcome of this transformation would be a straight horizontal line, crossing the y-axis at 2, which is the rate at which the size of the training set increases at each step. This relationship is shown by the lower thick dashed line in the diagram. Similarly, a quadratic relationship would produce the upper horizontal dashed line, intercepting the y-axis at the rate of 4.

Most of the algorithms (except PLS1) start with a very close to linear behaviour, some of them actually lying below the *linear threshold*. The explanation for this is given by the set-up cost involved in each algorithm, which is relatively fixed and accounts for most of the time-consumption for small training sets. This also explains the high increase rate, which is observed for most of the algorithms, at the second size-step (256 instances). As the size of the training set becomes quite large, different algorithms behave differently, although a general upwards trend in the rate of increase is observed. More specifically:

- For **C4.5** the effect of the fixed costs gradually disappears, leading to a behaviour very close to linear for the size-steps between 512 and 2,048. For size-steps above this point the rate estimate increases, leading to the conclusion that the overall behaviour of the algorithm is slightly above linear.

- **NewID** and **CN2** start with a low rate, which however increases steadily after the third size-step and this increase becomes steeper as the training set becomes larger. For the largest sets used, the two algorithms become very expensive, CN2 moving even above the *quadratic threshold*. This behaviour suggests that the algorithms use some optimisation method which cannot handle very large data sets.

- **PLS1** is the only algorithm which starts well above the linear threshold and remains above it throughout the experiment. For the first three steps its behaviour follows the normal pattern, explained by the fixed initialisation costs. After that, the rate of increase remains fairly steady, showing a temporary increase for the two penultimate tests and a decrease for the

[11]This formula is a substitute of the first-derivative, which cannot be used due to the small number of data points.

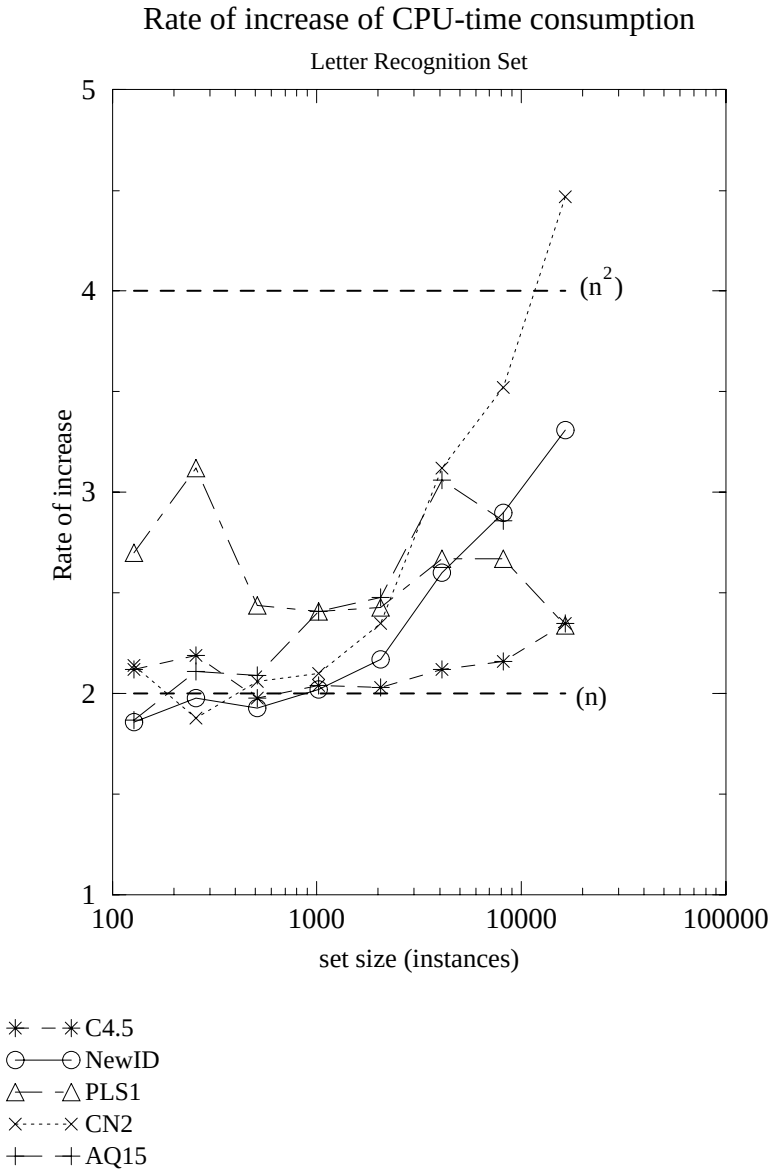

Figure 4.2: *Letter Recognition Set:* The rate of increase of the CPU-time consumed at each size-step. (corresponds to table C.2)

largest training set. Thus, the relationship between the size of the training set and the cost of the algorithm seems to be a steady one.

- **AQ15** behaves similar to the average behaviour of the other algorithms, starting from a very low rate of increase, which becomes large as the size of the training set increases.

In order to confirm the validity of the linearity claim, simple linear regression was used, attempting to fit a straight line to the original as well as a modified version of the data. In the modified version, the CPU-time consumption values were changed according to the following formula:

$$c'_s = \frac{c_s}{\ln(x_s)}$$

where c_s and c'_s are the original and the modified consumption values and x_s is the set size at step s.
Fitting a straight line to these data, approximates the $n \ln(n)$ relationship between the CPU-time consumption and the size of the training set. Tables 4.1 and 4.2 summarise the results of this analysis.

Algorithm	*correlation coefficient*	*regression coefficient (slope)*	*regression coefficient error*	*t-value for regression coefficient*
C4.5	0.997	0.013	0.0004	33.557
NewID	0.977	0.052	0.0430	12.056
PLS1	0.995	0.513	0.0190	26.272
CN2	0.957	1.200	0.1370	8.710
AQ15	0.985	13.81	0.9800	14.062

Table 4.1: Regression analysis for linear behaviour on the *Letter Recognition Set*.

Using those results the following observations can be made:

- Based on the values of the *correlation coefficient*, one concludes that there is a significant linear relationship between the CPU-time consumption of the algorithms and the size of the training set. This relationship is stronger for C4.5 and PLS1 and it becomes even stronger in the analysis of the log-linear behaviour.

- The slope of the fitted line, given by the *regression coefficient* is higher for the generalisation algorithms, especially for AQ15, than for the specialisation ones. This is a result of the high computational overhead imposed by the former category of algorithms, which has also been observed in figures 4.1 and 4.2.

Algorithm	*correlation coefficient*	*regression coefficient (slope)*	*regression coefficient error*	*t-value for regression coefficient*
C4.5	0.999	0.0010	$2.17e^{-5}$	60.57
NewID	0.982	0.0054	$4.0e^{-4}$	13.59
PLS1	0.998	0.0530	$1.3e^{-3}$	39.48
CN2	0.962	0.1230	$1.3e^{-2}$	9.28
AQ15	0.989	1.5300	$9.0e^{-2}$	16.88

Table 4.2: Regression analysis for log-linear behaviour on the *Letter Recognition Set*.

- Finally the low *regression coefficient error values* and the high *t-values* support the claim of a strong linear and an even stronger log-linear trend in the data.

Another important observation to be made on figure 4.1 is the large difference between the absolute values of the cost of each algorithm. Although all algorithms behave in a near-linear fashion, some of them become prohibitively slow for large data sets (e.g. AQ15). The extreme case is the difference of AQ15 from C4.5, which is given by the intercept of the corresponding lines in the diagram. In general, algorithms using specialisation (i.e. C4.5, NewID and PLS1) seem to impose a lower computational overhead than the generalisation ones. The slower of the former category (i.e. PLS1), which is a very old algorithm is comparable to the fastest of the latter category (i.e. CN2).

Classification Accuracy Results

The results of the classification performance analysis of the algorithms are presented in figure 4.3[12].

Although the classification performance of the algorithms is not the central issue of this analysis, a number of interesting observations can be drawn on the obtained results:

1. Most of the algorithms perform similarly. The only exception is AQ15, which performs substantially worse. The reason for that is probably the parameter-settings that have been used and in particular the maximum star size parameter.

[12]The reason why the learning curves have a slightly different shape than usually, is because a logarithmic scale is used on the x-axis.

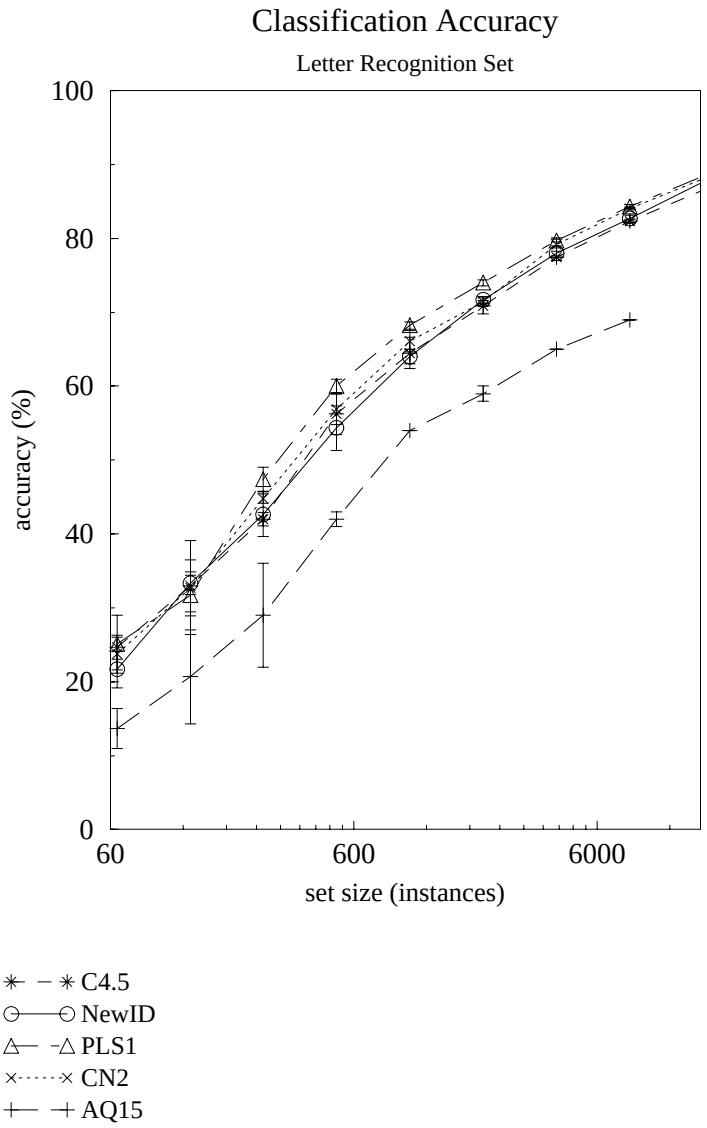

Figure 4.3: Classification Accuracy Results, using the *Letter Recognition* data set. (corresponds to table C.3)

2. The classification accuracy of the algorithms seems to converge just below 90%. This is a much better performance than the one presented in [Frey and Slate, 1991], who have used genetic classifiers on the same set. Their highest achieved accuracy, using a training set of 16,000 instances was just above 80%.

3. As expected the deviation from the presented mean value decreases as the size of the training set increases and the generated classifiers become more reliable.

4. A further indication that something is going wrong with the settings for AQ15 is the high deviation from the mean value for this algorithm.

4.4.2 Chromosome Classification

Set-up

This data set is smaller than the Letter Recognition one. It contains 8,106 data points which provide 7 size-steps, ranging from 64 to 4,096 instances. In addition to those, an eighth test was carried out, using 5,432 instances, which correspond to 2/3 of the whole set. In a similar manner to the Letter Recognition experiment, three tests were done per step, averaging the results to produce the estimated performance of each algorithm. In this experiment 1/3 of the whole data set (i.e. 2,716 data points) is used in the test set, when measuring classification performance.

As mentioned in the description of the data set, the attributes that are used are real numbers and because of that, only three of the algorithms can be used. The parameter settings for them are as follows:

C4.5: *Default settings.*

NewID: *Default settings.*

CN2:
1. In contrast to the Letter Recognition experiment, the *unordered* version of the algorithm was used.
2. The *Laplacian* error estimate was used here as well.
3. The *maximum star size* parameter is also the same as in the previous experiment (i.e. it is set to 7), although the performance of the algorithm suggests that this is not its optimum value for this problem. However, some experimentation with different settings has not provided better results.

Again default settings were used wherever possible.

Scalability Results

Figure 4.4 presents the results for the computational performance of the algorithms in this experiment.

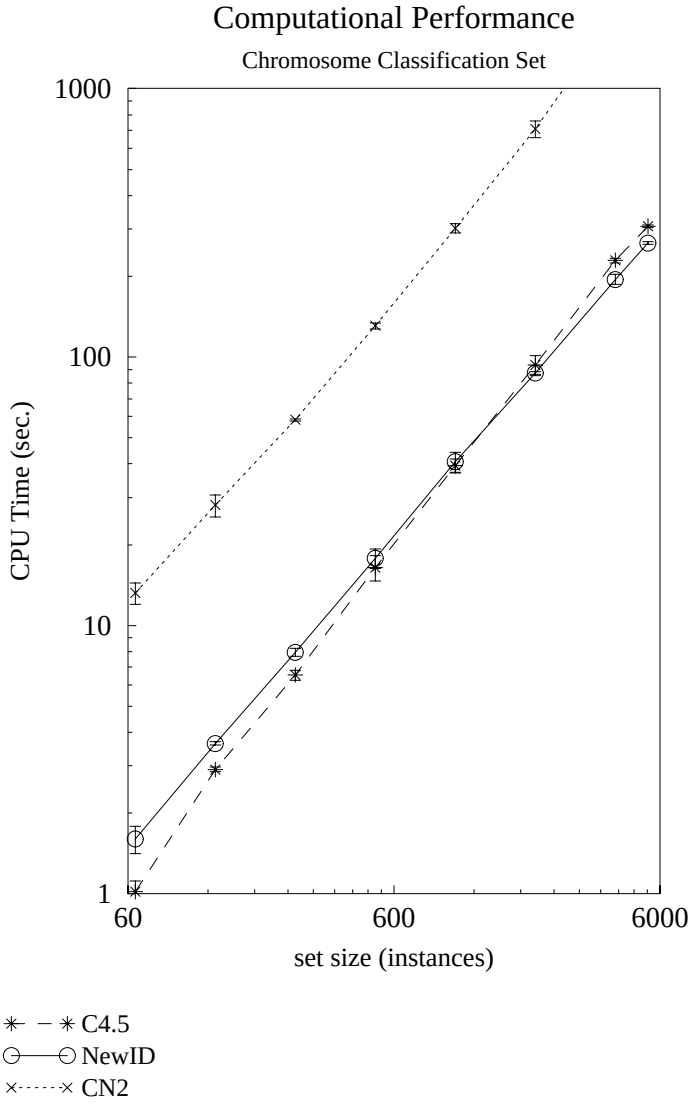

Figure 4.4: Scalability Results, using the *Chromosome Classification* data set. (corresponds to table C.4)

In general the behaviour of the algorithm is very similar to the previous experiment. All three algorithms have a near-linear behaviour and the "generalisation" one (i.e. CN2) is more expensive than the two "specialisation" ones. However there are a few things to be noted about the results, which differ from the previous experiment:

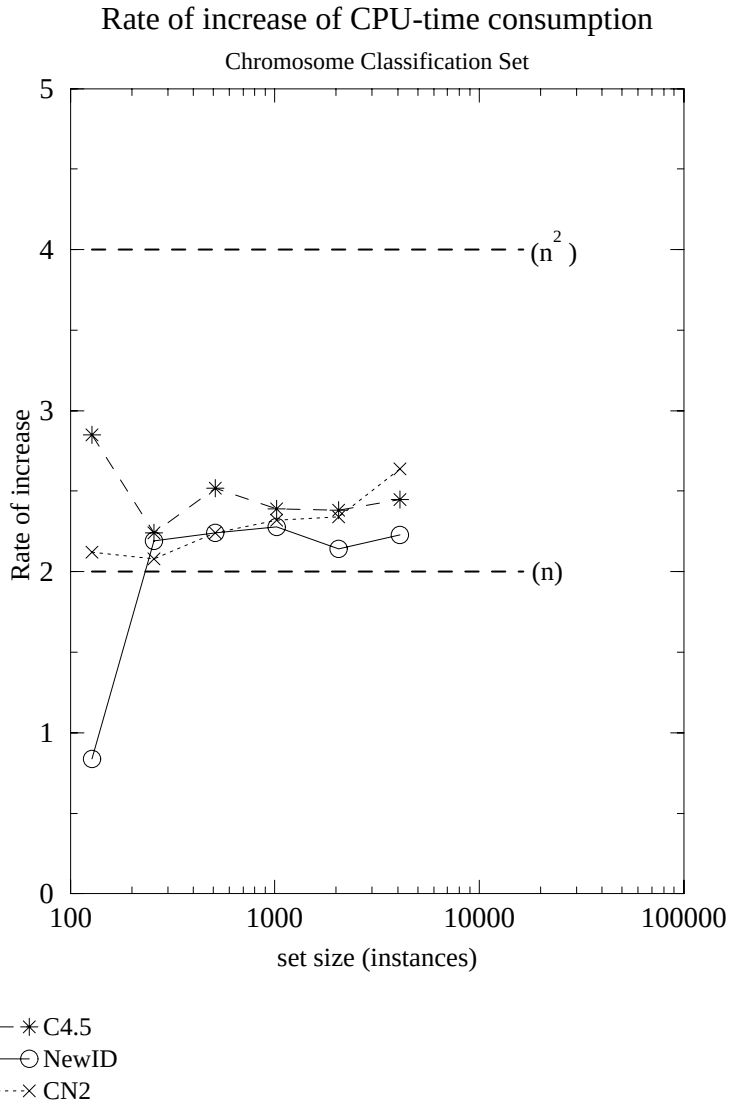

Figure 4.5: *Chromosome Classification Set:* The rate of increase of the CPU-time consumed at each size-step. (corresponds to table C.5)

1. The absolute CPU-consumption values are slightly higher than the corresponding ones for the previous set. Since the number of attributes and classes are smaller for this data set and the only major difference in the definition of the search space is caused by the type of the attributes that are used, it is suspected that the computational overhead imposed by each algorithm is higher for continuous than it is for integer attributes.

2. Another possible explanation for the higher CPU-time consumption values could be the higher complexity of the problem. In that case however a higher rate of increase rather than higher absolute values would be expected [Rendell, 1990]. Comparing the results presented in figure 4.5 with those in the previous section (figure 4.2) the rate of increase of the CPU-time consumption seems to be lower in general for this problem. In fact, the average increase rate for NewID and CN2 is considerably lower than in the Letter Recognition experiment[13].

3. A further interesting observation that can be made on the results of figure 4.5 is that the rate of increase for the three algorithms remains fairly stable. The behaviour of the algorithms remains very close to linear throughout the experiment, although CN2 becomes slightly more expensive for larger training sets. It seems possible however that if larger sets could be used a large increase in the cost of NewID and CN2, similar to the first experiment, would have been observed.

4. A final point to be made on figure 4.4 is that the absolute consumption values for C4.5 are only minimaly lower than the ones for NewID, for small training sets, while the algorithm becomes more expensive than NewID for larger sets. This is a major difference to the results of the previous experiment (figure 4.1), which seems to be caused by the fact that NewID can handle continuous attributes more efficient than C4.5. This conclusion is also supported by the fact that the rate of increase for NewID is lower than that of C4.5.

The regression analysis results (tables 4.3 and 4.4) support the conclusions drawn above. More specifically:

- The *correlation coefficient* values and the *t-values* show a stronger linear behaviour than in the Letter Recognition experiment, which is even better for the log-linear relationship. In fact, NewID seems to fit almost perfectly to the log-linear behaviour estimate.

- In contrast to the previous experiment, the linearity of NewID is higher than that of C4.5. This agrees with the conclusions drawn above.

[13]The actual average values are given in tables C.2 and C.5.

- The estimated *regression coefficient* values for CN2 and NewID are lower than in the previous experiment, due to the lower average increase rate for the two algorithms.

Algorithm	*correlation coefficient*	*regression coefficient (slope)*	*regression coefficient error*	*t-value for regression coefficient*
C4.5	0.998	0.058	0.0016	36.989
NewID	0.999	0.049	0.0009	57.656
CN2	0.996	0.482	0.0185	26.080

Table 4.3: Regression analysis for linear behaviour on the *Chromosome Classification Set.*

Algorithm	*correlation coefficient*	*regression coefficient (slope)*	*regression coefficient error*	*t-value for regression coefficient*
C4.5	0.999	0.0067	$1.03e^{-4}$	64.95
NewID	1.000	0.0057	$1.76e^{-5}$	323.24
CN2	0.998	0.0559	$1.48e^{-3}$	37.73

Table 4.4: Regression analysis for log-linear behaviour on the *Chromosome Classification Set.*

Classification Accuracy Results

Figure 4.6 shows the results of the classification performance part of this experiment. The conclusions drawn on these results are to a large extent similar to those in the Letter Recognition experiment:

1. The performance of NewID and C4.5 is very similar, converging to a classification accuracy around 80%.

2. The deviation from the estimated accuracy value is in general higher than in the previous experiment, especially for large data sets, because it decreases a lot slower. Using this as an indicator of reliability, one can argue that the algorithms do not provide very good solutions to the problem.

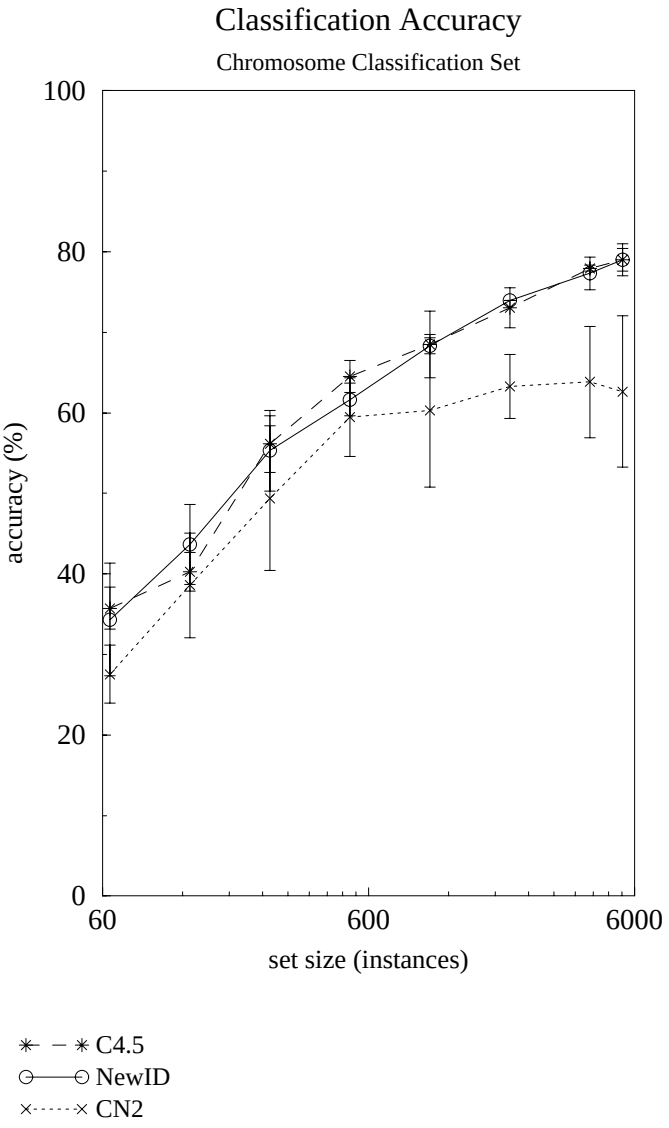

Figure 4.6: Classification Accuracy Results, using the *Chromosome Classification* data set. (corresponds to table C.6)

3. The suspicion that the performance of the algorithms is not very good is confirmed by the high classification accuracy that the Neural Network classifier achieves in [Errington and Graham, 1993a]. The authors of those papers report an accuracy of 91.2%, using only half of the data set for training (i.e. 4,053 instances). This can probably be attributed to the nature of the concept to be learned, which cannot be approximated sufficiently by a set of orthogonal hyperrectangles.

4. The behaviour of CN2 is very similar to that of AQ15 in the previous experiments (i.e. very low accuracy and high deviation from the mean value). This is probably caused by the parameter settings used for the algorithm.

4.4.3 Learning Even Numbers

Set-up

As mentioned above, this is an artificial problem and therefore training sets of unlimited size can be used. The '*size-step*' convention adopted in the other experiments is used here as well, with set-sizes ranging from 64 to the maximum number of instances that each algorithm can handle. The limits of each algorithm in that respect are as follows:

- **C4.5:** The maximum set size used for this algorithm was 8,192 instances, but the algorithm can handle more than that.
- **NewID:** 4,096 instances.
- **PLS1:** 1,024 instances.
- **CN2:** 4,096 instances.
- **AQ15:** 2,048 instances.

Another difference of this experiment with the other ones is that only one test is done per step, since the construction of the training sets is done deterministically, rather than by random selection of instances.

All five algorithms were used in this experiment and their parameter settings are as follows:

C4.5: The termination criterion for this algorithm examines whether at least two branches contain a minimum number of objects. By default this number is

set to 2, but for this experiment it has been changed[14] to 1, in order for a complete tree to be generated.

NewID: *Default settings.*

PLS1:
1. *mingoodcount* is set to 0 and *mintotalcount* is set to 1, achieving complete region splitting.
2. *tsubalphaclus* is set to -1, in order for all splits to be considered significant.
3. *maxspansteps* is set to the size of the training set, leading to the examination of all possible splitting points.
4. The constant controlling the *maximum number of numeric attribute values* is set to $2,000$, which is larger than the maximum number of values that the algorithm can handle.

CN2:
1. *Unordered rule-lists.*
2. *Laplacian error estimate.*
3. The *maximum star size* parameter is set to the minimum value of 2, since all the rules produced are single-complex ones.
4. The *significance threshold* is set to 0, so that the complete set of rules is produced. Actually for any higher value of the threshold no rules will survive, since they are all very specialised classifiers.

AQ15:
1. *disjoint covers mode.*
2. *maximum star size* was set to 2, similar to CN2.
3. The constant responsible for the *maximum number of numeric attribute values* was set to $8,192$, but the algorithm can handle less values than that.

Due to the peculiarity of the results, some further experimentation was carried out for the NewID algorithm, subdividing the range between $1,024$ and $4,096$ into smaller steps and examining both the CPU-time and the memory consumption[15] of the algorithm for this subdivision. The steps that were examined are the following: $1,367$, $2,048 = 1.5 \times 1,367$, $2,176 = 1.125 \times 2,048$, $2,562 = 1.25 \times 2,048$, $3,074 = 1.5 \times 2,048$, $3,586 = 1.75 \times 2,048$, $4,096$. The default parameters of the algorithm were used in this experiment as well.

[14] This parameter is set by flag -m.

[15] Memory consumption was again measured by the **getrusage** C function.

Scalability Results

The dominating element in the results of this experiment is the rate at which the computational performance of the algorithms decreases as the training set becomes larger. This can be seen in the original results, presented in figure 4.7, where the slope of the performance curves is very steep. More clearly however it is shown in figure 4.8, where the behaviour of most of the algorithms quickly becomes over-quadratic and in some cases approaches the cubic threshold. These results are very different from the ones obtained using the real data sets and much closer to the theoretical estimates drawn in the previous chapter.

Examining each of the algorithms in turn the following can be noted:

1. **C4.5** is the only algorithm that handled training sets larger than $5,000$ instances. However, its behaviour is not near-linear as it was in the previous experiments. It starts with an increase-rate well above the linear estimate, which increases fairly quickly, crossing the quadratic and approaching the cubic estimate. This behaviour is worse than the theoretical estimate for the algorithm, suggesting that some of the optimisations assumed in the analysis have not been fully implemented. As a result of that, the performance of the algorithm becomes worse than that of NewID and CN2 for large data sets (figure 4.7).

2. **PLS1** behaves similarly, although its performance is worse than that of C4.5. It starts with an over-quadratic increase-rate and approaches the cubic threshold for the size-step of $1,024$ instances, which is the largest it can handle. At this stage it becomes even worse than AQ15, which is the slowest of the examined algorithms.

3. **NewID** behaves in a way which differs from the other algorithms. For most of the experiment its behaviour is less than quadratic and in some cases very close to the linear threshold. For the last size-step however its performance suddenly deteriorates very much, leading to an increase-rate well above the cubic estimate. At the same time the memory requirements of the algorithm increase to an extent that makes it impossible to run any experiments with size-steps above $5,000$ examples[16]. This strange behaviour of the algorithm suggests that some optimisation method has been used, which can handle data of up to a certain size. In order to investigate this problem, a further experiment was done using NewID only.

4. **CN2** and **AQ15** behave very similarly and much better than what was expected. Their increase-rate remains fairly stable, close to the quadratic threshold. As a result of that, the performance of some of the *specialisation* algorithms becomes worse than those two *generalisation-based* ones, for

[16] More than 120 MBytes of memory are required at this stage.

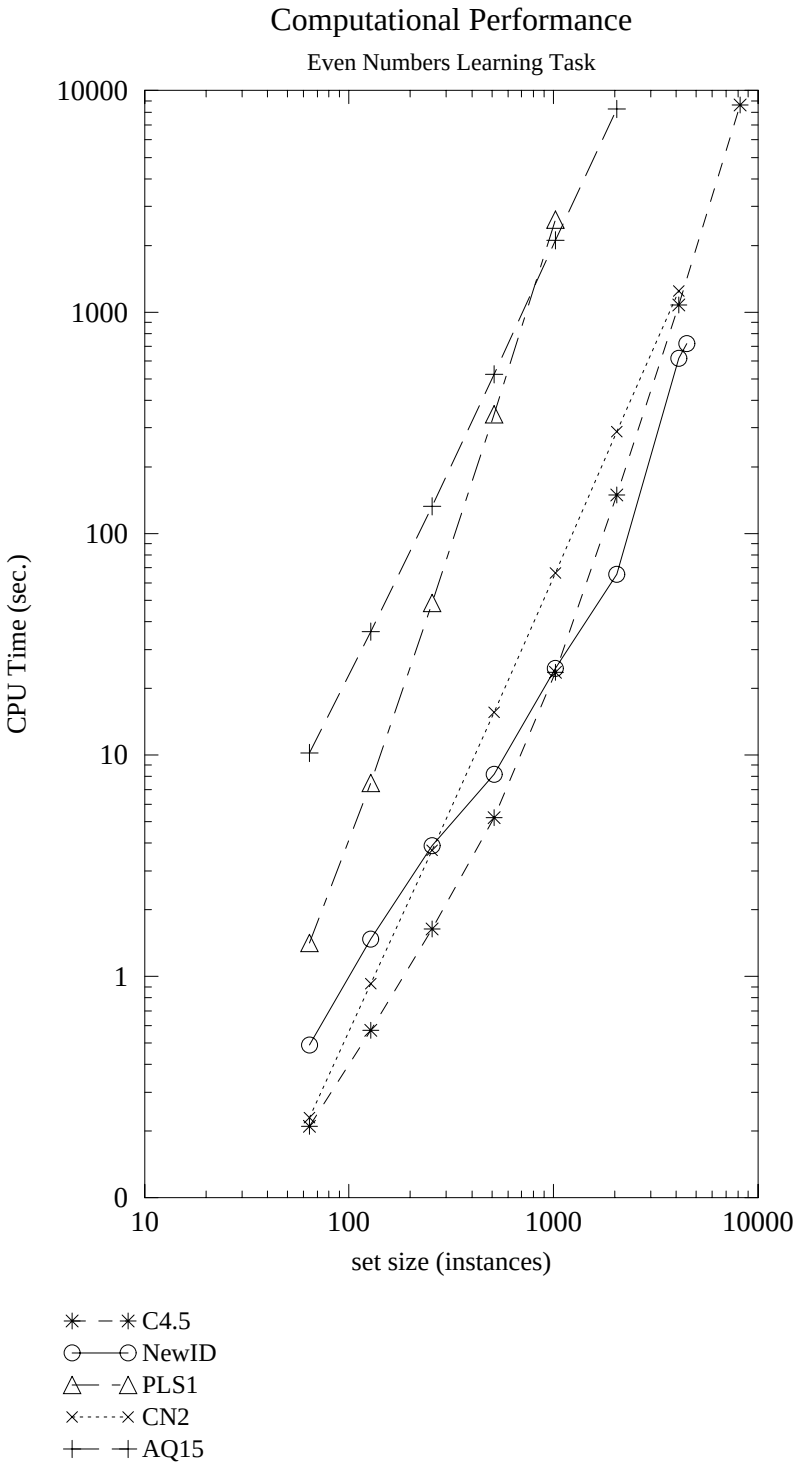

Figure 4.7: Scalability Results, using the *Even Numbers* learning task. (corresponds to table C.7)

large data sets (e.g. PLS1 vs. AQ15 and C4.5 vs. CN2). The fact that these algorithms behave near-quadratically, rather than cubicly, as predicted by the theoretical estimate, suggests that some decisive factor has not been taken into account in the theoretical analysis.

A further speciality of this experiment is that some components of the CPU-time consumption, which were considered fixed for most of the algorithms, are now expected to be related to the size of the training set, which is used in the definition of the search space (i.e. the number of attribute values that are used). As a result of that, the effect of the fixed costs cannot be isolated or minimised by the use of large training sets, although the behaviour of C4.5 and PLS1, can be interpreted as a gradual decrease of the interference of those costs, which keep the increase-rate relatively low for small sets.

The regression analysis that was carried out in this experiment is also substantially different from the previous two. Examining the average increase-rates (table C.8), quadratic and cubic models seem to fit better to the behaviour of the algorithms, than linear or log-linear ones. In order to prove that, a logarithmic transformation was applied to the data and linear regression was used on the result of this transformation. The results of this analysis are presented in table 4.5.

Algorithm	*correlation coefficient*	*regression coefficient (slope)*	*regression coefficient error*	*t-value for regression coefficient*
C4.5	0.990	2.183	0.130	16.84
NewID	0.987	1.665	0.110	15.11
PLS1	0.999	2.722	0.058	46.89
CN2	1.000	2.068	0.010	198.37
AQ15	1.000	1.939	0.017	111.13

Table 4.5: Regression analysis of log-time on sample-size for the *Even-number Learning Task*.

The results of this analysis need to be interpreted in a slightly different manner than the previous ones. What is important here is the *slope* of the fitted line, which determines the rate of increase in the consumption of each algorithm. In that respect C4.5, CN2 and AQ15 are very close to quadratic, PLS1 is almost cubic and NewID is more than linear but less than quadratic. These observations confirm the above presented conclusions.

Other things to be noted about the results of the regression analysis are the following:

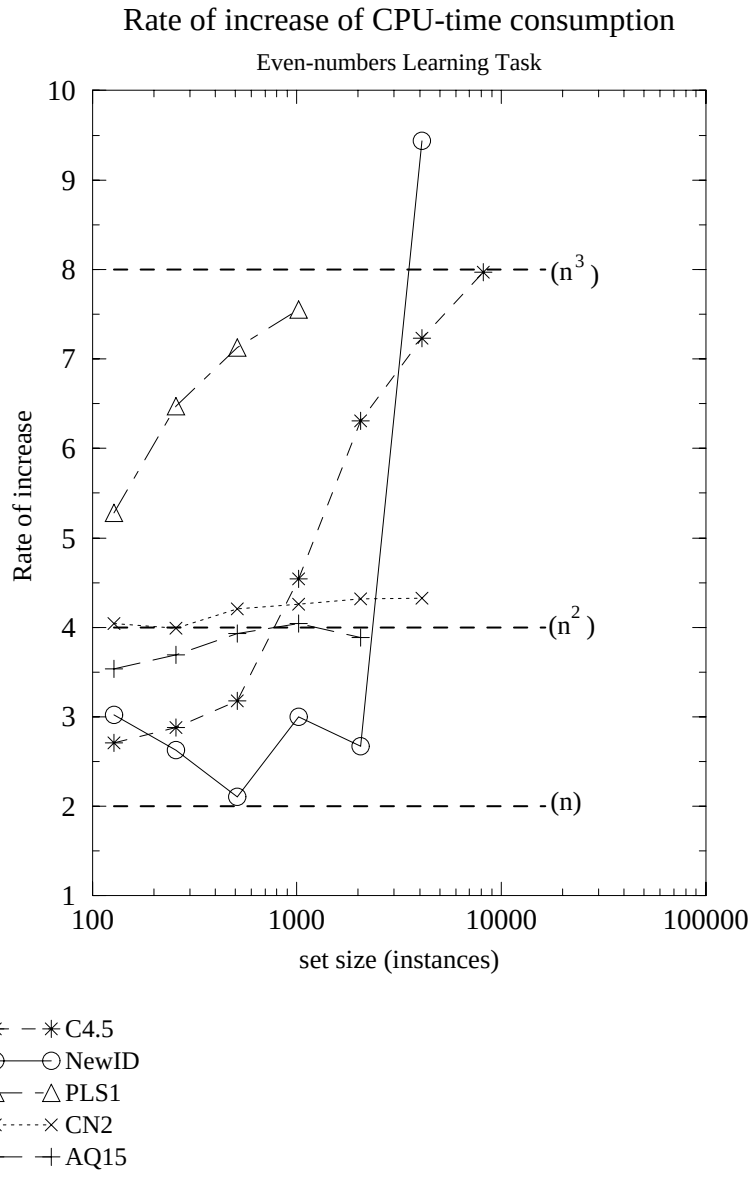

Figure 4.8: *Even-Numbers Learning Task:* The rate of increase of the CPU-time consumed at each size-step. (corresponds to table C.8)

1. CN2 and AQ15 fit very well to the quadratic estimate, having a *correlation coefficient* which is practically one, a very low *regression coefficient error* and very high *t-values*.

2. In general error rates are higher than in the previous experiments and *t-values* are lower. This is especially true for the specialisation algorithms, the behaviour of which is very unstable.

Thus the regression analysis has confirmed the conclusions drawn on the presented diagrams.

As mentioned above, the behaviour of NewID for a specific range of set-sizes has been investigated further. The results of this investigation are presented in figure 4.9.

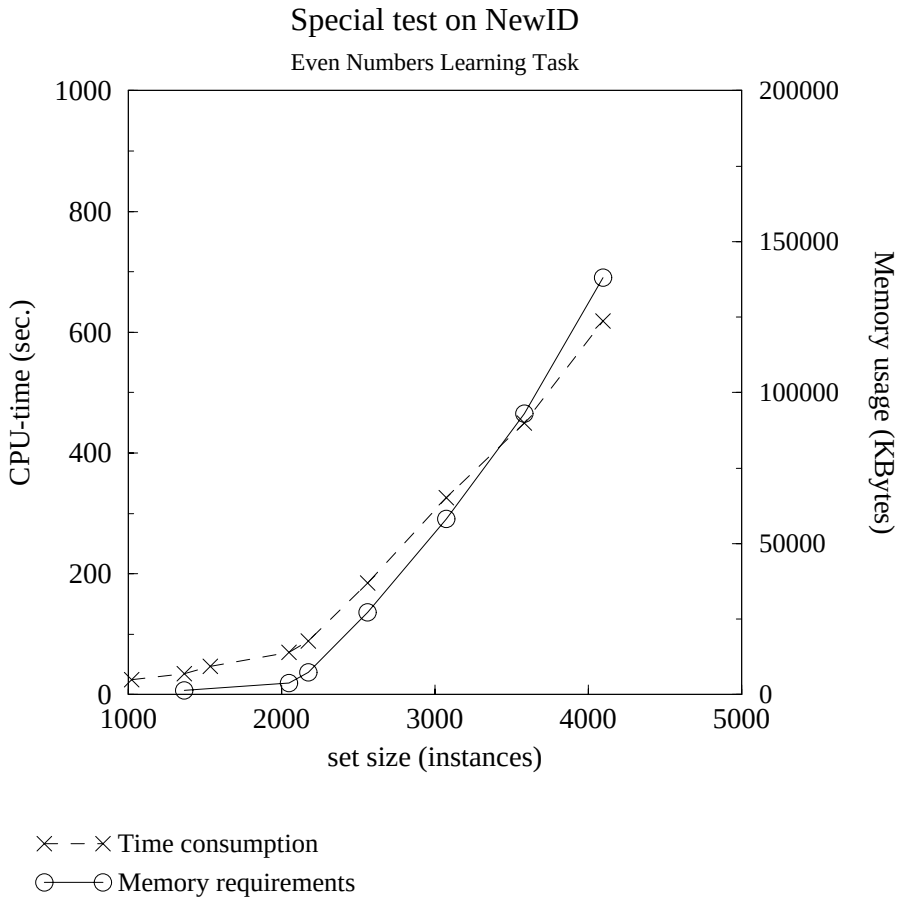

Figure 4.9: *Even-Numbers Learning Task:* Special examination of the NewID algorithm. (corresponds to table C.9)

By subdividing the interval where the sudden fall in the algorithm's performance is encountered, one notices that the problem occurs just above the 2,048 size-step. At this point suddenly the rate at which both CPU-time and memory

space are consumed increases enormously and remains fairly stable after that. In other words the behaviour of the algorithm before and after the critical point is very close to linear, but the difference in the slope of the two parts of the curve is very big. These results reinforce the conclusion drawn above that there is a "physical limit" to the capabilities of the algorithm, which is probably caused by some optimisation method incorporated in the algorithm[17].

Finally, the performance of the generated classifiers is not measured in this experiment, because the concept is not learnable by those systems and their output is of no use. In the case of decision trees (C4.5 and NewID), complete and highly skewed trees are generated, where each leaf node corresponds to a single instance in the training set. Similarly, in the case of the conceptual clusterer (PLS1), each region contains one data point and in the case of decision lists and covers (CN2 and AQ15) each rule (complex) corresponds to one example. In other words, overspecialisation appears in an extreme form, providing no classification power at all.

4.5 Summary

4.5.1 Real Data Sets

Performance Results

The main conclusion drawn on the presented results is that the performance of the algorithms in real data sets differs substantially from the worst-case estimates. The data sets that were used in the experiments are considered computationally demanding for the examined algorithms, because they contain only numeric attributes. Despite that, the behaviour of the algorithms was closer to log-linear rather than quadratic or cubic. This suggests that an average case estimate would be of $O(n)$ or $O(n \log(n))$ order of complexity.

A further conclusion of the above analysis is that the computational overhead of different algorithms can vary substantially. The two extreme cases, in the algorithms that were examined are C4.5 and AQ15, the latter of which becomes prohibitively slow for large data sets. Thus, when examining whether one of those algorithms is applicable on large-scale data, the constant part, contributing to the computational complexity of the algorithm is of equal importance to the order of its complexity.

Another important feature of the algorithms is their behaviour on different

[17]The exact reason for this behaviour is not known.

types of attributes. The most characteristic example of this problem, is the behaviour of C4.5 and NewID in the two experiments. In the first experiment, where integer attributes were used, C4.5 performed better than NewID, in absolute terms, while the order of complexity of the two algorithms was very similar. In the second experiment however C4.5 performed slightly better than NewID on small training sets, but due to its worse order of complexity, it became slower for large sets. This leads to the conclusion that NewID can handle continuous attributes more efficiently than C4.5.

Finally, an observation that dominates the results of the two real-data experiments is that the algorithms which are based on "generalisation" (i.e. AQ15 and CN2) are slower than the "specialisation ones" (i.e. C4.5, NewID and PLS1). In both experiments the performance of the slower of the algorithms falling to the latter category (i.e. PLS1) was comparable to that of the faster of the former category (i.e. CN2). This is mainly due to the complexity of the calculations involved in generalising from seed examples.

Accuracy and Usability Results

The classification performance of the algorithms was not of very much interest to this project. There is a large number of studies comparing the classification performance of similar algorithms to the ones examined here (e.g. [O'Rorke, 1982], [Gams and Lavrac, 1987], [Rendell *et al.*, 1989], [Clark and Boswell, 1991], etc.). What is mainly important here is that the algorithms have achieved similar levels of accuracy, at the same time that they had very different computational behaviour. This means that classification accuracy has not been sacrificed for speed, for the fastest of the algorithms (e.g. C4.5 and NewID). On the contrary, the slowest algorithms have achieved in two situations very low performance, which may be the result of suboptimal parameter-settings.

This last observation brings up an important point: the *usability* of the algorithms. The systems that are related to the AQ algorithm (i.e. AQ15 and CN2) have a major shortcoming, which is the setting of the *maxstar* (maximum star size) parameter. The optimal setting for this parameter depends on the nature of the learning problem and some rough guidelines which have been proposed (e.g. the size of the feature set [Hong *et al.*, 1986]) do not seem to work. Other problems related to the usability of the algorithms are the types of attribute they can handle and the number of values allowed for a numeric attribute. PLS1 for example can handle properly only integer attributes of a small value-range. Similar limitations exist in AQ15, which cannot handle continuous attributes and it only allows for a small range of integer values, which can be increased up to a certain extent, by changes in the source code[18].

[18]The same is true for PLS1.

4.5.2 Artificial Data

The results obtained in the final experiment, which made use of an artificial data set are very different from the other two. The behaviour of the algorithms departs substantially from the linear estimate, approaching quadratic and cubic levels. These results are closer to the worst-case theoretical estimates presented in the previous chapter, confirming their validity.

At the same time, however, a number of unexpected situations have occurred in the last experiment:

- The behaviour of the "generalisation" algorithms was near-quadratic, instead of cubic that was predicted. The exact reason for this is not known, although it seems likely that some optimisation possibility has not been taken into account in the theoretical analysis.
- The "generalisation" algorithms have also a more stable performance than the "specialisation" ones. The result of this is that AQ15 achieves a better result than PLS1 with $1,024$ instances and similarly CN2 does better than C4.5 on $4,096$ examples. This observation reverses the result presented in the real-data set experiments, where the "generalisation" algorithms perform consistently worse than the "specialisation" ones.
- Despite their instability, the behaviour of C4.5 and PLS1 seems to be following some pattern, which is one of increasing complexity, approaching the cubic estimate for large sets. NewID on the other hand exhibits a very strange behaviour, which is close to linear for all the tests, but suddenly becomes very steep around the $2,048$ step. This is probably caused by some limitation in the way in which the algorithm has been implemented.

Chapter 5

Conclusions

5.1 Summary of the Presented Work

This thesis has examined the behaviour of five concept-learning programs with respect to the size of the training set. The programs that were chosen implement popular learning methods (i.e. AQ, decision trees and conceptual clustering) and were designed for use in real-world applications (with the exception of PLS1). These programs are: NewID [Niblett, 1989], C4.5 [Quinlan, 1993], PLS1 [Rendell, 1983b], CN2 [Clark and Niblett, 1989] and AQ15 [Michalski *et al.*, 1986].

The first part of the presented work provided a theoretical analysis of the algorithms, concentrating on their worst-case computational complexity. The results obtained show that the complexity of the three *specialisation* algorithms (i.e. NewID, C4.5, PLS1) is *over-quadratic*, while those performing *generalisation* (i.e. CN2, AQ15) are *cubic*. These results deviate substantially from those previously presented (e.g. [O'Rorke, 1982] and [Rendell *et al.*, 1989]), which provide linear and log-linear estimates for earlier versions of the same algorithms.

The second part of the work is an experimental examination of the scalability of the five algorithms, using real and artificial data sets. Two real data sets have been used, selected from a large number of examined ones (see appendix B), mainly due to their size. The first of these data sets deals with the problem of recognising typed characters and the second is related to the task of chromosome classification. Both contain several thousand examples, allowing a thorough examination of the behaviour of the programs. The artificial data set was designed to provide a near-worst case situation, using the problem of discriminating between odd and even numbers.

5.2 Contribution of the Project

Scalability is an important issue to every program that is applied to real-world problems and may contribute to the fact that ML programs have not been widely used in large-scale applications. The aim of the project was to address the issue of using concept-learning programs with large-scale data, previously neglected in ML research, attempting to answer the question about the applicability for this type of programs.

The first contribution of the presented work was the thorough computational complexity analysis of popular concept-learning algorithms. Several aspects of the design of each algorithm and the way in which they have been implemented in a particular program have been discussed, showing the effect that these factors have on the efficiency of the programs. Particularly interesting was the effect that different types of attributes have on the worst-case complexity estimate. Numeric attributes define an infinitely large search space, limited only by the attribute-values that appear in the training set. As a result, some parameters of the worst-case estimate, assumed to be externally determined, can be shown to be limited only by the size of the training set. An example of such a parameter is the number of nodes in a decision-tree. This has the effect of producing an over-linear (i.e. quadratic or cubic) complexity estimate for the algorithms.

The second interesting aspect of the project was the experimental investigation of the scalability of the algorithms, using large real data sets. Each data set has been divided into a number of *size-steps*, starting from fairly small ones (i.e. $2^6 = 64$ examples) and moving exponentially up to the limits of the set. This approach has proved a convenient way of examining the rate of change in the performance of the programs, without having to examine a large number of intermediate steps, which could prove a very time-consuming process for large training sets.

The results of the experimental analysis using real data sets showed a significant difference from the theoretical estimates. All the algorithms behaved in a near-linear fashion and most were able to adequately handle large training sets. Those which had difficulties coping with large sets did not do so because of their order of complexity, but because of the amount of computation per datum, involved in their learning process. This leads to another important conclusion, namely that the per datum computational cost of an algorithm is relevant to its applicability to real-world problems.

The final part of the project involved the use of an artificial problem in the empirical analysis of the algorithms. The aim of this experiment was to test the validity of the theoretical claims by generating a near-worst case situation. The problem of learning how to distinguish between even and odd numbers, given a sequence of them, has a number of desirable features in that respect:

1. It is a very **common task** in human learning.
2. It contains **no encoded knowledge** in the definition of the feature set, which is where the "strength" of most learning algorithms lies (see [Rendell, 1987]).
3. The distribution of the instances in the feature space is such that orthogonal clustering cannot provide an adequate solution. As a result, **near-worst case** situations arise (i.e. complete and highly skewed decision trees and complete decision lists).
4. Using a single attribute and a binary class causes a **minimum amount of fixed computational costs**, allowing for the investigation of the complexity of the algorithms.
5. It is easily **reproducable.**

The results obtained in this experiment have been very similar to the theoretical estimates, confirming that the worst-case complexity of the algorithms is over-quadratic.

5.3 Further Work

There is a number of issues that this project has not examined, because of the high dimensionality of the scalability problem and the lack of time and resources (e.g. many large data sets). More specifically the following topics are of particular interest:

1. This project has examined only one type of algorithm, namely symbolic concept-learning ones, which perform orthogonal clustering. There are many **different types of algorithms**, which could be analysed in the same way and compared in terms of their scalability. One example of an investigation, using a large range of algorithms, is the Statlog project [Nakhaeizadeh *et al.*, 1993] [1].
2. The theoretical analysis examined only the worst-case performance of the algorithms, while an indication of the average-case behaviour of the algorithms has been gained through the experiments. However, a **theoretical average-case analysis** could add to the understanding of the behaviour of the algorithms, being at the same time more useful for real applications.

[1] However this project does not examine the problem of scalability.

3. Only two real data sets were used in the empirical analysis of the algorithms. The main reason for this is that very large data sets are difficult to acquire. It is expected that using other sets, which include different types of attributes and correspond to problems of variable difficulty, would reveal important aspects of the scaling behaviour of the algorithms. The Statlog project is again an example where **a number of different training sets** have been used.

4. This project has concentrated on one dimension of the scalability problem, namely the size of the training set. However, the scale of the learning problem can be examined from a number of **different perspectives**. Two of them which are particularly interesting are the following:

   The size of the search space.
   The size of the search space is determined by the number and the domain of the attributes used. Both these factors are relevant to the scalability problem. This project has examined only a small part of this issue, by estimating the effect that different types of attributes have on the complexity of the algorithms and examining numeric data sets, which is unusual for symbolic learning research. Further investigation of this topic could produce very intersting results.

   The complexity of the learning task.
   The complexity of the task depends on the distribution of the instances in the feature space. This shows for example the degree of overlapping between different classes and their separability, using different clustering methods. Measuring the complexity of the learning task is not a straightforward issue, but it is expected to have a significant effect on the performance of the algorithms.

5.4 Summary

This project has addressed the issue of scalability of learning algorithms, neglected in ML research so far. It has examined a number of popular algorithms, in a theoretical and an experimental way and has arrived at several conclusions about their behaviour on different scales of data. The presented work could serve as a starting point for a more thorough investigation of the scalability of learning algorithms, leading to the establishment of design principles that will make learning algorithms more applicable to real-world problems, utilising the research that has been done in the field.

Appendix A

Algorithms considered for the project.

This is a list of all the algorithms considered for the project, grouped according to their source:

1. Programs acquired from the *University of Sydney, Australia*:

   - **C4.5**: This is a very recent version of ID3, including many new features (e.g. turning decision-trees into meaningful rule lists, incremental decision-tree building, etc.). The software is being sold together with Quinlan's book *'C4.5: Programs for Machine Learning'*, published by Morgan Kaufmann[1].
   - **FOIL4**: A recent version of Quinlan's FOIL algorithm. This program is publicly available. The ftp address is:

     cluster.cs.su.oz.au: ftp/pub/foil4.sh [129.78.8.1]

2. The *Artificial Intelligence Group* of *George Mason University*, has provided the author with a version of Michalski's **AQ15** program. The correspondent for this program is Eric Bloedorn (*bloedorn@aic.gmu.edu*). This program has also been used in the project.

3. Two versions (one in Lisp and one in Pascal) of the **PLS1** program were acquired from the *University of Illinois at Urbana-Champaign*. Possible correspondents there are: Larry Rendell (*rendell@cs.uiuc.edu*) or Gunnar Blix (*blix@cs.uiuc.edu*). The Pascal version of this algorithm was also used for the project.

[1]The version acquired for the project was kindly provided by J.R. Quinlan.

4. Algorithms, included in the **MLT** (Machine Learning Toolkit) Esprit project. Only a few of those algorithms (CIGOL, NewID, CN2 and MOBAL) were actually acquired[2]. These programs were provided by their developers. The MLT programs are usually more sophisticated systems, which are intented to be used for solving real problems. The project included also an advisor system, that would help the user choose the learning system which suits best to his requirements.

   The MLT included the following learning programs:

   - **APT**: A learning apprentice, developed at ISoft SA, University of Coimbra and LRI (Universite Paris XI).
     (*e-mail: mlt@isoft.fr*)
   - **CIGOL**: An induction system, using first-order predicate logic, developed by Muggleton.
     (*e-mail: Steve.Muggleton@prg.oxford.ac.uk*)
     A more recent version of CIGOL, called **GOLEM**, which is written in C, was also provided by Steve Muggletton.
   - **NewID**: A version of ID3, developed by Tim Niblett, at the Turing Institute.
     (e-mail: *robin@turing.ac.uk*)
   - **CN2**: A version of the AQ algorithm developed also by P. Clark and T. Niblett.
     (*e-mail: robin@turing.ac.uk*)
   - **LASH**: A system based on Michalski's Induce algorithm, developed at British Aerospace PLC.
     (*e-mail: sims@src.bae.co.uk*)
   - **KBG**: A clustering and generalisation tool, based on a similarity measure between examples. This was developed at LRI and ISoft.
     (*e-mail: bisson@lri.lri.fr*)
   - **MOBAL**: A large Knowledge acquisition tool, performing model-learning, developed by GMD.
     (*e-mail: mlt@gmd.de*)
   - **DMP**: A symbolic clustering algorithm, developed by British Aerospace PLC.
     (*e-mail: parsons@src.bae.co.uk*)
   - **SICLA**: A set of algorithms, performing statistical symbolic and numerical data analysis, developed at INRIA.
     (*e-mail: lecheval@icare.inria.fr*)
   - **MAKEY**: Developed also by INRIA.
     (*e-mail: lecheval@icare.inria.fr*)

[2]NewID and CN2 have also been used in the project.

5. Simple Prolog versions of some well-known algorithms. Acquired from GMD in Germany. The ftp address for these is :

   ftp.gmd.de:/gmd/mlt/ML-Program-Library [129.26.8.90]

   The algorithms acquired from GMD are the following:

   - **ARCH**: 2 versions of Winston's Arch.
   - **VS**: A simple version of Mitchell's Version Space algorithm.
   - **AQ-PROLOG**: A prolog version of the Michalski's Aq algorithm.
   - **ID3**: A simple version of Quinlan's ID3 algorithm.
   - **EBG**: A version of Mitchell's EBG.
   - **COBWEB**: A simple version of the Fisher's Cobweb algorithm.
   - **DISCR**: A simple version of Brazdil's algorithm for generating discriminants from derivation trees.
   - **INVERS**: Muggleton and Buntine's Inverse Resolution learning method.
   - **ATTDSC**: A simple learning algorithm, proposed by Bratko.
   - **MULTAGENT**: A simulation of a tutoring system between two agents, by Brazdil.
   - **LOGIC**: Some useful, for learning, logic procedures, implemented by Muggleton.

6. Simple Lisp versions of well-known algorithms. Acquired from the University of Texas (developed by Ray Mooney). The ftp address is:

   cs.utexas.edu:pub/mooney

   The following programs were acquired from the University of Texas:

   - **AQ**: A simple version of the Aq algorithm.
   - **ID3**: A version of the ID3 algorithm.
   - **COBWEB**: A simple version of Cobweb.
   - **FOIL**: A simple implementation of Quinlan's FOIL (also available in Prolog).
   - **PERCEPTRON**: A simple learning system, using perceptron.

Appendix B

Available DataBases

The databases that will be presented here were acquired by the *UCI Repository of Machine Learning Databases* [Murphy and Aha, 1993]. They are only part of this repository and at the end of this section some of the non-acquired interesting databases will be described. The structure of the following description of the databases follows the criteria set in chapter 4 and includes most of the acquired databases.

B.1 Classification according to Size

Some of the databases that have been acquired are considered, by the ML research community, as being large ones:

1. **Letter Recognition Database**
   As described in chapter 4.

2. **Thyroid Disease Databases**
   These databases have a medical subject, namely that of predicting several types of thyroid-diseases. There are 8 databases, each dealing with a different type of thyroid-disease, six of them have 2,800 instances each and the other two 3,163 each. There are also test-sets, which are of substantial size too (972 instances). Their attribute-sets are similar to each other, containing roughly 29 attributes.

3. **Mushroom Database**
   This is a biological database containing 8,124 instances of poisonous and non-poisonous mushrooms. There are 22 attributes describing each mushroom instance, in terms of its appearance.

4. **Heart Disease Databases**
   These are four different databases, which use the same attributes. They have 76 attributes, of which, though, only 14 are considered relevant. In total they have 1,020 instances.
   The problem is to decide whether a patient suffers from a heart-disease. In this respect, there are five possible classes, corresponding to the probability of a patient to suffer from a heart-disease.

5. **Chess: King-Rook vs. King-Pawn endgame**
   This database describes the KR-KP endgame problem, in terms of 3,196 instances. Each instance is a board position defined by 36 attributes. The possible classifications are White-WIN or White-NotWin.

6. **Annealing data**
   The information provided about the subject of this database is not very much, but it contains 798 instances, which can be classified into one of 6 classes, using 38 attributes.

7. **IRAS Low Resolution Spectrometer Database**
   This is part of the data derived from the Infra-Red Astronomy Satellite (IRAS), which attempted to map the full-sky at infra-red wavelengths. It contains 531 instances, each described by 103 attributes. There is also a large number of possible classes (100).

8. **Mechanical Analysis**
   This is about a fault-diagnosis problem ('diagnosis of faults in electro-mechanical devices from vibration measurements'). It contains 209 instances, each of which is described by a different number of components. Each component has 8 attributes. There are 6 possible classifications.

9. **Soybean Disease**
   This is a very-well known database, aimed at predicting the disease from which a soybean plantation suffers. It was used by Michalski and Chilauski to test the AQ11 algorithm in 1980. It contains 307 instances, each described by 35 attributes. There are 19 possible diseases that may be predicted.

Apart from those large databases, there are some data-generating programs, which one can use to generate arbitrary large databases. Some examples of these are the following :

1. **Chess: King-Rook vs. King-Knight endgame**
   This program generates legal board positions for the specific endgame problem. It is based on 13 attributes and the number of instances to be generated is defined by the user, since the potential number of legal states is in the range of millions [Quinlan, 1983a].

2. **Waveform**
   Not much information is provided for the exact subject of this database. It seems that an arbitrary number of waveforms can be generated. There are 3 classes and 21 attributes that can be used.

3. **Led Display**
   This generator has been created by the same people who made the Waveform one. The problem is to identify one of the 10 digits, displayed using LED's. There are 7 attributes (the LED's) and 10 classes (the digits). The number of instances is again non-restricted[1].

Finally there is a number of small and medium-size databases. The small ones are useful in measuring the *sample complexity*[2] of an algorithm. However, since one can always use a small part of a big database to achieve the same outcome, there seems to be no special reason for favouring a small database.

B.2 Attributes and Classes

With respect to the attributes and classes used in a data set, there are two things that are of interest:

- The **type** of the attributes (discrete vs. continuous, structured vs. numeric, etc.).
- Any **missing values** of the attributes in the data set.

Concerning the type of the attributes, what is of most interest to the project is the existence of numeric attributes that makes the learning task substantially more computationally demanding. For this purpose, the following databases would be chosen:

1. **IRAS Low Resolution Spectrometer Database**
   The spectrometer database instances are described by 103 attributes, of which 93 are continuous.

2. **Auto Imports Data**
   This database contains 205 instances of cars, described by 26 attributes. From those, 15 are continuous. Several attributes can be used as "class" attributes.

3. **Echocardiogram Data**
   The problem here is to predict, according to the cardiogram data, whether

[1] In this problem, duplication of instances is a necessity.
[2] That is the number of examples needed to produce a "correct" hypothesis.

a patient will survive for at least one year. There are 132 instances and 13 attributes, of which most are continuous. The class is a combination of two attributes and is boolean-valued.

4. **Iris Plants Database**
   This is another database that has been widely used. The problem is to distinguish between different Iris plants, using information about their sepal and their petals. There are 3 classes, each represented by 50 instances. There are 4 attributes all of which are continuous (width and length).

5. **Labor Negotiations**
   This database contains information about labour agreements. The idea is to distinguish between acceptable and unacceptable contracts. There are 57 instances, each described by 16 continuous attributes.

6. **Waveform**
   All 21 attributes are continuously valued in the range (0..6).

7. **Relative CPU Performance Data**
   The problem set in this database is to estimate the relative performance of a CPU. The database contains 209 instances, which are described by 6 continuous attributes. What is special about this data set is that the class is continuously valued too.

Some of the databases are missing several attribute-values. Examples of those are the following :

1. **Annealing data**
2. **Heart Disease Databases**

B.3 Noise

Although some of the databases are bound to contain noise, no measurement of that is given in the database information. However, there are two database generators that will produce training sets with noise:

1. **Led Display**
2. **Waveform**

B.4 Special types of Learning

The vast majority of the databases is meant to be used with classification systems. However, there are some sets that suit other types either of induction or other learning methods. Some of these are the following:

1. **Pittsburgh Bridges Database**
   This database has been constructed to be used with classification systems, but the actual task is not clearly a classification one. What is asked is to predict a design description, based on specification properties of the bridge. The design is described by 5 properties, which correspond to the "class". There are 108 instances and 7 specification properties.

2. **Scientific Function Finding**
   This database contains 271 cases, each of which is used for finding some scientific rule. Each of the cases contain a variable number of data sets (352 in total), which describe the case.

3. **Logic Theorist**
   This is a Prolog program that learns logic theorems from the first 4 chapters of '*Principia Mathematica*'.

4. **Explanation Based Learning (EBL and PRODIGY)**
   EBL and PRODIGY contain domains to be used with EBL programs. The former contains very few of them, while the second contains several variations of the *blocksworld*, *STRIPS*, *R1*, etc. These have all been used to test PRODIGY.

B.5 Unusual Structure

Some of the databases have unusual features, which may require a modification of the algorithm that is to be tested on them. The most outstanding of those are the following:

1. **Audiology Data**
   This is another medical database, which contains 200 instances, which fall into 24 distinct classes. What is peculiar about this database, is that it does not have a pre-set attribute set. The attribute name is given together with the attribute value at each case.

2. **Pittsburgh Bridges Database**
   Described above.

3. **Relative CPU Performance Data**
   As mentioned above, this database has a continuous class, which is an unusual phenomenon.

4. **Mechanical Analysis**
   The instances of this data set contain a variable number of components. Thus they will probably have to be treated separately.

B.6 Real-world vs. Toy cases

Most of the databases that have been acquired are based on real-world problems. However, the number of instances that they contain, which is the main concern of this project, is relatively small, compared to real-world applications. The major problem categories, which one could identify are the following:

1. **Medical:** There are many medical problems that are being dealt with. For example : Heart-Disease, Echocardiogram and Thyroid-Disease.

2. **Mechanical:** Relative CPU Performance, Mechanical Analysis, etc.

3. **Biological:** Mushroom, Iris Plants.

4. **Politics:** Labor Negotiations, Voting Records.

Apart from those, there are some toy-cases. Toy-cases are usually derived from games and especially in Machine Learning chess is the favourite one. As mentioned above, there are a database and a database-generator for chess, which are accompanied by 6 domain theories, written in Prolog, which generate legal moves.

B.7 Additional Databases

Some interesting databases which exist in the UCI repository and have not been acquired yet are the following:

1. **Credit Screening Databases**
   These are apparently two databases, which have been the product of the knowledge acquisition process for an expert system. They are relatively large and it is claimed that they contain '*a good mix of attributes*'.

2. **Data Generation Program DGP/2**
   This is a sophisticated domain generator which facilitates the formation of a search space with a specific structure. It may be useful for testing special cases that don't arise usually with real data.

Appendix C

Experimental Results.

The results of the experimental analysis of the algorithms are summarised in the following tables.

Size-steps	*Algorithm*				
	C4.5	NewID	PLS1	CN2	AQ15
64	0.52[a](0.04)[b]	1.15 (0.08)	4.23 (0.07)	10.34 (1.02)	266.8(31.8)
128	1.10 (0.02)	2.14 (0.01)	11.41 (0.18)	22.16 (0.49)	497.7(95.3)
256	2.41 (0.29)	4.23 (0.19)	35.56 (1.56)	41.60 (0.67)	1,050 (78)
512	4.78 (0.29)	8.17 (0.25)	86.78 (7.62)	85.65 (7.69)	2,195 (131)
1,024	9.73 (0.96)	16.49 (0.23)	209.04(61.73)	180.07 (8.47)	5,283 (595)
2,048	19.74 (0.74)	35.73 (0.97)	506.96(28.04)	423.33 (8.94)	13,135 (54)
4,096	41.94 (1.67)	92.89 (3.58)	1,352 (17)	1,322 (7)	40,067[c]
8,192	90.58 (1.81)	269.69(19.16)	3,606 (373)	4,654 (23)	114,728
16,384	212.69 (5.92)	892.91(63.21)	8,423 (399)	20,814(1,148)	—[d]

[a] CPU-time consumption in seconds.

[b] Maximum deviation from the mean value.

[c] A single test was carried out for steps 7 and 8, due to the very long time that each test takes.

[d] This test was not carried out, because it is expected to take prohibitively long time.

Table C.1: Scalability Results, using the *Letter Recognition* data set.

Size-steps	*Algorithm*				
	C4.5	NewID	PLS1	CN2	AQ15
128	2.12	1.86	2.70	2.14	1.87
256	2.19	1.98	3.12	1.88	2.11
512	1.98	1.93	2.44	2.06	2.09
1,024	2.04	2.02	2.41	2.10	2.41
2,048	2.03	2.17	2.43	2.35	2.48
4,096	2.12	2.60	2.67	3.12	3.06
8,192	2.16	2.90	2.67	3.52	2.86
16,384	2.35	3.31	2.34	4.47	—
Avg.	**2.12**	**2.35**	**2.60**	**2.70**	**2.41**

Table C.2: *Letter Recognition Set:* The rate of increase of the CPU-time consumed at each size-step.

Size-steps	*Algorithm*				
	C4.5	NewID	PLS1	CN2	AQ15
64	24.67[a] (1.60)[b]	21.67(2.5)	25.10(3.95)	23.80(2.20)	13.67(2.67)
128	32.97 (3.50)	33.33(1.5)	31.67(2.75)	32.73(6.35)	20.67(6.33)
256	42.00 (0.90)	42.67(3.0)	47.43(1.60)	44.80(0.60)	29.00 (7)
512	56.23 (2.75)	54.33(3.0)	60.07(0.90)	56.87(2.05)	42.00 (1)
1,024	64.50 (2.10)	64.00(1.0)	68.23(0.50)	66.07(1.45)	54.00 (0)
2,048	70.87 (1.10)	71.67(0.5)	74.00(0.40)	71.43(0.20)	59.00 (1)
4,096	77.50 (0.15)	78.00(1.0)	79.70(0.15)	79.20(0.90)	65.00[c]
8,192	82.33 (0.25)	82.67(0.5)	84.33(0.30)	84.03(0.20)	69.00
16,384	86.53 (0.80)	87.67(0.5)	88.43(0.70)	88.03(0.75)	—[d]

[a] Percentage of correctly classified instances.

[b] Maximum deviation from the mean value.

[c] A single test was carried out for steps 7 and 8, due to the very long time that each test takes.

[d] This test was not carried out, because it is expected to take prohibitively long time.

Table C.3: Classification Accuracy Results, using the *Letter Recognition* data set.

Size-steps	*Algorithm*		
	C4.5	NewID	CN2
64	1.02(0.10)	1.60(0.19)	13.22 (1.22)
128	2.91(0.06)	3.64(0.05)	28.05 (2.66)
256	6.53(0.29)	7.96(0.28)	58.40 (0.46)
512	16.47(1.79)	17.82(1.48)	130.67 (3.39)
1,024	39.33(2.41)	40.70(3.50)	302.62 (12.39)
2,048	93.47(8.06)	87.30(1.00)	706.76 (49.41)
4,096	228.78(4.79)	194.60(8.31)	1,869(127.14)
5,432	307.24(4.01)	265.95(3.52)	2,594(188.36)

Table C.4: Scalability Results, using the *Chromosome Classification* data set.

Size-steps	*Algorithm*		
	C4.5	NewID	CN2
128	2.85	0.84	2.12
256	2.24	2.19	2.08
512	2.52	2.24	2.24
1,024	2.39	2.28	2.32
2,048	2.38	2.14	2.34
4,096	2.45	2.23	2.64
Avg.	**2.47**	**1.99**	**2.29**

Table C.5: *Chromosome Classification Set:* The rate of increase of the CPU-time consumed at each size-step.

Size-steps	*Algorithm*		
	C4.5	NewID	CN2
64	35.73(2.6)	34.33(7)	27.53(3.6)
128	40.30(2.4)	43.67(5)	38.60(6.5)
256	56.13(3.5)	55.33(5)	49.40(9.0)
512	64.53(2.0)	61.67(2)	59.47(4.9)
1,024	68.50(4.1)	68.33(1)	60.27(9.5)
2,048	73.03(2.5)	74.00(0)	63.30(4.0)
4,096	77.97(0.4)	77.33(2)	63.83(6.9)
5,432	79.03(1.4)	79.00(2)	62.63(9.4)

Table C.6: Classification Accuracy Results, using the *Chromosome Classification* data set.

Size-steps	*Algorithm*				
	C4.5	NewID	PLS1	CN2	AQ15
64	0.21	0.49	1.41	0.23	10.21
128	0.57	1.48	7.50	0.93	36.13
256	1.64	3.89	48.53	3.71	133.16
512	5.21	8.20	346.10	15.63	523.68
1,024	23.66	24.58	2,611	66.53	2,116
2,048	149.39	65.60	—[a]	287.61	8,235
4,096	1,080	619.15	—	1,244	—
8,192	8,610	—	—	—	—

[a]Some of the tests could not be done, because the algorithms could not handle the size of the search space.

Table C.7: Scalability Results, using the *Even-Numbers* learning task.

Size-steps	*Algorithm*				
	C4.5	NewID	PLS1	CN2	AQ15
128	2.71	3.02	5.28	4.04	3.54
256	2.88	2.63	6.47	3.99	3.69
512	3.18	2.11	7.13	4.21	3.93
1,024	4.54	3.00	7.55	4.26	4.04
2,048	6.31	2.67	—	4.32	3.89
4,096	7.23	9.44	—	4.33	—
8,192	7.97	—	—	—	—
Avg.	**4.97**	**3.81**	**6.61**	**4.19**	**3.82**

Table C.8: *Even-Numbers Learning Task:* The rate of increase of the CPU-time consumed at each size-step.

Size-steps	*CPU-time consumption* (sec.)	*memory requirements* (KBytes)
1,367	34.30	1,456
2,048	70.53	3,923
2,176	88.72	7,426
2,562	184.42	27,187
3,074	325.91	58,251
3,586	450.10	93,110
4,096	619.15	137,998

Table C.9: *Even-Numbers Learning Task:* Special examination of the NewID algorithm.

Bibliography

[Boswell, 1993a] **Boswell, R.** A Manual for CN2 v. 6.1. Technical Report TI/P2154/RAB/4/1.5, The Turing Institute, February 1993.

[Boswell, 1993b] **Boswell, R.** A Manual for NewID v. 6.1. Technical Report TI/P2154/RAB/4/2.5, The Turing Institute, February 1993.

[Breiman *et al.*, 1984] **Breiman, L. and Friedman, J.H. and Olshen, R.A. and Stone, C.J.** *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[Clark and Boswell, 1991] **Clark, P. and Boswell, R.** Rule Induction with CN2: some recent improvements. In Y. Kodratoff, editor, *Proceedings of the European Workshop in Learning*, pages 151–163. Springer-Verlag, 1991.

[Clark and Niblett, 1989] **Clark, P. and Niblett, T.** The CN2 Algorithm. *Machine Learning*, 3(4):261–283, 1989.

[Clark, 1989] **Clark, P.** Functional Specification of CN and AQ. Technical Report TI/P2154/PC/4/1.2, The Turing Institute, September 1989.

[de Merckt, 1993] **Thierry Van de Merckt**. Decision Trees in Numerical Attribute Spaces. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 1016–1021, 1993.

[Errington and Graham, 1993a] **Errington, P.A. and Graham, J.** Application of Artificial Neural Networks to Chromosome Classification. *Cytometry*, 14:627–639, 1993.

[Errington and Graham, 1993b] **Errington, P.A. and Graham, J.** Classification of Neural Networks using a Combination of Neural Networks. In *Proceedings of the IEEE Int. Conf. on Neural Networks*, pages 1236–1241, 1993.

[Fisher and McKusick, 1991] **Fisher, D.H. and McKusick, K.B.** An Empirical Comparison of ID3 and Back-propagation. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 788–793, 1991.

[Frey and Slate, 1991] **Frey, P.W. and Slate, D.J.** Letter Recognition Using Holland-Style Adaptive Classifiers. *Machine Learning*, 6:161–182, 1991.

[Gams and Lavrac, 1987] **Gams, M. and Lavrac, N.** Review of Five Empirical Learning Systems Within a Proposed Schemata. In *Proceedings of 2nd European Workshop on Machine Learning*, pages 46–66, 1987.

[Good and Card, 1971] **Good, I.J. and Card, W.I.** The Diagnostic Process with Special Reference to Errors. *Methods of Information in Medicine*, 10(3):176–188, 1971.

[Haussler, 1988] **Haussler, D.** Quantifying Inductive Bias : AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177–221, 1988.

[Haussler, 1992] **Haussler, D.** The Probably Approximately Correct (PAC) and Other Learning Models. In A. Meyrowitz and S. Chipman, editors, *Foundations of Knowledge Acquisition: Machine Learning.* Kluwer Academic, 1992.

[Hong *et al.*, 1986] **Hong, J. and Mozetic, I. and Michalski, R.S.** AQ15: Incremental Learning of Attribute-Based Descriptions from Examples. Technical Report UIUCDCS-F-86-949, Department of Computer Science, University of Illinois at Urbana-Champaign, July 1986.

[Hunt *et al.*, 1966] **Hunt, E.B. and Marin, J. and Stone, P.** *Experiments in Induction.* Academic Press, New York, 1966.

[Kalbfleish, 1979] **Kalbfleish, J.** *Probability and Statistical Inference*, volume 2. Springer-Verlag, New York, 1979.

[Kohonen, 1984] **Kohonen, T.** *Self-Organisation and Associative Memory.* Springer-Verlag, Berlin, 1984.

[Kononenko *et al.*, 1984] **Kononenko, I. and Bratko, I. and Roskar, E.** Experiments in Automatic Learning of Medical Diagnostic Rules. Technical report, Ljubljana, Yugoslavia: E. Kardelj University, Faculty of Electrical Engineering, 1984.

[Michalski *et al.*, 1986] **Michalski, R.S. and Mozetic, I. and Hong, J. and Lavrac, N.** The Multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *AAAI Proceedings*, pages 1041–1045, 1986.

[Michalski, 1983] **Michalski, R.S.** A Theory and Methodology of Inductive Learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–138. Kaufmann, 1983.

[Mingers, 1989] **Mingers, J.** An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[Mitchell, 1977] **Mitchell, T.M.** Version Spaces: a Candidate-Elimination Approach to Learning. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 305–310, 1977.

[Mooney *et al.*, 1991] **Mooney, R. and Shavlik, J. and Towell, G. and Gove, A.** An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 775–780, 1991.

[Murphy and Aha, 1993] **Murphy, P.M. and Aha, D.W.** UCI Repository of machine learning databases. Machine Readable data repository, 1993.

[Murthy *et al.*, 1993] **Murthy, S. and Kasif, S. and Salzberg, S. and Beigel, R.** OC1: Randomised Induction of Oblique Decision Trees. In *AAAI Proceedings*, pages 322–327, 1993.

[Nakhaeizadeh *et al.*, 1993] **Nakhaeizadeh, G. and Henery, R.J. and Richmond, J. and Perdrix, H. and Stender, J.** *Machine Learning, Neural and Statistical Classification.* (in print), 1993. StatLog: Comparative Testing of Statistical and Logical Learning, Deliverable 4.1.

[Niblett and Bratko, 1987] **Niblett, T. and Bratko, I.** Learning Decision Rules in Noisy Domains. In M.A. Bramer, editor, *Research and Development in Expert Systems*, volume 3, pages 25–34. Cambridge: Cambridge University Press, 1987.

[Niblett, 1989] **Niblett, T.** Functional Specification for RealID. Technical Report TI/P2154/TN/4/3.2, The Turing Institute, September 1989.

[O'Rorke, 1982] **O'Rorke, P.** A Comparative Study of Inductive Learning Systems AQ11P and ID3 Using a Chess-End Game Problem. Technical Report ISG 82-2, Computer Science Department, University of Illinois at Urbana-Champaign, 1982.

[Paterson and Niblett, 1982] **Paterson, A. and Niblett, T.** *ACLS manual, Version 1.* Glasgow, Scotland, 1982.

[Quinlan, 1983a] **Quinlan, J.R.** Learning Efficient Classification Procedures and Their Application to Chess End Games. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 463–482. Kaufmann, 1983.

[Quinlan, 1983b] **Quinlan, J.R.** Learning from Noisy Data. In *Proceedings of the International Machine Learning Workshop*, pages 58–64, 1983.

[Quinlan, 1986a] **Quinlan, J.R.** Induction of Decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1986b] **Quinlan, J.R.** The effect of noise in concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 149–166. Kaufmann, 1986.

[Quinlan, 1992] **Quinlan, J.R.** Themes and Issues in Empirical Learning. In *Proceedings of the Sixth Annual Conference of the Japanese Society for Artificial Intelligence*, 1992.

[Quinlan, 1993] **Quinlan, J.R.** *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[Rendell *et al.*, 1989] **Rendell, L. and Cho, H.H. and Seshu, R.** Improving the Design of Similarity-Based Rule-Learning Systems. *International Journal of Expert Systems*, 2:97–133, 1989.

[Rendell, 1983a] **Rendell, L.** A Doubly Layered, Genetic Penetrance Learning System. In *AAAI Proceedings*, pages 343–347, 1983.

[Rendell, 1983b] **Rendell, L.** A New Basis for State-Space Learning Systems and a Successful Implementation. *Artificial Intelligence*, 20(4):369–392, 1983.

[Rendell, 1985] **Rendell, L.** Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 650–658, 1985.

[Rendell, 1986] **Rendell, L.** A General Framework for Induction and a Study of Selective Induction. *Machine Learning*, 1(1):177–226, 1986.

[Rendell, 1987] **Rendell, L.** Conceptual Knowledge Acquisition in Search. In L. Bolc, editor, *Computational Models of Learning*, pages 89–159. Springer Verlag, 1987.

[Rendell, 1990] **Rendell, L.** Induction as Optimization. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):326–338, 1990.

[Rivest, 1987] **Rivest, R.L.** Learning Decision Lists. *Machine Learning*, 2:229–246, 1987.

[Rosenblatt, 1962] **Rosenblatt, F.** *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC, 1962.

[Rumelhart *et al.*, 1986] **Rumelhart, D.E. and Hinton, G.E. and Williams, R.J.** Learning Internal Representations by Error Propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, 1986.

[Schlimmer and Fisher, 1986] **Schlimmer, J.C. and Fisher, D.** A Case Study of Incremental Concept Induction. In *AAAI Proceedings*, pages 496–501, 1986.

[Shortliffe, 1976] **Shortliffe, E.H.** *MYCIN: Computer-Based Medical Consultations*. Elsevier, New York, 1976.

[Utgoff, 1986] **Utgoff, P.E.** Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2. Kaufmann, 1986.

[Utgoff, 1989] **Utgoff, P.E.** Incremental Induction of Decision Trees. *Machine Learning*, 4(2):161–186, 1989.

[Valiant, 1984a] **Valiant, L.G.** A Theory of the Learnable. *ACM Communications*, 27(11):1134–1142, 1984.

[Valiant, 1984b] **Valiant, L.G.** Deductive Learning. *Phil. Trans. of the Royal Soc. London*, A312:441–446, 1984.

[Vapnik and Chervonenkis, 1971] **Vapnik, V.N. and Chervonenkis, A.Ya.** On the uniform convergence of relative ferquencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[Weiss and Kapouleas, 1991] **Weiss, S.M. and Kapouleas, I.** An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 781–787, 1991.

[Weiss and Kulikowski, 1991] **Weiss, S.M. and Kulikowski, C.A.** *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems*. Morgan Kauffmann, Palo Alto, CA, 1991.

[Winston, 1975] **Winston, P.H.** Learning Structural Descriptions from Examples. In P.H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.