Knowledge aquisition for expert systems: inducing modular rules from examples

Thesis

Knowledge Acquisition for Expert Systems: Inducing Modular Rules from Examples

Jadzia Cendrowska
B.A., B.Sc., F.B.C.O.

A thesis submitted for the degree of Doctor of Philosophy
in the
Discipline of Computing
of the
Faculty of Mathematics
of
The Open University
August 1989

The Open University EX12

HIGHER DEGREES OFFICE Higher Degrees Office

07 SEP 1990

LIBRARY AUTHORISATION FORM: .....

Pass to .....

STUDENT: J CENDROWSKA Disposal ~~SERIAL NO:~~ K0042058

DEGREE: PhD

TITLE OF THESIS: KNOWLEDGE ACQUISITION FOR EXPERT SYSTEMS: INDUCING MODULAR RULES FROM EXAMPLES

I confirm that I am willing that my thesis be made available to readers and maybe photocopied, subject to the discretion of the Librarian.

SIGNED: J Cendrowska DATE: 7.9.90

Abstract

Knowledge acquisition for expert systems is notoriously difficult, often demanding an enormous effort on the part of the domain expert, who is essentially expected to spell out everything he knows about the domain. The task is non-trivial and can be time-consuming and tedious. Machine learning research, particularly into automatic rule induction from examples, may provide a way of easing this burden.

Arguably, the most popular and successful rule induction algorithm in general use today is Quinlan's ID3. ID3 induces rules in the form of decision trees. However, the research reported in this thesis identifies some major limitations of a decision tree representation. Decision trees can be incomprehensible, but more importantly, there are rules which cannot be represented by trees. Ideally, induced rules should be modular and should capture the essence of causality, avoiding irrelevance and redundancy.

The information theoretic approach employed in ID3 is examined in detail and some of its weaknesses identified. A new algorithm is developed which, by avoiding these weaknesses, induces rules which are modular rather than decision trees. This algorithm forms the basis of a new rule induction program, PRISM.

Given an ideal training set, PRISM induces a complete and correct set of maximally general rules. The program and its results are described using training sets from two domains, contact lens fitting and a chess endgame. Induction from incomplete training sets is discussed and the performance of PRISM is compared with that of ID3 with particular reference to predictive power.

A series of experiments is described, in which PRISM and ID3 were applied to training sets of different sizes and predictive power calculated. The results show that PRISM generally performs better than ID3 in these two domains, inducing fewer, more general rules, which classify a similar number of instances correctly and significantly fewer incorrectly.

Acknowledgements

In undertaking this research, I have become deeply indebted to Professor Max Bramer for his help and encouragement over the years, for his enthusiasm when rapid progress was being made and his patience when it was not. I thank him.

I should also like to acknowledge the provision of facilities by the Open University, and to thank the staff of the Academic Computing Service for their help with the production of this document.

Part of this thesis has appeared in *International Journal of Man-Machine Studies* Vol. 27, pp. 349—370, 1987.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There has been a rapid increase in the number and variety of expert systems applications over recent years, particularly in commerce and industry. With this increase has come a demand for improved development techniques, and a call for standardization of established techniques, in an attempt to ensure that future systems are increasingly robust and reliable. As the technology matures so confidence in it grows and applications become more ambitious, domains more complex. The task of knowledge acquisition for expert systems is becoming more difficult, and although significant advances in this field are being made on many fronts, problems which were recognized two decades ago still exist.

Machine learning research has played an important role in trying to ease some of the difficulties associated with knowledge acquisition. A variety of approaches has been tried, some with reasonable success. The aim of the project reported in this thesis was to research a small but increasingly important part of this field — that of automatic rule induction from examples.

The thesis begins with a brief introduction to some of the issues involved in expert systems design and development (Chapter 2), including knowledge representation and knowledge acquisition. In section 2.4 it is suggested that machine learning, and in particular rule induction from examples, may hold the key to solving some of the problems of knowledge elicitation. Chapter 3 describes four classic expert systems, DENDRAL, MYCIN, Prospector and XCON, to illustrate the use of decision rules in these systems, and Chapter 4

describes four programs which were 'milestones' in the history of learning from examples.

Some of the most successful rule induction systems in general use are derivatives of Ross Quinlan's ID3 [44,46,47,48]. ID3 was developed in 1978/9 to induce classification rules in the form of decision trees from large sets of examples, and was itself based on a learning algorithm, CLS (Concept Learning System), designed by Earl Hunt in the early 1960s [29]. Chapter 5 describes CLS, ID3 and some of its derivatives and enhancements.

Although the ID3 algorithm is arguably the most popular rule induction system in general use, it expresses its output in the form of a decision tree. The research reported in this thesis identifies some major weaknesses of a decision tree representation (Chapter 6). The incomprehensibility of decision trees has proved to be a significant disadvantage in real-world applications. They are difficult to manipulate — to extract information about any single classification it is necessary to examine the complete tree. This problem can be only partially resolved by trivially converting the tree into a set of individual rules, as the amount of information contained in some of these rules is often more than can be easily assimilated.

More importantly, there are rules which cannot be represented by trees, for example, two or more rules which do not share a common attribute. The consequence of forcing a decision tree representation on such a set of rules is that the individual rules, when extracted from the tree are often too specific, i.e. they reference attributes which are irrelevant. An expert system using a decision tree in these cases frequently demands the results of more tests than are necessary, with possibly serious consequences if the tests are expensive or dangerous to perform. Furthermore, the inclusion of irrelevant attributes may prevent relevant and correct attributes being identified. Ideally, the induced rules should be modular and should capture the essence of causality, i.e. a rule's premise should consist of those features which *cause* a set of instances to be classified in a particular way. Irrelevance and redundancy are potentially misleading and should be avoided. Chapter 6 describes an ideal training set and the sort of rules which should be expected from it, arguing

that if an induction algorithm is to perform well in real-world applications, it must first be known to perform well under ideal conditions. Section 6.5 explains why ID3, designed to induce decision trees, cannot always produce a perfect set of rules even from a training set which is ideal.

Chapter 7 describes how ID3 partitions a training set according to the values of an attribute which is selected using an information theoretic approach. When the tree is being formed, at each node available attributes are tested for expected information gain in the resulting tree if that attribute were selected for partitioning. The attribute which maximizes average information gain is selected. This is repeated until the leaves of the tree are each of a single class. Thus at each node, ID3 searches for the attribute which is most relevant overall, dividing a training set into homogenous subsets without reference to the class of this subset. Section 7.3 describes how this approach can be modified to eliminate redundancy by searching for only relevant values of attributes within subsets of a specified class. A new algorithm is developed which maximizes not *average* information gain but the *actual* amount of information contributed by knowing the value of the attribute to the determination of a *specific* classification, with the result that the induced decision tree is replaced by a set of modular rules.

This algorithm forms the basis of a new rule induction program, which has been called PRISM. Given an ideal training set as described in Chapter 6, PRISM induces a complete and correct set of maximally general rules. It is described in detail in Chapter 8.

The main value of rule induction is that rules induced from incomplete training sets can be used to predict the classification of new instances, i.e. instances not in the original training set. Induction from incomplete training sets is discussed in Chapter 9 which describes a series of experiments performed to assess the predictive power of rules induced using PRISM compared with decision trees induced using ID3. Rules induced from incomplete training sets are prone to errors. The algorithm may fail to induce one or more rules, some rules may be too specific, some rules may be too general, or there may be a combination of errors. Chapter 9 discusses how and

why some of these errors occur. Unlike ID3, the basic algorithm used by PRISM can induce rules which contradict each other. This does not occur in decision trees produced by ID3 because there is always at least one common attribute, e.g. at the root of the tree, whose value is specified in all branches. It is this feature which causes over-specialization in ID3, and in avoiding it, PRISM may induce rules which are not specific enough to discriminate between classes. The basic algorithm has therefore been enhanced to enable it to identify and remove ambiguity by selectively specializing one or more over-general rules. The procedure, described in detail in section 9.4, significantly improves the performance of PRISM.

To compare PRISM and ID3 with reference to predictive power of induced rules and decision trees, a series of experiments was performed, in which a fixed number of instances was selected at random from a complete data set. PRISM and ID3 were applied to these instances and the resulting rules were tested on the full set of instances to calculate the percentage of instances which were classified correctly, the percentage of instances which were classified incorrectly and the percentage of instances which could not be classified. The average number of induced rules (or branches of a decision tree) and the total number of terms comprising these rules were also calculated. This was repeated one hundred times each for ten different sizes of training set and the results averaged for each size.

These experiments and their results are described in Chapter 9 (section 9.6). The experiments were performed for two different types of data, the first in the domain of fitting contact lenses and the second in a chess endgame domain. The results show that the numbers of correctly classified instances is similar for both PRISM and ID3, PRISM performing slightly better in one domain and very slightly worse in the other, but the numbers of incorrectly classified instances differs significantly in both domains. PRISM regularly classifies fewer instances incorrectly than does ID3. Furthermore, ID3's decision trees are in general considerably more specific than rules induced by PRISM, indicating that PRISM has achieved its goal of reducing irrelevance and redundancy. Thus PRISM reduces over-specialization

without sacrificing predictive power; performance is improved because of a reduced likelihood of incorrect classification; and incomprehensibility is reduced because, on average, each rule has fewer terms and is therefore easier to assimilate.

Finally, Chapter 10 suggests how PRISM might be enhanced to deal with attributes with linear values. This and other directions for further research are discussed in Chapter 11. A full listing of PRISM is given in Appendix A. Appendix B gives an example of a static data base and training set, and PRISM's output when applied to this training set is given in Appendix C.

Chapter 2

Issues in expert systems development

2.1 Expert systems design

Feigenbaum [22] describes an expert system as ...

> ...an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. The knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners in that field.

There have been many varied attempts at designing and building such systems (see [8] or [5] for a description of some of them), but the modern day consensus is that an expert system should consist of two basic parts — a knowledge base, which contains all the necessary domain-specific knowledge, and an inference engine, or control structure, which accesses this knowledge to reason about the domain. There should also be a third (subsidiary) part — an explanation program to provide the user, on demand, with an explanation of the line of reasoning. The expert system should also necessarily have a dynamic database, or working memory, to be used for storing information pertinent only to the current application/consultation, and of course, an interface to enable the user to communicate with the program.

Current consensus also indicates that the knowledge base and control structure should be separate, i.e. there should be no domain-dependent pro-

cedures in the control program.

2.1.1 The knowledge base

Domain-specific knowledge can present itself in a variety of ways. The knowledge base may contain:

- facts, i.e. factual statements about the domain; information which is widely known and probably available from textbooks or other such sources, e.g. (from MYCIN) the MORPHOLOGY of E.COLI is ROD. Facts may be stored in numerous ways — as lists, tables, rules, or in semantic networks, etc.

- heuristics or rules of thumb i.e. the knowledge used when reasoning about the domain. This is the knowledge which constitutes 'professional judgement' and may be (and often is) imprecise or uncertain. Heuristics are most often expressed as inference rules, of the form:

  if *premise* then (with some certainty) *action*

  where *premise* is usually a conjunction of conditions describing a situation and *action* is the action to be taken or decision to be made if all the conditions of the *premise* are satisfied. Inference rules are discussed in more detail in section 2.2.1.

- meta-knowledge. This is knowledge about knowledge and is an extremely important part of the knowledge base. It describes the structure of the domain and any relationships between the various concepts. For example, a large part of MYCIN's knowledge base is concerned with contexts, their types and positions in the context tree and with parameters, their types and the contexts which they describe. (See section 3.2.3 for a fuller description of MYCIN's knowledge base.) This part of the knowledge base is often the most difficult for which to find an adequate representation.

META-RULE 001

IF (1) the infection is a pelvic abscess, and
(2) there are rules that mention in their premise Enterobacteriaceae, and
(3) there are rules that mention in their premise gram positive rods,
THEN There is suggestive evidence (.4) that the rules dealing with Enterobacteriaceae should be evoked before those dealing with gram positive rods.

Figure 2.1 One of TEIRESIAS' meta-rules.

- meta-rules. These are rules which act on other (domain) rules. They are generally used for deciding in which order those rules should be fired. TEIRESIAS (see section 2.4.1) makes extensive use of such rules. Figure 2.1 is an example of a meta-rule from TEIRESIAS. Meta-rules are not always present in an expert system's knowledge base. Frequently, rule-ordering information is implicit in the program (see section 2.2.1), or in another part of the knowledge base[1] or even in the language in which the system is written[2].

As expert systems research continues, it is becoming increasingly evident that vast amounts of expert knowledge are going to be necessary for future systems — 'in the knowledge lies the power' [23]. Although perfectly adequate expert systems have been built with surprisingly few rules (e.g. PUFF [31]), the biggest impact on society will be created by systems with many thousands of rules (or the equivalent amount of knowlege if a different representation is chosen). For example, XCON [32], a system of over 6000 rules which is used by Digital Equipment Corporation for configuring computer systems to customers' needs[3], outperformed their best technical

[1] In MYCIN one of the properties of a parameter is a list of the rules which conclude about it; and in most cases these rules will be evoked in the order in which they are listed.
[2] OPS5 has an inbuilt conflict resolution strategy such that if there are two rules, one of which is a specialized version of the other, then the more specialized rule will be selected.
[3] See section 3.4 for a brief description of XCON.

salesmen [38] within two years of being built, and was successful in saving the company a large amount of money.

2.1.2 The control structure

The control structure (or inference engine) is the program which determines how facts and heuristics in a knowledge base should be applied to the problem under consideration. The design of a control structure will depend mostly on the problem solving strategies employed by the domain expert. Also, to a certain extent it may depend on the structure of the domain knowledge, and even sometimes on the language in which the system is to be implemented.

The basic reasoning strategy will generally be either data-driven, in which the knowledge is used to infer as much as possible from known facts about the domain (e.g. DENDRAL and XCON, both described in Chapter 3) or goal-driven, in which a hypothesis is formed and the knowledge is used to try to prove that hypothesis (goal) by iteratively forming and proving sub-goals (e.g. MYCIN and Prospector, also described in Chapter 3).

The various search techniques used are a central theme of AI research and have been widely documented (see, for example, [3], [4], [40] and [58]).

Whichever reasoning strategy and search techniques are used, the control structure must be kept conceptually simple, otherwise it becomes opaque to users, with the result that the system becomes difficult to build, difficult to understand, and consequently, difficult to use.

2.1.3 Explanation

It is generally accepted nowadays, that an expert system should be able to provide the user with some sort of explanation of its reasoning strategy. Many researchers believe that systems without this facility are not likely to be used seriously. Shortliffe [53] states:

> [Explanation] provides the program with a mechanism for justification of decisions; a physician will be more willing to ac-

cept a program's advice if he is able to understand the decision steps that the system has taken. This gives him a basis on which to reject the system's advice if he finds that the program is not able to justify its decisions sufficiently. It thereby helps the program conform to the physician's requirement that a consultation system be a tool and not a dogmatic replacement for the doctor's own decisions.

Rule-based systems which provide explanations all employ similar techniques to do so; namely, they unravel rules which have been used in the reasoning chain from the point at which the request was made. The sequence in which rules are unravelled depends on the inference procedure used.

Suggestions have been made that this level of explanation is not sufficient; that it is too shallow. However, attempts at automatic explanation have already shown that, even to provide this basic level of explanation, much attention has to be paid to the way in which heuristics are represented. A recurring theme is that of modularity of rules. Modular rules are easy to insert, delete and/or modify. Moreover, they represent a 'chunk' of knowledge which is easy to handle and meaningful to experts, and thus, they are useful for explanation purposes. Michalski [33] has proposed a 'comprehensibility postulate'. He states:

> As a practical guide, one can assume that the components of descriptions (single sentences, rules, labels or nodes in a hierarchy, etc.) should be expressions that contain only a few (say, less than five) conditions in a conjunction, few single conditions in a disjunction, at most one level of bracketing, at most one implication, no more than two quantifiers, and no recursion (the exact numbers may be disputed but the principle is clear). Sentences are kept within such limits by substituting names for appropriate subcomponents. Any operators used in descriptions should have a simple intuitive interpretation. Conceptually related sentences are organized into a simple data structure, preferably a shallow hierarchy or a linear list, such as a frame.

RULE085

IF: 1) THE STAIN OF THE ORGANISM IS GRAMNEG, AND
2) THE MORPHOLOGY OF THE ORGANISM IS ROD, AND
3) THE PATIENT IS A COMPROMISED HOST
THEN: THERE IS SUGGESTIVE EVIDENCE (.6) THAT THE IDENTITY OF THE ORGANISM IS PSEUDOMONAS

Figure 2.2 One of MYCIN's rules

Michalski applies this postulate to rules which are induced automatically, but it is equally applicable to rules provided by humans.

2.2 Knowledge representation

There are many ways in which knowledge can be stored. A description of some of the most common representational techniques can be found in [3] and in [2]. The first part of this section is concerned with the structure and use of inference rules[4], which have been the most widely used representational form for expert systems. The second part gives a brief description of semantic networks and frames, both techniques which are often used.

2.2.1 Inference rules

Inference rules have the general form:

if *premise* then (with some certainty) *action*

where *premise* is usually a conjunction of conditions describing a situation and *action* is the action to be taken or decision to be made if all the conditions of the *premise* are satisfied.

Figure 2.2 (from [53]page 75) shows the English translation of a typical MYCIN rule (RULE085) . RULE085 references four attributes (called clinical parameters in MYCIN)[5]: STAIN of the organism (which has a possible

[4]Also called situation $\Longrightarrow$ action rules, if ...then rules, condition $\Longrightarrow$ action rules, decision rules.

[5]Also called descriptors, properties.

value GRAMNEG), MORPHOLOGY of the organism (which has a possible value ROD), patient is a COMPROMISED HOST (which is TRUE or FALSE) and IDENTITY of the organism (which has a possible value PSEUDOMONAS). This rule is applied to a database of facts (MYCIN's dynamic database is described in section 3.2.4). Each condition (or clause) in the premise tests the value of one of the attributes (a fact) in the database. If all conditions are satisfied, then the action part of the rule is activated, and the database is updated accordingly. RULE085 fires when the system is trying to discover the identity of an offending organism. If conditions 1, 2 and 3 are found to be true, then MYCIN concludes that the identity of this organism is Pseudomonas with a certainty factor of .6 (for a discussion of certainty factors see [14]).

Feigenbaum, when discussing expert systems under development at Stanford University [24] says:

> Situation ⇒ action rules are used to represent experts' knowledge in all of the case studies. Always the situation part indicates the specific conditions under which the rule is relevant. The action part can be simple (MYCIN: conclude presence of particular organism; DENDRAL: conclude break of particular bond). Or it can be quite complex (MOLGEN: an experimental procedure). The overriding consideration in making design choices is that the rule form chosen be able to represent clearly and directly what the expert wishes to express about the domain.

Inference rules are a most popular form of representation for expert systems. Firstly, they are modular, i.e. each rule is self-contained, and as such can be altered, inserted into or deleted from the database without affecting any other rule. This allows ease of modification of the knowledge base, a necessity if the knowledge base is to grow and/or change with time. It also allows ease of explanation — again vitally important if the system is to be widely accepted in the community for which it is intended, as explained in section 2.1.3. Secondly, they are easily understood by experts. This also allows for ease of explanation, but furthermore, the user can recognise it

as a 'chunk' of knowledge relevant to the advice being sought. Rules are a natural way of expressing what to do in a particular situation — frequently the sort of information an expert wishes to pass on when explaining how he does his job.

Most early expert systems were rule-based,designed to operate essentially as production systems. A brief description of four such systems (MYCIN, DENDRAL, Prospector and XCON) and a rule-based expert system shell (Xi Plus[6]) is given in Chapter 3.

In all production systems, rules are applied to a database of facts (the dynamic database) to infer new facts, which are then added to the database. The process is iterative, continuing until the user's request has been met (or until all possible rules have been exhausted). DENDRAL and XCON operate by using a data-driven (or forward-chaining, antecedent-driven or bottom-up) mechanism, whereas MYCIN and Prospector use a goal-driven (or backward-chaining, consequent-driven or top-down) mechanism, although both of the latter systems employ a data-driven approach occasionally. Xi Plus can be used in either backward-chaining or forward-chaining mode. To illustrate the two modes of operation, consider the following rule set:

1. if A and B then E

2. if C and D then G

3. if E then F

4. if F and G then H

5. if H then X

6. if D and F then J

7. if A and J then X

8. if D and E then X

Let the database contain the facts A, B and D.

[6]Xi Plus is a trademark of Expertech Ltd.

Forward-chaining

In the forward-chaining mode of operation the premise of each rule is tested against the database of facts and all rules whose premises are satisfied are triggered. If there are two or more such rules, then one of these is selected by what is known as conflict resolution. Several strategies have been used for conflict resolution (see, for example [18]), including:

- Rule order, in which all rules are ordered, and the rule with the highest priority is selected.
- Data order, in which facts in the database are ordered, and that rule is selected which matches facts with highest priority.
- Generality order, in which the most specific rule is selected.
- Rule precedence, in which precedence is determined by a hierarchy or network.
- Recency order, in which that rule is selected which references the most recently matched fact in the database.

Once a rule has been selected, it is fired, thus updating the database. The whole cycle is then repeated until the required information has been deduced.

If, in the above example, we wish to deduce X, the sequence of events is as follows:

1. All rules are tested on the database, and any rule whose premise is satisfied is triggered. In this case, only rule 1 (if A and B then E) is triggered.
2. As there is only one rule, it is fired, and the fact E is added to the database.
3. The remaining rules are tested on the database. This time rule 3 (if E then F) and rule 8 (if D and E then X) are triggered.

4. Assuming that the conflict resolution strategy is to select the most specific rule, rule 8 is selected. Rule 8 concludes about X, so X is added to the database, and the program terminates.

Backward-chaining

In the backward-chaining mode of operation, only those rules which conclude about the required information are retrieved and tested. If their premises are not matched by facts in the database, then rules which conclude about facts specified in these premises are retrieved and tested, and so on until either the premises can be matched or all rules are exhausted. Using the above set of rules to deduce X, the sequence of events is as follows:

1. The goal is to deduce X. All rules which conclude about X, i.e. rule 5 (if H then X), rule 7 (if A and J then X) and rule 8 (if D and E then X), are retrieved and their premises tested on facts in the database.

2. None of the premises is matched, so one of these rules is selected and a subgoal set up. If the conflict resolution strategy this time is rule order, then rule 5 is selected and H becomes the subgoal.

3. The rule which concludes about H (there is only one) is retrieved and tested. This is rule 4 (if F and G then H).

4. Neither F nor G are in the database, therefore F becomes the new subgoal and rule 3 (if E then F) is retrieved.

5. Again, E is not in the database, so this time E becomes the subgoal and rule 1 (if A and B then E) is retrieved and its premise tested.

6. This time both A and B are known, therefore rule 1 is fired and E is added to the database, which in turn enables rule 3 to be fired and F to be added to the database. The database now contains facts A, B, D, E and F.

7. The system is testing rule 4 (if F and G then H). F has been deduced, so now G becomes the new subgoal and rule 2 (if C and D then G) is

retrieved and tested to try to deduce G. D is known but C is not, and as there are no rules which conclude about C, rule 4 fails.

8. Rule 4 was retrieved to conclude about H, which was needed by rule 5 to conclude about X. As rule 4 has failed, rule 5 also fails and the system returns to step 2, and selects the next rule in turn. This is rule 7 (if A and J then X). A is known but J is not, so J becomes the subgoal.

9. There is only one rule which concludes about J, namely rule 6 (if D and F then J). The premise of rule 6 is tested on the database.

10. D and F are both known, therefore rule 6 succeeds and J is added to the database.

11. A and J are now both known, therefore rule 7 fires and X is added to the database and the program terminates.

The choice between a forward-chaining and a backward-chaining control structure will depend to a certain extent on the domain, and on the type of deductions the system is intended to make. If the intention is to deduce as many facts as possible, then the forward-chaining mechanism would be the better one to employ. If, on the other hand, the purpose is to confirm or deny a particular hypothesis, then backward-chaining would probably be the better choice to make.

2.2.2 Other representational techniques

Semantic networks

In some cases, domain-specific knowledge is most readily represented by semantic networks. These generally consist of nodes, which represent objects or concepts in the domain, and arcs, which represent relationships between objects or concepts. The basic functional unit of a semantic network is two nodes linked by an arc. The arc is usually directed to indicate which node is the subject and which is the object of the relation represented by the arc.

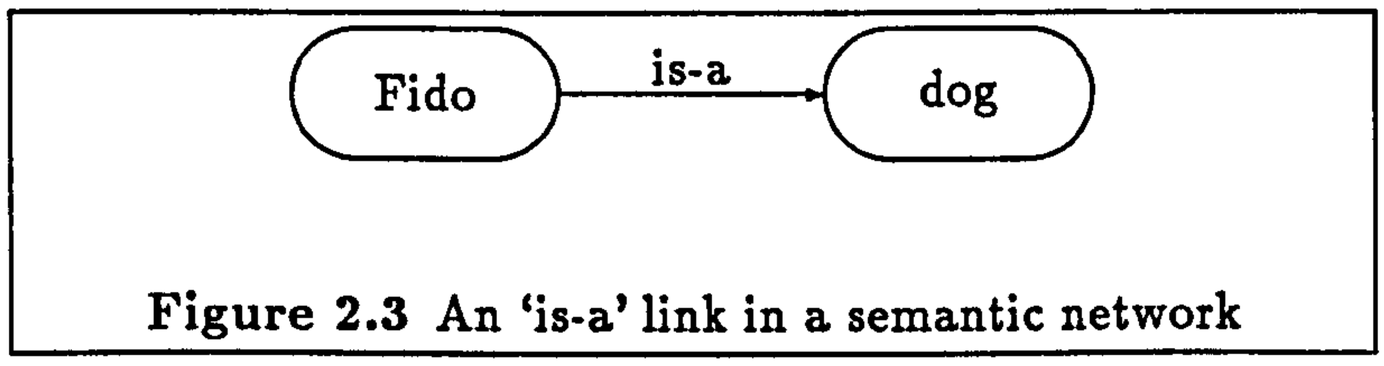


Figure 2.3 An 'is-a' link in a semantic network

For example, the fact that Fido is a dog can be represented by two nodes, 'Fido' and 'dog', linked by an 'is-a' link, as shown in figure 2.3. Each node can have any number of links, and thus quite complex networks can be built. Prospector (see section 3.3) successfully uses such a network when reasoning about the likelihood of certain mineral-ore deposits and advising geologists about the favourability of an exploration site.

Frames

Frames are data structures which comprise the name of a concept, e.g. NOVEL, and either a general or a specific description of it, made up of a number of filled-in 'slots'. Frames representing specific examples are said to be instantiations of general frames and have the same slots, i.e. properties describing a general concept are inherited by specific examples of that concept. Each slot has a name and, in the general case, a description of how it is to be filled when the frame is instantiated. This description may itself be the name of another frame. For example, figure 2.4 shows an example of a frame for the general concept 'NOVEL'. This frame indicates that 'NOVEL' is a specialization of the general concept 'BOOK' and is described by (the slots) title, author, publisher, year and type-of-cover. The slot 'publisher' is filled by a frame 'PUBLISHER' which has its own slots, e.g. name, address. Figure 2.5 shows a possible instantiation of the general frames 'NOVEL' and 'PUBLISHER'.

```
name : NOVEL

        specialization-of : BOOK
        title : title
        author : Surname, First-name
        publisher : PUBLISHER
        year : year
        type-of-cover : hard-back, paperback
```

Figure 2.4 An example of a frame.

```
name : Novel-1

        specialization-of : NOVEL
        title : Paradise Postponed
        author : Mortimer, John
        publisher : pub-1
        year : 1986
        type-of-cover : paperback


name : pub-1

        specialization-of : PUBLISHER
        name : Penguin
        address : Middlesex, England
```

Figure 2.5 Possible instantiation of general frames 'NOVEL' and 'PUBLISHER'

Because there can be several levels in the frame hierarchy, and because each slot can itself be filled by a frame with its own hierarchy, quite a complex data structure can be built. An example of an expert system using a frame-like representation is MOLGEN, which assists biologists in designing experiments in molecular genetics. A description can be found in [56] and [25].

2.3 Knowledge representation in future expert systems

The early classic expert systems (e.g. MYCIN, Prospector, DENDRAL) all used more than one representational form for their respective knowledge bases. This had been found to be necessary mainly because of the complexity of their respective domains. However, the result was that the final systems turned out to be extremely complex. Each had to be hand-crafted over a number of years. It was soon realised that this was a very inefficient way of building expert systems and that some way of speeding up their development had to be found.

Expert system shells began to appear on the market. Most of these used a single fixed representation (usually rules) for domain knowledge, which greatly simplified the development process. However, it was not long before dissatisfaction was being expressed about the limited performance of some of these systems in some domains (e.g. see [7]). The realisation is now spreading that a single representation is not enough, and that future systems will probably need to use different representations for different parts of the domain.

Much recent research has been concerned with methods of representing and using 'deep knowledge' in expert systems. Deep knowledge can be thought of as the knowledge required for a detailed understanding of underlying causal mechanisms of a domain. In order to reason with such knowledge, an expert system will have to have a functional or causal model, and this model will have to be represented somehow. It is unlikely that

inference rules alone will be adequate. Steels [55] states:

> The kernel of a [second generation] expert system consists of two components: A representational component and a problem solving component. ...The representational component is typically frame-based. Information is structured in units with various slots that hold information about the concept described by the unit. This information can be in the form of defaults, rules, procedures to compute information, etc. and is inherited by more specific units. ...In first generation expert systems, the problem solving component consists solely of a collection of rules ...In second generation expert systems there is an additional problem solving component which performs deep reasoning.

Thus it seems that future expert systems will be as complex as (if not more complex than) the original expert systems. However, their knowledge bases will be much more structured, and therefore easier to manipulate, build and understand. Irrespective of the structure of these systems and of the chosen representational forms for deep knowledge, it is likely that most will still have a problem solving component (the heart of the expert system) in the form of inference rules.

With this view of future expert systems the question now arises of how all this knowledge in all its different forms is going to be captured. The next section discusses briefly some past and current techniques of knowledge acquisition.

2.4 Approaches to knowledge acquisition

Expert systems such as the ones described in Chapter 3 owe their success largely to the vast amount of knowledge that they contain. In the early days of expert systems research, it was generally believed that in order for a system to be 'expert', it would be necessary for that system to embody powerful reasoning/search/control techniques. As time passed, it slowly became clear that this was not in fact the case, that it was better to keep

the control strategies simple, that 'the power resides in the knowledge' [23]. Knowledge representation became a major issue — it was necessary to find ways of representing knowledge such that simple control structures could access it efficiently and use it effectively [24]. As expert systems started to be built an even more difficult problem emerged — that of acquiring the vast amounts of expert knowledge required for a system's performance to approach that which had been envisaged. Feigenbaum [23] states:

> [Knowledge acquisition] is the most important of the central problems of Artificial Intelligence research. The reason is simple: to enhance the performance of AI programs, knowledge is power. The power does not reside in the inference procedure. The power resides in the specific knowledge of the problem domain. The most powerful systems will be those which contain the most knowledge.

The problem of acquiring this knowledge is one which has been well documented [6,9,10,23,26,37,53,57,60]. The task of collecting and synthesizing all relevant knowledge has generally required many hours to be spent in consultation with a domain expert. Elicitation is an extremely laborious process, demanding enormous effort on the part of the expert, who is essentially expected to spell out everything he knows about the domain. The task is particularly daunting if the expert has difficulty in articulating what he knows or expressing it in a format suitable for coding. However, this should not be surprising. Welbank [57] points out:

> There is no logical reason why it should be necessary to be aware of how a thing is done in order to do it. The impression given by those psychological experiments that have addressed the question is that people are, in general, not very aware of their own reasoning. Furthermore, the more expert they are, the less they are aware of it. As reasoning becomes more practised and faster, it sinks out of consciousness [21].

2.4.1 Early approaches

The systems described in Chapter 3 were hand-crafted over many years. The acquisition problem became clear early on in the work on DENDRAL. By 1969 it was apparent that something would have to be done to speed up the process of knowledge acquisition. Work on Meta-DENDRAL began.

Meta-DENDRAL [11] is a program which automatically infers rules of mass spectroscopy (i.e. fragmentation or cleavage rules). Like DENDRAL, it is data-driven and uses a plan-generate-test strategy. It comprises three programs — INTSUM, RULEGEN and RULEMOD. INTSUM takes as input a molecular structure and its associated spectrum and produces a set of very specific cleavage rules. It uses what has been called a 'half-order theory' of mass spectroscopy to simulate the bombarding of the molecular structure and then compares the simulated spectrum and original spectrum to infer the causes of the observed peaks. RULEGEN then generates a set of plausible rules, i.e. rules for which positive evidence has been provided by INTSUM, and finally, RULEMOD tests and refines these rules, specializing, generalizing, merging rules or removing redundancies, as necessary. Meta-DENDRAL is described in more detail in section 4.2.

While work on DENDRAL and Meta-DENDRAL was in progress, work on another major expert system project started — MYCIN. Again, it became clear very early on that knowledge acquisition was going to be a problem. The difficulty was somewhat different, though. With DENDRAL the major problem had been that for certain families of molecules, there simply were no experts whose knowledge was broad enough to provide the required cleavage rules. With MYCIN it was discovered that the experts had difficulty in articulating their knowledge to the required level of detail and the best way of eliciting this knowledge was to observe the experts while they worked their way through some difficult cases, asking pertinent questions as and when necessary. Davis [17] found that he was able to automate this procedure to a great extent and developed an interactive program, TEIRESIAS, to help with the knowledge elicitation problems which were clearly hindering the development of MYCIN (and other MYCIN-like systems). TEIRESIAS

is brought into use by an expert to help debug a rule base. The procedure can be divided into three stages:

1. Run a consultation with the current knowledge base and find a diagnosis with which the expert does not agree.

2. Run TEIRESIAS' explanation program to identify the faulty or missing rule.

3. Modify, delete or add a rule as necessary. TEIRESIAS prompts for clauses, diagnoses, etc.

TEIRESIAS uses meta-rules to aid the debugging process. These rules contain knowledge about MYCIN's reasoning strategy and about the knowledge contained in its standard rules. It also uses 'rule models' to help the expert formulate new rules. These rule models contain information about what any particular rule should look like, e.g. what parameters should be contained in the premise of a rule concluding about the identity of an organism. Thus TEIRESIAS can identify and prompt for a missing clause in a rule.

2.4.2 More recent approaches

Prospector and XCON both 'evolved', i.e. building the knowledge bases was an iterative process, in which the first step involved an expert learning a little about the system, and the system builder learning a little about the domain of expertise — enough to construct a rudimentary version of the system. Once the system was working — no matter how badly — it was relatively simple (although time-consuming and tedious) to iteratively run, debug and modify it until its performance was acceptable.

This method of knowledge elicitation is the one which is most frequently used today. Welbank [57] divides the procedure into three stages:

1. Eliciting the domain structure and terminology; defining the important concepts; describing the attributes.

2. Getting enough knowledge to construct the initial working system.

3. Refining the knowledge base, i.e. testing and debugging.

She states that although the stages may not be clearly separate, each stage does have its own problems associated with it. Some of the problems identified with the first stage (and to a certain extent the second stage) are:

- the expert may be inaccessible
- the expert may be unenthusiastic
- there may be a lack of communication
- the expert may be inarticulate
- the expert may be totally unaware

These problems can be overcome by choosing appropriate elicitation techniques. Questions must be of the right sort. They must be asked the right way. They must be specific. Different types of knowledge may require different elicitation techniques.

Interviews

Interviewing is the most popular method of knowledge elicitation. Unfortunately, it is a lengthy and tedious process, more suited to eliciting knowledge about the basic domain structure and concepts, than acquiring the fine details necessary for high performance. This remains so despite the many and varied interview and questioning techniques which have been developed (see [57] for a brief discussion of some of these).

Protocol and task analysis

These methods involve watching the expert while he works through a problem to its solution. This may provide more detailed information than straightforward interviewing, but has the disadvantage that it may not cover unusual cases.

Debugging

With many of the early expert systems, it became clear that experts were far more adept at debugging a faulty set of rules than formalizing new ones. Many system builders found that it was worthwhile trying to build a prototype as early on as possible. Once the system was running and making mistakes, it was far easier to elicit information in the context of these mistakes. This was the basis on which TEIRESIAS was built. It is also the method recommended for building knowledge bases for Xi Plus and other modern expert system shells.

Automatic rule induction from examples

All of the above knowledge elicitation techniques are open-ended, i.e. the only way in which gaps or errors in the knowledge are discovered is if the system makes a mistake. Furthermore, as the knowledge bases grow, it is often difficult to keep a check on rule interaction and consistency.

Automatic induction techniques may hold the key to some of these problems. Programs which accept as input a number of examples of concepts or decisions and produce a set of inference rules explaining these concepts or decisions have a distinct number of advantages over traditional knowledge acquisition techniques:

- experts find it easier to give examples than to formalize rules
- elicitation can be accomplished in a fraction of the time
- gaps in the knowledge can (often) be more easily identified
- errors can be more easily identified and remedied by the use of counter-examples

However, automatic rule induction also has a number of disadvantages:

- only rules can be elicited
- knowledge representation is uniform, therefore some of the domain knowledge may have to be forced into an unnatural representation

- the expert still needs to produce the original structure of the domain and definition of concepts and attributes

- (at the present time) usually only one type of rule can be induced at any one time, i.e. rules which describe an object or decision in terms of relationships between attributes e.g. *if* $A > B$ *then* ...or in terms of structural descriptions e.g. *if A is on top of B then* ...cannot be mixed with rules which describe an object or decision in terms of attribute descriptions e.g. *if attribute A = blue and attribute B = square* ....

If, as envisaged, the expert systems of the future are going to be multi-representational, it seems that knowledge acquisition techniques will have to be multi-faceted, with different techniques being used for different types of knowledge.

Despite current shortcomings, automatic rule induction from examples holds the promise of enabling a great amount of knowledge to be acquired in a very short time. This was recognised many years ago, and research in this area (under the umbrella of the broader topic of machine learning) has been active for over two decades. An overview of machine learning is given in Chapters 3 and 4.

Chapter 3

A Review of some Classic Expert Systems

This chapter describes the structures, representational techniques and control mechanisms used in four 'classic' expert systems — DENDRAL, MYCIN, Prospector and XCON. It is included in this thesis to illustrate the different forms and uses of inference rules. The chapter also includes a description of Xi Plus, as an example of a modern, commercially available expert system shell.

3.1 Heuristic DENDRAL

Heuristic DENDRAL [11] is a set of programs which were designed for use by organic chemists to infer plausible molecular structures for unknown organic compounds from their chemical formulae, mass spectroscopic and other data. It was developed by Joshua Lederberg, Bruce Buchanan, Edward Feigenbaum and others from about 1965 onwards as a collaborative project between Stanford University and the Stanford Mass Spectroscopy Laboratory.

The basic inference procedure of Heuristic DENDRAL comprises a sequence of three steps; namely, plan, generate and test.

If the spectrum for the molecule has two peaks
at masses x_1 and x_2 such that
a. $x_1 + x_2 = M + 28$, and
b. x_1 - 28 is a high peak, and
c. x_2 - 28 is a high peak, and
d. at least one of x_1 or x_2 is high,
Then the molecule contains a ketone group.
(M is the molecular weight which is inferred from
the chemical formula.)

Figure 3.1 One of DENDRAL's rules.

3.1.1 The Planning Program

DENDRAL's planning program takes as input the chemical formula and mass spectrum of the compound to be analyzed, and returns two lists:

goodlist — a list of molecular fragments that must be in the final molecular structure, and

badlist — a list of molecular fragments that must not appear in the final molecular structure.

These lists are used in the 'generate' stage as constraints to limit the number of plausible structures generated. Their construction is enabled by the use of a great deal of judgemental knowledge, which is encoded as production rules. Figure 3.1 (from [4]page 107) shows an example of such a rule.

The planning program uses a forward-chaining mechanism to infer as much as possible from the given facts, and thus makes **goodlist** and **badlist** as complete as possible. This will increase the constraints placed on the generator, which in turn will infer fewer plausible structures to be passed to the test program.

3.1.2 The 'Generate' program

When the planning program comes to an end, the lists **goodlist** and **badlist** are passed to the 'generate' program. This program is known as CONGEN

If N—C—C—C Then N—C * C—C

Figure 3.2 One of DENDRAL's fragmentation rules

and is responsible for systematically generating all the possible molecular structures for the unknown compound. **Goodlist** and **badlist** together with other constraints, which can be input directly, are used to limit the generator to enumerate only plausible structures, thus drastically limiting the number of structures generated.

The structures are generated in stages. A small part of a molecule will be generated first, and then new atoms or molecule fragments added in all possible configurations. The molecule structures thus 'grow'. Several constraints should be added at each stage, otherwise the combinatorial explosion in molecule structures becomes quite prohibitive.

CONGEN can be used (and often is used) on its own. It has been proved mathematically to produce an exhaustive and non-redundant list of legal candidate structures, and is unrivalled by human performance [24].

3.1.3 The Testing and Ranking programs

Heuristic DENDRAL's final stage is the testing and ranking of the candidate structures generated by CONGEN. This is done by two programs, MSPRUNE and MSRANK.

First, for each possible structure, MSPRUNE generates a hypothetical mass spectrum using a fairly simple model of mass spectrometry, which is encoded as a set of production rules. Figure 3.2 shows an example of such a rule. Fragmentation rules such as this indicate expected peaks in a mass spectrum. MSPRUNE uses these rules to build a hypothetical mass spectrum for the candidate structure. It then compares this with the original mass spectrum and if the two are not similar, the candidate structure is removed from the list of possible structures. Every candidate structure

generated by CONGEN is tested in this way.

Finally, all remaining plausible structures are ranked by MSRANK according to the number of predicted peaks found (or not found) in the original mass spectrum. This process also involves the use of detailed knowledge of cleavage and migration laws which are encoded as production rules.

3.2 MYCIN

MYCIN [53], [14] is a rule-based expert system developed by Edward Shortliffe and others at the Stanford Heuristic Programming Project, in collaboration with the Infectious Diseases Group at the Stanford Medical School. It was designed to assist physicians in the diagnosis and treatment of diseases caused by certain kinds of bacterial infection. Work on the MYCIN project started in 1972. The task was to design a system which could play a similar role to that of a human specialist in infectious diseases. Thus, it was to interact with a physician to collect all the relevant information available about a patient under consideration, and then to examine this information for evidence upon which to base a diagnosis and recommendation for therapy.

The entire MYCIN system comprises three subprograms: the consultation program, the explanation program and the rule-acquisition program. It stores its information in two databases: a static database which contains all the rules used during a consultation, and a dynamic database which is created afresh for each consultation and contains patient information and details of any questions asked in the consultation.

3.2.1 The structure of the domain

As a consultation proceeds, MYCIN builds up information about a number of entities in its domain, such as an offending organism or the culture from which it was isolated. These entites are known as contexts. The information is either provided directly by the user or deduced using rules.

There are a number of context types employed, e.g. PERSON, CURORGS.

Contexts are arranged hierarchically in a tree structure known as a context tree, which varies in detail from one consultation to another. In every consultation there is exactly one context of the type PERSON (i.e. the patient himself or herself). This context has no 'parent context' and serves as the root node of the context tree. Contexts of every other type can occur as many times as necessary (including zero times). Every context type except PERSON has a corresponding 'parent' context type, i.e. a CURCULS context can only be a direct descendant of the PERSON, a CURORGS context can only be a direct descendant of a CURCULS context, and so on.

3.2.2 Inference rules

MYCIN uses inference rules to embody its expert knowledge about infectious diseases, as inference rules of the general form

if *premise* then *action*

where *premise* generally involves testing the value of one or more clinical parameters (e.g. SITE — the site of the culture) and *action* generally involves concluding the value of one or more further parameters.

For many of its clinical parameters, MYCIN usually computes not one definitive value but a number of alternative possibilities each with its own probability-like value called a certainty factor. This is a number between -1 and +1 and is used to indicate the degree of belief that the value of the clinical parameter is the true value.

Each of MYCIN's rules is intended to correspond to an item of knowledge meaningful to the physician. Each rule has both an internal (stored) form and an external English translation. In the internal form, both the premise and the action part of the rule are held as a (LISP) list structure. Figure 2.2 on page 19 is an example of one of MYCIN's rules.

3.2.3 The static database

MYCIN has a static database which contains its production rules and all the other fixed information needed by the consultation program. Every

context type and every clinical parameter used by MYCIN has a number of properties which fully describe it. These properties enable the program to make all the correct associations between parameters and contexts and between the contexts themselves, and provide information which tells it which rules to invoke and when to invoke them, when to ask a question, which question to ask and what answers to expect. They also enable MYCIN to find the right position in the context tree for a particular context, and indicate which basic questions to ask when a context is first instantiated.

The static database is set up only once, when the system is being built, but it can be modified by experts using the Rule-Acquisition Program[1].

3.2.4 The dynamic database

MYCIN makes use of a dynamic database which is set up afresh for each consultation. This contains data of the kinds described below. In each case the data can be thought of as taking the form of object-attribute-value triples:

1. patient data, i.e. the values of clinical parameters (as supplied by the physician or inferred by the program);

2. so-called dynamic data, which records the details of acquisition of data mainly for explanation purposes;

3. properties of context types used when instantiating contexts;

4. information about the context tree as it is built up.

3.2.5 The control structure

When the physician feels that he needs advice about the management of a particular patient, he enters the system by starting the Consultation Program. The ensuing interaction is the core of the program, during which the system asks the necessary questions, draws its inferences and makes its recommendations as to the diagnosis and therapy. A typical consultation

[1]For a description of the Rule-Acquisition Program see [17] or [53].

lasts about 20 minutes. The questions asked depend upon answers previously given, and are only asked if the information cannot be inferred from data already acquired. They are asked in a logical sequence so that the physician can follow the course of the consultation. Redundancy is avoided. If the physician feels that some part of the consultation is obscure, e.g. if he cannot see the reason for a particular question, he can temporarily adjourn from the consultation to ask for clarification. The system can then explain the reason for the question and give examples of the type of answer it expects. Afterwards, the physician can return to the basic consultation without having to retrace his steps from the beginning.

MYCIN's control structure is (principally) a goal-directed backward-chaining of rules. At any point, the program is working towards the goal of finding the value of some parameter of a context, i.e. tracing the parameter, and it does this by invoking all the rules which make a conclusion about that parameter in their *action* part. This leads to a depth-first search of an implicit AND/OR tree formed by the constituent conditions of the rules. At the leaf nodes of this tree, the values of parameters are provided by the user in response to questions.

The aim of MYCIN's backward-chaining approach is to avoid asking questions unnecessarily. Instead, with few exceptions, questions are asked only when needed to trace the value of a clinical parameter.

The consultation starts by instantiating the patient context as the root node of the context tree and then attempts to find the value of the REGIMEN parameter for that context. There is a single relevant rule, called the goal rule, which leads to a deduction of the value of REGIMEN. In order to find the value of REGIMEN, MYCIN traces the values of the parameters referenced in the *premise* part of the goal rule. The physician probably does not know these values, so the rules which infer these values have to be invoked, and the parameters in their *premise* parts traced, and so on until the physician can supply some answers.

When the consultation is over, the system passes automatically to the second subprogram, the Explanation Program, which answers questions

from the user and explains its line of reasoning. It does this by showing an English version of the rules used, to explain why it needed a particular piece of information, and how certain conclusions were reached. The main purpose of this is to allow the physician to decide if MYCIN's reasoning is sound, and to reject its advice if he feels that it is not.

3.3 Prospector

Prospector [19] was developed by R. Duda, J. Gaschnig, P. Hart and others at SRI International in California, in collaboration with a number of economic geologists and the U.S. Geological Survey. It was designed to assist field geologists in evaluating possible exploration sites for the existence of certain ore deposits. Like MYCIN, it is an interactive consultation program which allows the user to interrupt and question it.

The consultation starts with the user giving the program some basic information about rock types and minerals which have been observed at the site in question. The program uses this information to form some tentative hypotheses about ore deposits at the site. It then uses its stored geological knowledge, in a goal-directed fashion, to try to either prove or refute each of its hypotheses, asking for further information from the user if necessary. The eventual outcome of the consultation is a numerical indication that a certain type of ore deposit exists at the site (if indeed it does exist), together with a list of favourable factors supporting the hypothesis and a list of possible unfavourable factors (if there are any).

3.3.1 The structure of the domain

The core of Prospector's knowledge base consists of a number of computational 'models', each of which is designed to represent a body of knowledge about a particular class of ore deposit. The knowledge base is completely independent of Prospector's control mechanism. Each model is encoded as a separate inference network of facts and hypotheses, known as assertions. These assertions are the nodes in the network and are connected together

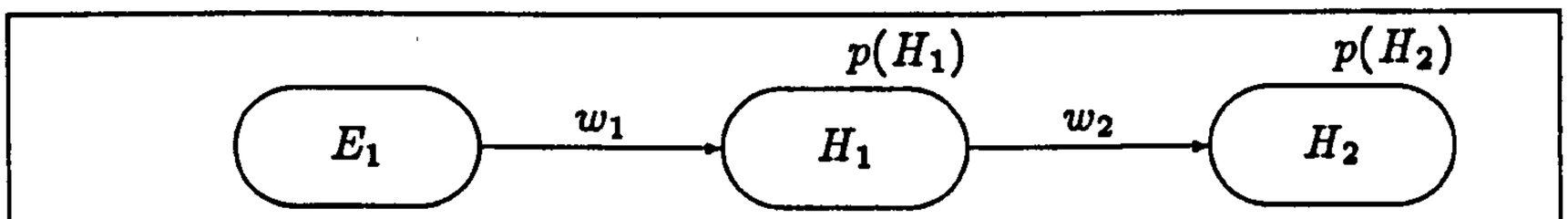


Figure 3.3 Nodes and relations in a Prospector-like inference network

by arcs or relations. Each assertion has a probability-like value associated with it, indicating the degree of belief in it. Each relation between two assertions is also quantified, i.e. a numerical value indicates to what degree one assertion affects the other. Thus, the network can be thought of as a set of interlinking inference rules, each rule consisting of two nodes joined by a relation. For example, if observation E_1 implies hypothesis H_1, which in turn implies hypothesis H_2, then this would appear in an inference network as shown in figure 3.3, in which w_1 and w_2 are weights assigned to the rules indicating strength of implication and $p(H_1)$ and $p(H_2)$ are the prior probabilities of H_1 and H_2 respectively. Leaf nodes in the network correspond to field evidence supplied by the user. Thus E_1 in figure 3.3 would be a leaf node. When the user supplies this piece of evidence, he must also input his degree of belief in its existence. This is expressed as a number between -5 and $+5$ (where $+5$ indicates that the evidence is definitely present and -5 indicates that it is definitely absent), and is converted by the program into a probability-like value.

Each Prospector model also has a semantic network which defines the logical relations between various entities. This network includes a large taxonomy of minerals which allows facts such as 'Pyrite is a Sulphide' and 'Sulphide is a Mineral' to be directly represented. Thus if the program needs to know if sulphide is present, and the user volunteers the information 'Pyrite is present', the program can make the relevant deductions automatically.

The semantic network is partitioned into higher level spaces which form the nodes of the inference network. Thus each assertion in the inference network has its own structure which is explicitly described.

There are three types of relation between assertions in Prospector: plau-

sible, logical and contextual.

- **Plausible Relations.** Each inference rule has an associated weighting factor which indicates how a change in the probability of one assertion affects the probability of the other. The weighting factor comprises two numbers, LS and LN. LS is called the 'sufficiency factor' and is used if the evidence in question is observed to be present. LN is called the 'necessity factor' and is used if the evidence in question has been proved to be absent. Frequently, the user may not be certain about the evidence, in which case an interpolation formula is used to update the probability of the hypothesis. LS and LN are derived from Bayes' Theory.

- **Logical Relations.** There are three logical relations: AND, OR and NOT. A hypothesis may be defined as the logical conjunction (AND) of several pieces of evidence, or it may be the logical disjunction (OR) of two or more assertions, or it may simply be the logical negation (NOT) of an assertion. Fuzzy set theory is used to compute the truth value of the hypothesis from the truth value of its component assertions, when these are not known with certainty.

- **Contextual Relations.** These are used when an assertion cannot be used in the reasoning process until another assertion has been determined, or a piece of evidence has been made available.

3.3.2 The control structure

A consultation with Prospector begins with the user giving the program some basic information about rock types and minerals which have been observed at the site in question. The information does not have to be definite. The user can use phrases such as *'There may be ...'* or *'It is unlikely that ...'*, and the program will translate these into probability-like values. The user can also express his degree of belief numerically using a scale from -5 to $+5$ where $+5$ means 'definitely' and -5 means 'definitely not'.

The volunteered information is matched against the models, any relevant probabilities are updated and the change is propagated through the inference network via the inference rules. When the user has finished volunteering information, Prospector examines each of its models and selects one as the current hypothesis. The selection is made on the basis of the number and types of connections between the model and the given information, i.e. the best-matching model is chosen. Prospector's control strategy now becomes goal-driven, or backward-chaining. The initial goal corresponds to the selected model. The program tries to establish the assertion that the given field evidence matches the model. It does this by trying to establish a number of hypotheses which support this assertion. These hypotheses in turn may require third-level assertions to be determined. The program will chain in this way until assertions can be established directly, i.e. asked of the user. When all the evidence has been collected, the program is able to assign a truth value to the top-level assertion. If, at any time, the field evidence gathered by Prospector indicates that the top-level goal is unlikely, then the model is discarded and a new one selected.

At any time, the user may interrupt the consultation to ask for an explanation, or to volunteer new information, or to change a previous statement. Occasionally, the program asks the user if he wishes a particular line of reasoning to be pursued, or if he wishes to discount any particular hypothesis.

The final outcome is a numerical indication (between -5 and $+5$) of the degree of belief that a particular ore deposit is present at a given site.

3.4 XCON

XCON (formerly R1) [32] is a rule-based expert system which was designed to configure Digital Equipment Corporation VAX-11/780 computer systems. It was developed by John McDermott at Carnegie-Mellon University from about 1978 onwards, and is now thought to be arguably the most successful expert system ever, in that it is saving the company in the region of $40 million per year.

XCON's task is to take a customer's order, check that it is complete and correct, configure the components which make up the order in some suitable and satisfactory way, and output a diagram of the computed configuration.

3.4.1 The knowledge base

XCON is implemented as a production system using the production system language OPS5. The knowledge base comprises a production memory (or rule base) and a database of component descriptions and is equivalent to MYCIN's static database.

The rule base contains approximately 6000 if *premise* then *action* rules. As usual, the *premise* is generally a conjunction of conditions which are to be matched by elements in the working memory, and *action* is the action to be taken if the conditions are matched and normally involves adding, deleting or changing an element in working memory. There are three types of rules:

- Sequencing rules. These rules are responsible for dividing up the task into sub-tasks and for determining the order in which these sub-tasks have to be tackled.

- Operator rules. As well as checking that all necessary components are available and are of the correct type, the operator rules are responsible for determining optimal configurations or part-configurations. Figure 3.4 is an example of an XCON operator rule (from [32]).

- Information-gathering rules. These rules access the database or perform appropriate calculations to provide sequencing and operator rules with relevant information.

The configuration task is divided into a (large) number of sub-tasks, which are called contexts, e.g. ASSIGN-POWER-SUPPLY or CHECK-FOR-MISSING-ESSENTIAL-COMPONENTS. Each context has a fairly small number of rules associated with it, and the first condition in the *premise*

```
ASSIGN-POWER-SUPPLY-7

IF:     THE MOST CURRENT ACTIVE CONTEXT IS ASSIGNING
        A POWER SUPPLY
AND     A UNIBUS ADAPTOR HAS BEEN PUT IN A CABINET
AND     THE POSITION IT OCCUPIES IN THE CABINET (ITS
        NEXUS) IS KNOWN
AND     THERE IS SPACE AVAILABLE IN THE CABINET FOR A
        POWER SUPPLY FOR THAT NEXUS
AND     THERE IS AN AVAILABLE POWER SUPPLY
AND     THERE IS AN H7101 REGULATOR AVAILABLE
THEN:   PUT THE POWER SUPPLY AND THE REGULATOR IN THE
        CABINET IN THE AVAILABLE SPACE.
```

Figure 3.4 One of XCON's rules

of every domain-specific rule indicates the context to which the rule belongs. The rule shown in figure 3.4 belongs to the context ASSIGN-POWER-SUPPLY, and will not fire unless this is the most current active context.

XCON's database contains descriptions of components which are necessary for the configuration task. At the present time it knows about approximately 20,000 such components. Each entry in the database includes the name of the component and a description of its properties, stated as attribute-value pairs. Every component has a 'type' attribute (e.g. disk drive) and a 'class' attribute (e.g. bundle, backplane). On average, each component description comprises eight attribute-value pairs. XCON's database also contains cabinet templates, which describe the space available in each cabinet type. Cabinet templates are used by XCON to keep track of the availability of cabinet space, and to enable it to assign a specific location to each component placed in the cabinet.

The working memory contains details of components which have been ordered for a particular configuration task, descriptions of partial configurations and other dynamic information such as the results of various computations and context symbols indicating which contexts are currently active and the relative length of time they have been active for. The working memory

is initially empty. As the program proceeds, details of components and then partial configurations are stored in the working memory. New descriptions are added; old ones, which are no longer necessary, are deleted. Eventually, the working memory contains the full computed configuration.

3.4.2 The control mechanism

XCON divides its task into sub-tasks, which it then divides into sub-sub-tasks, etc. At the top level, there are six major sub-tasks:

1. check the order for missing or mismatched components;
2. put the appropriate components into the cpu and cpu expansion cabinets;
3. put boxes in the unibus expansion cabinets and put appropriate components in those boxes;
4. put panels in the unibus expansion cabinets;
5. do the floor lay-out;
6. work out the cabling.

For each of these sub-tasks, a context is selected and the relevant rules retrieved. XCON works in a data-driven (forward-chaining) mode. Unlike DENDRAL, MYCIN and Prospector, it does not use the generate-and-test approach in which a hypothesis is generated and then tested on the data. Instead, it has a powerful matching algorithm and enough knowledge in its rules to enable an appropriate action to be taken at each stage of the configuration task. No backtracking is necessary.

For each context there are a small number of relevant rules. Some of these rules are specializations of other rules (the more specialized rule will always fire first); some rules check for missing components; others are sequencing rules. In each case, there is enough information in the rules for the program to know what to do next; whether to add a missing component, or to put a

device in a cabinet, or to go on to the next context. The program proceeds in a step-wise manner until the configuration task is complete.

3.5 Xi Plus

Xi Plus is a modern, commercially available, rule-based expert system shell. It comprises an empty database, an empty knowledge base and a control mechanism, as well as facilities for editing, interfacing to external programs, graphics packages, etc.

3.5.1 The database

Xi Plus' database is the equivalent of MYCIN's dynamic database or XCON's working memory. When a user runs an application for advice about a particular problem, he may volunteer information concerning the problem or he may be asked to give information in response to specific questions. The system stores his answers (or any volunteered information) in the database, which is then accessed by the system's knowledge base or by an external program via the control structure. Any inferences made by the system are also stored in the database.

3.5.2 The knowledge base

Domain-specific knowledge is stored in Xi Plus' knowledge base as inference rules or facts, or it may be in an external program which is then referenced by an item in the knowledge base. A rule's *premise* and a fact may both be either an *assertion* (a statement which is true or false) or a *relation* of the form

⟨identifier⟩ ⟨relation⟩ ⟨value⟩

or

⟨attribute⟩ of ⟨identifier⟩ ⟨relation⟩ ⟨value⟩

where ⟨identifier⟩ or its ⟨attribute⟩ is the object to be described, and can be single-valued or multi-valued, ⟨value⟩ is its value, and ⟨relation⟩ is either a numerical relation (=, ≠, >, >=, < or <=) in which case ⟨value⟩ must be numerical, or one of:

- is/are [not] (⟨identifier⟩ must be single-valued)
- [does not] include[s] (⟨identifier⟩ is multi-valued)
- is [not] a (⟨identifier⟩ is a member of the set ⟨value⟩)
- or it can be user defined.

Defaults may be specified, in which case they are also stored in the knowledge base.

The action part of a rule may deduce the ⟨value⟩ of an ⟨identifier⟩ or it may call an external program, load a knowledge base, instigate a new query or even change the control mechanism from backward-chaining to forward-chaining or vice versa.

The knowledge base also contains questions to be asked of the user to determine values of certain identifiers.

3.5.3 The control structure

Rules and facts may be processed in either a backward-chaining or forward-chaining mode. The default is backward-chaining, but the knowledge engineer responsible for building an application may request single-level or full forward-chaining if he feels that it is appropriate.

When a user enters a query, the program searches for the answer to this query in a set pattern. It looks first in the database to see if the answer is already there. If not, it searches the list of facts in the knowledge base. It then (if still unsuccessful) either asks the user for the answer (if there is a relevant question stored in the knowledge base) or tries to infer it using its rules. If this fails to produce an answer, the program may call an external program if one has been defined. As a last resort, the program can generate a relevant question to ask the user.

Chapter 4

Overview of machine learning from examples

Machine learning has been a topic of interest to AI researchers since AI research began. Different groups have approached the subject from different angles and with different aims. Some saw it as a means to understanding more about human learning; others were more interested in equipping computers with the ability to learn (not necessarily in the same way as humans do) with a view to making them more 'intelligent'. More recently, interest in the subject has surged again as a result of the need for automatic knowledge acquisition techniques for expert systems.

Machine learning is now a broad subject, with research being active on many fronts. It can be classified in a number of ways[1]. One of these is a classification based on what is being learned, e.g. procedures, skills, structural descriptions, inference rules, mathematical and/or scientific laws, etc. A second way of classifying learning is according to how learning takes place, e.g. learning by being told, learning by analogy, learning from examples or learning by discovery. All of these learning methods, and others, have at one time or another, been the subject of learning programs.

Future expert systems will probably require learning programs which are able to learn in a variety of ways and acquire many different types of knowledge. However, such programs will have to be built step-by-step, and

[1] Carbonell et al [12] describe three ways.

are not likely to be available for some time to come.

This thesis is concerned with a very small (but expanding) section of this field — that of automatic induction of classification rules from sets of examples. This chapter sets the background to the work by describing some programs which were 'milestones' in the history of learning from examples.

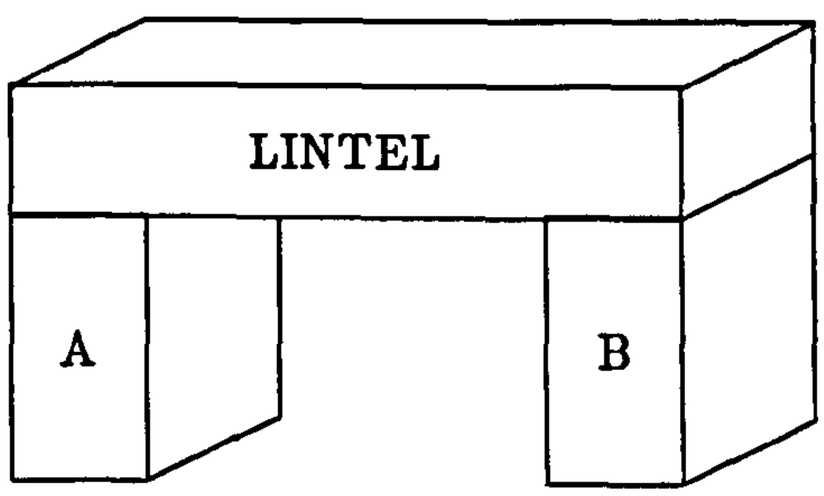
4.1 Winston's blocks

In 1970, Winston published his now well-known thesis [59][2] describing his research on the machine learning of structural descriptions from examples. At the heart of this work was a program which could learn the concept of an arch when presented with examples of arches and non-arches, one at a time. He used a semantic network to represent the learned concept. Figure 4.1 (from [15]) shows a positive example of an arch and a simplified description of it.

The program starts with a positive example of an arch such as that shown in figure 4.1 as its concept description. It is then given a negative example of an arch. This negative example must be a 'near-miss', i.e. a structure which is not an arch but differs from an arch in only one respect. Figure 4.2 is an example of a near-miss. The only difference between the descriptions of figure 4.1 and figure 4.2 is that the description of the near-miss does not include 'is-supported-by'. The program therefore infers that the two conditions:
is-supported-by (LINTEL, POSTA)
is-supported-by (LINTEL, POSTB)
are necessary for the structure to be an arch. It therefore changes these conditions to 'must-be-supported-by' and updates its concept description accordingly.

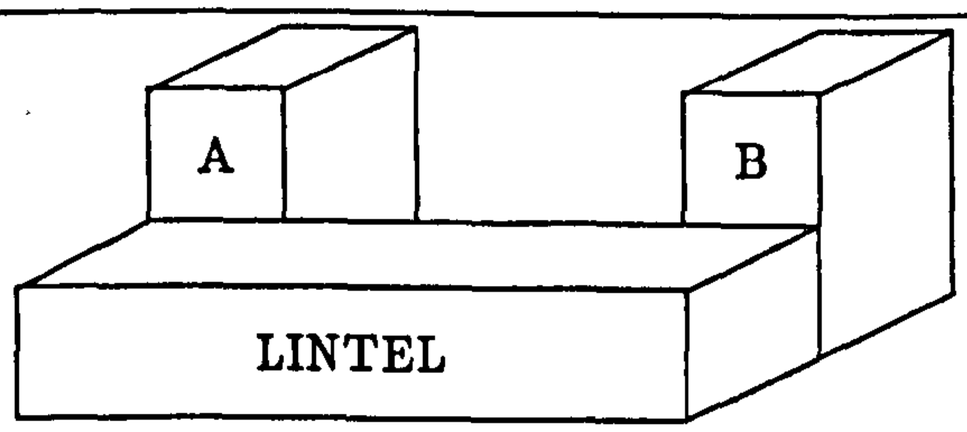
The program continues accepting examples of near-misses as long as new examples are available. Each time, some 'sufficient' condition is converted to a 'necessary' condition. It is also able to generalize by accepting new positive

[2]See also [58]



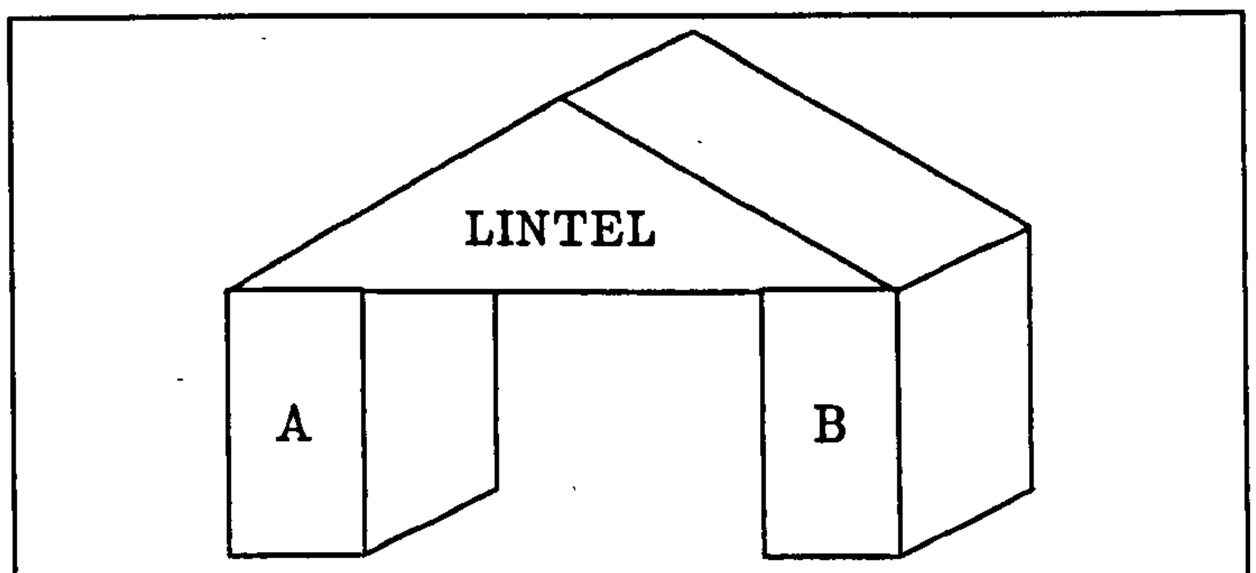
is-supported-by (LINTEL, POSTA)
is-supported-by (LINTEL, POSTB)
has-property (LINTEL, lying)
has-property (POSTA, standing)
has-property (POSTB, standing)
does-not-abut (POSTA, POSTB)
does-not-abut (POSTB, POSTA)
has-shape (LINTEL, rectangular)
a-kind-of (LINTEL, brick)
a-kind-of (POSTA, brick)
a-kind-of (POSTB, brick)

Figure 4.1 A positive example of an arch



has-property (LINTEL, lying)
has-property (POSTA, standing)
has-property (POSTB, standing)
does-not-abut (POSTA, POSTB)
does-not-abut (POSTB, POSTA)
has-shape (LINTEL, rectangular)
a-kind-of (LINTEL, brick)
a-kind-of (POSTA, brick)
a-kind-of (POSTB, brick)

Figure 4.2 A negative example of an arch



has-property (LINTEL, lying)
has-property (POSTA, standing)
has-property (POSTB, standing)
does-not-abut (POSTA, POSTB)
does-not-abut (POSTB, POSTA)
has-shape (LINTEL, triangular)
a-kind-of (LINTEL, brick)
a-kind-of (POSTA, brick)
a-kind-of (POSTB, brick)

Figure 4.3 A positive example of an arch

examples. Figure 4.3 is a positive example of an arch. The only difference between figures 4.1 and 4.3 is the shape of the LINTEL (rectangular in figure 4.1 and triangular in figure 4.3). As these are both positive examples of an arch, the program concludes that the shape of the LINTEL does not matter and drops the 'has-shape' condition. The program continues learning in this step-wise manner until all possible examples have been exhausted.

4.2 Meta-DENDRAL

At about the same time as Winston was developing his program for inferring structural descriptions at MIT, work was in progress at Stanford on the DENDRAL project. There, researchers were having difficulties in extracting domain-specific rules from experts to incorporate into their system. They decided to try to simplify the problem by building a program to infer these (cleavage) rules automatically. This program, Meta-DENDRAL [11], is the first example of automatic rule induction from examples for expert systems.

Like DENDRAL, it uses a plan-generate-test strategy. It consists of three programs — INTSUM, RULEGEN and RULEMOD.

4.2.1 INTSUM

INTSUM takes as input three-dimensional structures of a class of molecules and their associated mass spectra. These can be thought of as training instances of the form

$$\langle \text{whole molecular structure} \rangle \Longrightarrow \langle \text{mass spectrum} \rangle.$$

INTSUM converts these training instances into a set of very specific cleavage rules of the form

$$\langle \text{whole molecular structure} \rangle \Longrightarrow \langle \text{one designated broken bond} \rangle.$$

In order to do this, it makes use of what has been called a 'half-order theory'. This is a set of general rules of fragmentation and migration, e.g.

- double bonds and triple bonds do not break
- no aromatic bonds break
- two bonds to the same carbon atom cannot break together
- at most two hydrogen atoms can migrate after a fragmentation

These rules are applied to each molecular structure to simulate the action of the mass spectrometer and produce a spectrum. This spectrum is then compared with the original spectrum. If there are matching peaks then INTSUM infers that the cause of the simulated peak is a possible cause of the observed peak. Once the data has been interpreted, the results are summarized to produce a set of highly specific cleavage rules for each fragmentation in each molecule, together with its total evidence.

4.2.2 RULEGEN

RULEGEN also generates a set of cleavage rules, but these are much more general than those produced by INTSUM. It starts with the most general rule possible:

$$x * x \text{ (x is any atom type)}$$

i.e. all bonds break. It then specializes this rule in all possible ways. The specialization is made either by adding new neighbour atoms or by specifying an atom feature. There are four features which can be specified:

- atom type, e.g. carbon, nitrogen
- the number of non-hydrogen neighbours
- the number of hydrogen neighbours
- the number of double-bonded neighbours.

As each specialization is made, it is tested against the positive training instances produced by INTSUM. Those specializations for which there is no supporting evidence are pruned. After each cycle of specializations, the new rules are compared to their 'parent' rules and an 'improvement criterion' is computed. This indicates the plausibility of the specialization, ensuring that it:

- predicts fewer fragmentations per training molecule than its parent
- predicts fragmentations for at least half of all the training molecules
- predicts fragmentations for as many molecules as its parent (unless the parent was too general)

Any specializations which do not meet these criteria are pruned. If all specializations of a parent are pruned, then the parent is output as a cleavage rule. The cycle of specialize-and-prune is repeated until no more specializations are possible.

The outcome is a set of general cleavage rules. However, these rules are only approximate, frequently being too general, redundant, or simply incorrect.

4.2.3 RULEMOD

RULEMOD is responsible for testing and refining the rules produced by RULEGEN. It does this by first removing redundant rules and merging overlapping rules. It then tests the remaining rules on negative evidence (i.e. incorrect predictions — the spectrum shows that the designated bond did not break). It specializes the rules when necessary to remove negative evidence. Some of the rules produced by RULEGEN are too specific, so RULEMOD generalizes these, and finally any redundancies which are introduced during these procedures are removed. The final outcome is a set of cleavage rules which have been proved to be highly accurate [11].

4.3 Mitchell's candidate elimination algorithm

At the same time as work on Meta-DENDRAL was in progress, a post-graduate student at Stanford, Tom Mitchell, was working on a more general learning algorithm — the candidate elimination algorithm[3] [39], [15]. Candidate elimination is based on the formation and modification of rule version spaces.

Back in 1974, Simon and Lea [54] had described the problem of learning rules from examples as one of using training instances to discover general rules. They coined the terms 'rule space' and 'instance space', which refer to the space of all possible rules, and the space of all possible training instances, respectively. For any learning problem of appreciable size, however, the entire rule space is too large to be manageable. Mitchell observed that rules can be partially ordered according to their generality. For example, *if A then X* is a generalized version of *if A and B then X*, which in turn is a generalized version of *if A and B and C then X*. Therefore, it was not necessary to list the entire set of rules to describe the rule space — a set of most specific and a set of most general versions would suffice; all other rules would lie between these boundaries. A correct rule or rules describing

[3]This algorithm was later (1978) applied to Meta-DENDRAL as part of Mitchell's thesis work.

	instance	+ve or −ve	most general rule boundary	most specific rule boundary
1	*A, B, C, D*	+ve	null	*if A and B and C and D then X*
2	*B, C, D, ¬A*	−ve	*if A then X*	*if A and B and C and D then X*
3	*A, B, C, ¬D*	+ve	*if A then X*	*if A and B and C then X*
4	*A, C, D, ¬B*	−ve	*if A and B then X*	*if A and B and C then X*
5	*A, B, D, ¬C*	+ve	*if A and B then X*	*if A and B then X*

Figure 4.4 Example of candidate elimination

a set of training instances, being more general than the instances themselves but more specific than the null rule, would therefore lie somewhere between these two boundaries. Mitchell called the space between the boundaries the rule version space (i.e. the space which includes all plausible hypotheses or rule versions).

Mitchell's candidate elimination algorithm makes use of version spaces. It starts with the entire rule space as its version space and proceeds to test it on training instances, which are presented one at a time. If the instance is a positive one, the most specific rule is removed or eliminated from the space, i.e. it is generalized; if the instance is a negative one, the most general rule is removed, i.e. it is specialized. The version space gradually shrinks as more instances are presented until it contains only the correct rule or rules.

As an example, consider the rule *if A and B then X*, which the program must induce. Suppose the first instance presented to the program is a positive one in which *A, B, C* and *D* are all true. A version space will be set up in which the most general rule boundary is the null rule, and the most specific rule boundary is *if A and B and C and D then X*. Let the next example be a negative one in which *B, C* and *D* are true and *A* is false. The program infers that as this is a negative example, *A* must be necessary and changes the most general rule boundary to *if A then X*. The course of events is summarized for clarity in figure 4.4. As can be seen from figure 4.4,

instance 3 (A, B, C and $\neg D$) which is positive, generalizes the most specific rule boundary to *if A and B and C then X*; instance 4 (A, C, D and $\neg B$) which is negative, specializes the most general rule boundary to *if A and B then X*, and finally instance 5 (A, B, D and $\neg C$) which is positive, generalizes again the most specific rule boundary to *if A and B then X*. At this time the most specific and most general rule boundaries are identical — *if A and B then X*. The program, therefore, outputs this rule as the learned rule.

One of the disadvantages of the original candidate elimination algorithm was that it could not deal with disjunctive concepts. However, a solution to this was quickly found [15] and the algorithm was modified to enable it to be applied iteratively to a set of examples. The improved algorithm induces one rule at a time, removing the instances covered by it at each iteration.

4.4 AQ11

AQ11 is a multiple-concept learning program which inductively determines a set of classification rules for a complex domain. It was developed by Ryszard Michalski and others at the University of Illinois in 1977/78 and is described in [35] and [36].

AQ11 has been applied to the domain of disease diagnosis; specifically, to diseases of the soybean plant. The induced rules are required to diagnose any one of fifteen diseases when presented with the symptoms of the diseased plants. A selected set of examples of all diagnoses is used as input to the program and a further set is used for testing purposes. The domain is non-trivial, particularly as it has associated with it an element of uncertainty, both in the sense that a case description may be classified differently by different experts, and that the actual descriptions may contain uncertainties.

Each example of a diseased plant (called an event) is characterized by a conjunction of attribute-value pairs. There are 35 attributes (called descriptors) each with its own set of possible values, and each concerned with one particular aspect of the plant's condition. Some attributes describe

environmental factors, such as time of occurrence or temperature; others describe the plant as a whole (e.g. height) or parts of the plant (e.g. condition of leaves or stem). A description in terms of all 35 attributes should give enough information to diagnose the disease. The representation language used is a variable-valued logic calculus called VL1, which allows each attribute to be related in a variety of ways to one or more of its values. It is a very rich language, capable of set manipulation as well as conjunction and disjunction. There is also provision for denoting degree of certainty.

AQ11 simplifies the problem of learning multiple concepts by converting it into a series of single-concept learning problems. It uses a 'covering' algorithm, Aq (developed by Michalski in the late 1960's and extended in the 1970's), which was designed specifically for use with this representation language.

Aq is very similar to Mitchell's candidate elimination algorithm. It chooses as its starting point a positive example from the training set. This example is maximally generalized in all possible ways by removing all terms except one, and each generalization is applied to the complete set of negative examples. If any generalization is found to cover some of the negative examples, it is specialized by adding another term from the original positive example. It is then tested again and the process is repeated until no negative examples are covered. The result is one or more generalizations, each of which is a conjunction of terms. The best of these is selected, using pre-defined selection criteria, and all positive examples covered by it are removed from the training set. This generalization constitutes the first 'rule'. Aq then selects another positive example which is not covered by any of the generalizations, and repeats the process to find the second rule. It continues in this way until all positive examples are covered.

Because the same representation is used for both examples and rules, the Aq algorithm can substitute a generalization for the set of negative examples. AQ11 uses this technique whenever possible, and in doing so, makes substantial savings in computational effort.

The general induction process comprises four steps:

Step 1 For each concept (classification) a selected hypothesis is tested on the data creating two sets of examples:

- F^+ contains those examples which should be covered by the hypothesis but are not, and
- F^- contains those examples which are covered but should not be.

Step 2 The covering algorithm is used to determine a set of generalizations which covers all of the examples in F^-, but no others (these are called 'exception' examples).

Step 3 For each concept, the covering algorithm is again used to determine a set of generalizations which covers all of the positive examples. The generalizations developed in Step 2 and any previously generated hypotheses for other concepts are used as negative examples whenever possible. This 'corrects' the original hypothesis by effectively removing from it the set of exception examples and adding it to the set of examples in F^+.

Step 4 The rules are simplified according to pre-specified criteria.

This process produces a set of if *premise* then *action* rules, where each *premise* is a conjunction of terms, and the *action* is a diagnosis. Statistical information is also provided, which indicates the number of examples to which each rule applies. Where rules represent only a few examples, these may be considered as 'exceptions' or may indicate errors. Evaluation of the derived rules has shown AQ11 to be highly accurate [35].

Chapter 5

The ID3 family

ID3 [44,46,47,48] is an algorithm which was developed by Ross Quinlan in 1978/9 to induce classification rules in the form of decision trees from large sets of examples. It was based on a learning algorithm, CLS (Concept Learning System), designed by Earl Hunt in the early 1960s [29]. Hunt defines a concept to be a decision rule which is applied to a description of an object (given in terms of values of pre-defined attributes) to determine whether or not the object is a member of a specified set or class of objects. He found it convenient to represent such decision rules as sequences of tests of the values of individual attributes, and as he was concerned only (initially) with simple conjunctive characteristic[1] descriptions of single concepts, these sequences of tests naturally formed binary decision trees. For example, the concept *cat* can be described as 'a four-legged, furry animal which purrs'. This is a conjunctive characteristic description. Cats have four legs (mutations and mutilations are not considered here); cats have furry coats; cats purr. Thus a cat can be described in terms of three attributes — number of legs (possible values : 2, 4, more than 4), type of coat (possible values : fur, feathers, scales, hide) and purrs (possible values : yes, no). The description of a cat given above is the only one possible in terms of these attributes. There is no alternative description. A system (human or otherwise) which, when presented with an animal, must decide whether or not it is a cat,

[1]Hunt also applied his algorithm to disjunctive concepts but always found that the results were considerably worse than for conjunctive concepts.

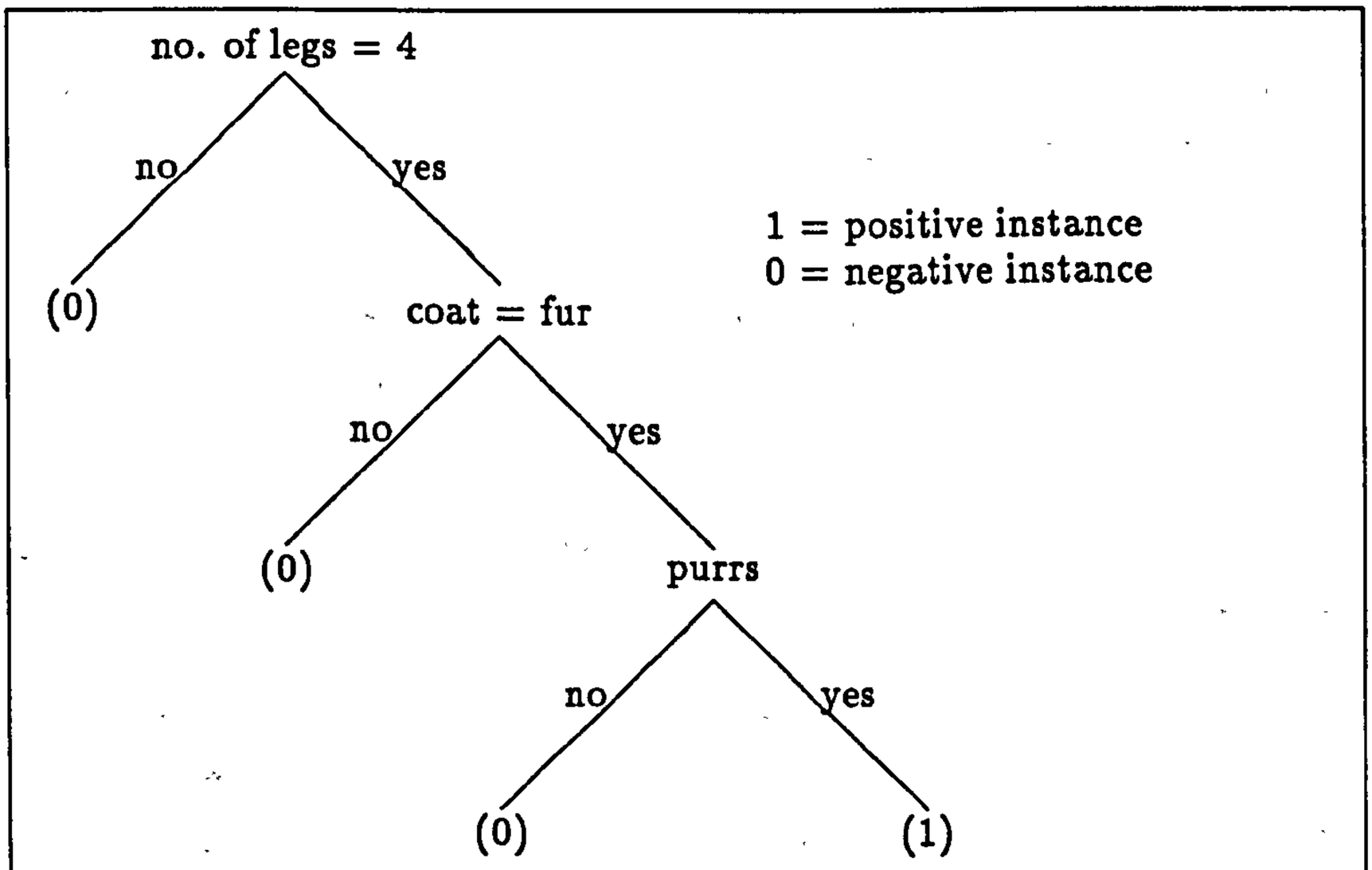


Figure 5.1 Binary decision tree for concept *cat*

has to ask three questions — *Does it have four legs?*, *Does it have a furry coat?* and *Does it purr?*. Hunt found it most convenient to represent this test sequence as a decision tree e.g. figure 5.1. Thus the framework for the original rule induction problem addressed by Hunt and his co-workers was set. Objects were described in terms of values of a fixed number of predefined attributes. Each object was either a positive example or a negative example of a concept. The decision as to whether an object was a positive or negative example of a concept could be made on the basis of a sequence of tests performed on individual attributes. This sequence of tests could be represented as a binary decision tree.

5.1 The CLS experiments

Hunt devised a number of CLS algorithms (CLS1 – CLS9) [29], of which CLS2 – CLS9 were all modifications of the basic algorithm CLS1. CLS1 proceeds as follows:

1. Search for a value of an attribute which appears in the description of all positive instances and no negative ones. If found, the problem is solved as the concept can be described in terms of that single attribute.

2. If no such value exists, then search for the reverse, i.e. a value of an attribute which appears in all negative instances and no positive ones.

3. If steps 1 and 2 fail, count the frequency with which values appear in the descriptions of positive instances only. Choose the attribute-value pair with the highest frequency of occurrence as the root node of the decision tree.

4. Divide the set of examples into two sets — one containing examples which have the attribute-value pair chosen in step 3, the other containing all other examples. Assign each set to a branch of the decision tree.

5. Apply the algorithm to each branch of the decision tree until all leaf nodes contain instances which are either all positive examples or all negative examples.

CLS2 and CLS3 differed from CLS1 only in the amount of memory which was made available to the algorithm. CLS4 and CLS5 were adapted so that critical examples, i.e. examples which had previously been misclassified, could be added to the example set. Step 3 in the above algorithm was altered for CLS6, such that the *relative* frequency of occurrence was calculated. CLS7 and CLS8 were allowed to select which of two groups of instances would be called positive and which negative. Finally, CLS9 was developed as an algorithm which combined a number of features from previous versions.

Hunt applied all nine algorithms to a number of sets of test data. These data were all artificial, designed to represent various levels of complexity ranging from simple conjunction to double implication (see [29] for a more detailed description of both algorithms and data). The results of applying algorithms CLS1 – CLS8 to the data were, with some minor variations, very similar. Each algorithm performed very well on simple conjunctive data and less well as disjunction was introduced. Different algorithms were best suited to different types of data. Hunt attempted to resolve this problem with CLS9 by defining a 'cost' of selecting an attribute in step 3 of the algorithm. Thus, for every attribute, CLS9 computes

$$H_i = \sum_{j=1}^{V_i} n_{ij*}$$

where $V_i =$ number of possible values of attribute i,
$n_{ijk} =$ number of instances at current node which have value j of attribute i and are in class k, and
$n_{ij*} =$ maximum of n_{ijk} over k.

The attribute which maximizes H_i is selected. To his great surprise, Hunt found that on the whole, CLS9 performed less well than previous versions, particularly CLS2, to which it was most similar. He attributed this to the fact that CLS9 made finer distinctions between the data and thus was more prone to errors. The biggest difference between CLS9 and the other versions was, of course, that the resultant decision tree was no longer binary. At each node, CLS9 sets up as many branches as there are values of the selected attribute. This leads to a much more complex tree, with many more branches for irrelevant attributes to be included in.

5.2 ID3

The core of Quinlan's ID3 algorithm [44,46,47] is essentially the same as Hunt's CLS. Quinlan however, chose to use a different strategy for selecting

an attribute at each node in the tree from Hunt's cost calculation. His concern was that the tree be as simple as possible. He argued that at each internal node of the tree there is a collection of positive and negative instances. The simplest sub-tree which could be formed at this node would be one in which, for each value of a selected attribute, the corresponding instances were either all positive or all negative. Failing this, if an attribute could be chosen such that for each of its values, the corresponding instances were mostly positive or mostly negative, then the resulting sub-trees would be less complex than otherwise. After some trial and error, Quinlan decided that the complexity of a tree could be adequately represented by the formula

$$\sum_{j=1}^{V_i} \sqrt{\text{minimum}(s_j, l_j)}$$

which is calculated for each attribute, and where

$V_i =$ number of values of candidate attribute,

$s_j =$ number of positive instances which have the jth. value of the candidate attribute,

$l_j =$ number of negative instances which have the jth. value of the candidate attribute.

The attribute which minimizes this sum is selected.

Later (see [46]), Quinlan abandoned this ad hoc complexity estimate in favour of a formula derived from information theory. For this, the decision tree is thought of as a source of a message, and the amount of information conveyed by this message is related to the complexity of the tree. When the tree is being formed, at each node the attributes can be tested for expected information gain in the resulting tree if that attribute were selected for partitioning. That attribute is selected which minimizes entropy (and thus maximizes average information gain). This formula and its use are described in detail in section 7.1.

Quinlan chose as his initial domain a chess end game in which the only pieces left on the board are the two kings, white rook and black knight, and it is black's turn to move. The program must derive a rule which,

given the positions of these pieces on the board, can determine whether or not the knight's side is lost in two ply. There are about eleven million legal configurations of these pieces. However, Quinlan argues that it should be possible to derive a rule, or collection of rules, which can classify each position as lost or safe taking into account only the relationship between the pieces, i.e. without applying the laws of chess.

The attributes which characterize each configuration of pieces on the board are chosen by a domain expert and describe the 'adjacency' of pieces, e.g. black king, knight, rook in line; distance of white king to knight. No two positions having different classifications share the same description. Thus the training set consists of a large number of instances, each of which is a description of a position in terms of a pre-defined set of attributes, and the class of which (i.e. lost or safe) is known. The induced rule(s) should be able to deduce this class given a set of values for the attributes.

ID3's decision tree is built recursively:

1. Each instance in the training set is examined to determine its class. If all instances belong to the same class, or the training set is empty, then the objective has been reached and the program terminates, returning either a class value or 'null'. Otherwise, the training set consists of two or more instances of different classes.

2. An attribute is selected as the root node of the decision tree.

3. The training set is partitioned according to the values of this selected attribute, forming the branches of the decision tree.

4. The algorithm is applied to each subset to build a descendant subtree for each branch.

Each leaf of the resultant tree consists of a set of instances of a single class, which may be empty. The class of instances at each leaf is then described by the attribute-value pairs used in the path from the root node to that leaf.

5.2.1 The KRKN experiments

Quinlan applied the above algorithm to the King-Rook-King-Knight chess end game problem referred to earlier, in which the attributes describe relationships between these chess pieces on a board. The classifications are *lost* (for knight's side, two-ply) and *safe*, where knight's side is lost two-ply (black to move) if a) black is checkmated, or b) black is not stalemated and on its next move, white can either checkmate black, or capture the knight without producing stalemate and without leaving the rook en prise.

Quinlan tackled this problem in stages. He conducted a series of seven experiments in which he first placed severe constraints on the number of allowable configurations of the pieces, and then gradually relaxed these constraints until in the seventh experiment he could apply his algorithm successfully to the original unrestricted problem. Thus, for the first experiment checkmate and stalemate were ignored (i.e. *lost* was true only if the knight was captured without leaving the rook en prise, even if this resulted in stalemate) and the knight was pinned; for the second experiment checkmate and stalemate were still ignored, but the position could be a pin, fork or skewer; edge effects were excluded for the third experiment; stalemate was allowed for the fourth; for the fifth and sixth experiments checkmate was also allowed but the position of the black king was restricted; and finally, all restrictions were lifted for the seventh experiment. As the complexity of the experiments increased, both the time taken for computation and memory requirements also increased. For the second experiment it was found necessary to adopt a tutorial approach in which the algorithm was first applied to a training set containing forty lost and forty safe positions, the resulting rule being checked by a human tutor, who would then select further instances which contradicted the rule, add these to the training set and apply the algorithm again. This was repeated until the rule was correct.

This tutorial approach was found to be extremely tedious and frustrating. Furthermore, it was difficult to prove that a rule was correct. For the third experiment, therefore, the concept of a 'window' was introduced and the algorithm was iteratively applied to a subset of the training set which

was modified until the resultant rule was correct. Thus the procedure was as follows:

1. Select a subset of the training set (the window)
2. Repeat:
   (a) Apply the algorithm to the window.
   (b) Test the rule on the remaining instances.
   (c) If exceptions are found modify the window.

   until no exceptions are found.

Two ways of forming the window were investigated. In the first, a predetermined number of exceptions was added to the window at each iteration, and in the second, the window size was kept constant by allowing exceptions to replace instances currently in the window. The second method was later abandoned because of the possibility of looping (see [6] for a more detailed discussion of this).

As the experiments became more complex, so it became increasingly difficult to define an adequate set of attributes. In the later experiments many plausible sets had to be tried before a satisfactory one was found. The final set (for the seventh experiment) contained 25 attributes varying in complexity from

> distance from black king to knight (possible values : 1, 2 and 3)

to

> knight can move to a restful square which is either between the mating square and the black king, or from which the knight can move to a square between the mating square and the black king (if a mating square exists) (possible values : t and f) [44]

Even this final set of attributes proved to be inadequate in that there were some instances which shared the same description but were of different

Classification Method	CPU Time (msec)
Minimax search	7.67
Specialized search	1.42
Using first decision tree	1.37
Using second decision tree	0.96

Table 5.1 Comparison of classification methods for lost 2-ply

classes. The problem was circumvented by allowing some leaves of the decision tree to be labelled 'search', meaning that the instances at such leaves had to be discriminated between by some other method[2].

Despite the difficulties which had been encountered along the way, the results of Quinlan's KRKN experiments were extremely impressive. The final tree was induced from only 7% of the initial training set and will give a correct answer with probability 0.9974 [44]. Furthermore, Quinlan showed that for classification purposes, decision trees were much more efficient than other search methods. Table 5.1 (taken from [48]) shows the average time required to classify a position by various methods on the DEC-KL10. The two decision trees referred to in table 5.1 were formed using different attribute sets. Specialized search is a classification method which uses domain knowledge.

The success of Quinlan's experiments created much interest in the AI research community. Various research groups began to develop their own ID3-like systems. Quinlan himself has enhanced his algorithm so that it can deal with noisy data, missing attributes, continuous values, etc. [43,45,46,50]. The basic induction algorithm described above has now been widely acknowledged as the father of many modern rule induction systems, notably ACLS [42], of which there have been a number of commercial derivatives, and ASSISTANT [30], both of which are described briefly below.

[2]This was not such a disaster at might appear at first sight. There were only 75 out of a total of 29,236 instances at 'search' nodes.

5.3 ACLS

ACLS (Analog Concept Learning System) is a UCSD Pascal program which uses a modified version of ID3 to produce a classification rule either in the form of a decision tree or as a Pascal conditional expression. It was written by Andy Paterson, Andrew Blake and Alen Shapiro at Intelligent Terminals Limited, Glasgow in 1981.

The core of the program is an induction algorithm which is effectively ID3 but which allows more than two class values and also allows attribute values to be one of two types — integer or logical. An attribute value of type *integer* can be any integer within the range allowed by Pascal. An attribute value of type *logical* can be any one of a number of specified discrete values.

The program was designed to be used interactively. Examples are held in two separate stores. A rule is induced from a training set of examples held in primary store. This rule may be induced using the complete training set, or it may be induced iteratively using a growing window as described in the previous section. In the latter case it may be that a rule which is correct for the whole training set is induced from only a subset of the examples in primary store. Any examples not used are then moved to a secondary store. Once a rule has been induced it is displayed and the user can then correct or refine it by adding new examples or counter-examples. These can be read in either from the terminal or from a file and are added either to the primary store if they contradict the current rule, or to secondary store if they supply new information but do not contradict the current rule, or if they 'clash' with examples already in primary store. Duplicate examples are ignored. When the user has finished adding examples, a new rule is induced using examples in primary store. The rule is displayed and clashes or exceptions in secondary store reported. The user continues to add new examples until he feels that the induced rule is correct. The program allows him to enter examples with 'don't care' values for attributes, to move examples between primary and secondary stores, and to examine and/or delete examples from either store.

Apart from being interactive, ACLS differs from ID3 in that it allows

continuous integer attribute values. Like ID3, ACLS computes an information theoretic measure, entropy, for each attribute at each internal node of the decision tree. For integer-type attributes, however, this measure has to be computed many times. Let attribute A be of type integer, and let the training set contain N examples each with a different value $V_i (i = 1 \ldots N)$ for A. For each V_i, the N examples are divided into two subsets — one with values of A less than V_i, and the other with values greater than or equal to V_i. Entropy is calculated for a decision tree having these two subsets as branches. Entropy is thus calculated N times, and that value of A which minimizes it is selected for comparing with entropy values for other attributes.

One other major difference between the two types of attribute is that whereas logical attributes can only appear once in any path from root node to leaf node in a decision tree, integer attributes can appear many times, the split being at a different value each time.

5.4 ASSISTANT

ASSISTANT is also an enhanced version of ID3. It was developed by Kononenko, Bratko and others at the Jozef Stefan Institute in Ljubljana, Yugoslavia in 1984.

Like ACLS, ASSISTANT is able to classify objects into more than two classes. Also, it allows attributes to have continuous values (real as well as integer) which it handles in the same way as ACLS. Unlike ACLS, however, ASSISTANT performs a binary split for *every* attribute, discrete (logical) as well as continuous. The reason for this is that, whilst working with ID3 in a medical domain, the research group found that the information theoretic measure used for attribute selection tended to favour attributes with many values [46,28]. By performing a binary split for every attribute, this bias was avoided, with a further advantage that the resulting binary decision tree tended to be smaller and more precise.

Further enhancements include the ability to handle missing information,

and a tree pruning heuristic which makes the algorithm reasonably robust in the presence of noise. Missing information is handled by the use of Bayes theorem, which is used to calculate the most likely value of an attribute (in an instance of given class) or to calculate the most likely class at a 'null' leaf node in the decision tree. The tree pruning ability enables the algorithm to terminate if there are too few instances in a training set to perform reliable computations. The leaf nodes of a pruned tree then contain instances of more than one class, the class with the highest relative frequency being considered to be the correct one.

ASSISTANT has been used to aid construction of knowledge bases for a number of expert systems in different domains.

5.5 Other enhancements

Many of the enhancements described above have also been implemented in a system called C4, which has recently been developed by Quinlan himself [50]. C4 induces a rule iteratively using a growing window as described in section 5.2.1. The basic algorithm is the same as in ID3 with the exception that, as in ASSISTANT, the program terminates if a specified 'stopping criterion' is satisfied, leaving a set of instances of more than one class at the leaf nodes. The stopping criterion is based on the chi-square statistical test for independence [46]. A rule is induced in the form of a decision tree. Frequently, because of noise in the data, this tree is overly complex. C4, therefore, allows a full tree to be generated, but then prunes it by examining subtrees and assessing the increase in error rate when classifying unseen instances if certain subtrees were replaced by leaves. While the error rate does not increase significantly, C4 continues to replace subtrees by leaves. C4 generates and prunes several decision trees and then selects the most promising one, based on size and complexity.

The complexity of decision trees has been a major consideration in the development of ID3-like systems. Trees of any appreciable size are notoriously difficult to assimilate and may be totally incomprehensible even to the

domain expert. This problem has been addressed by Alen Shapiro at the University of Edinburgh, who has developed a system which uses an interactive version of ID3 (probably a predecessor of ACLS) to perform 'structured induction' [52]. Shapiro's program relies on a domain expert to break the induction problem into a series of subproblems by introducing some high-level attributes. Thus the concept to be learned can be described in terms of these high-level attributes, which in turn are described by lower-level attributes, which can then be described by even-lower-level attributes, and so on. Using this system, extremely complex trees can be re-induced as a series of relatively simple trees which are much less opaque to the user.

Despite these enhancements, the potential complexity of decision trees is still one of the major limitations of the ID3 family of rule induction programs. This and other problems associated with decision trees are discussed more fully in Chapter 6.

Chapter 6

Induction of decision trees

The project reported in the remaining chapters of this thesis was designed to tackle the problem of automatically inducing basic classification rules in modular form from sets of examples. Each induced rule is to be an if ...then ...(inference) rule whose premise describes an object or situation in terms of a conjunction of features, where each feature is of the form <attribute> <relation> <value>, and which concludes that if the premise is satisfied then the object or situation is a member of a particular class from a pre-specified set of classes. The algorithm to be used must induce these rules by searching and selecting relevant features from a set of examples, each example being a description of a particular class in terms of values of attributes. The algorithm must be simple, efficient and robust, and must produce reliable rules which can be used with some degree of confidence to classify an object or situation and to discriminate between the various possible classes.

The most successful attempts so far at tackling this problem have arguably been Michalski's AQ11 and Quinlan's ID3. ID3, in particular, its potential having been demonstrated in the domain of chess endgames, came under the scrutiny of a substantial number of researchers, was modified and improved, tested in other domains, and soon adopted for use in a number of commercial applications. However, before long, some major limitations to the ID3 algorithm became apparent. In particular, the algorithm's inability to deal with noisy data and the incomprehensibility of its decision tree

output proved to be stumbling blocks in real-world applications. Attempts at remedying these faults have resulted in some of the enhancements to ID3 described in Chapter 5. However, the fact that ID3 produces its output in the form of a decision tree creates a much deeper (and more serious) problem than simply one of incomprehensibility. Some rules cannot be represented easily by decision trees. An expert system using a decision tree in these cases frequently demands the results of more tests than are necessary, with possibly serious consequences if these tests are expensive or dangerous to perform. This problem is discussed in section 6.5 of this chapter and is highlighted by means of a simple example, introduced in section 6.1.1.

The training set of examples to which ID3 (or any similar algorithm) is applied must meet certain requirements. These are discussed briefly in section 6.2 and sections 6.3 and 6.4 describe the characteristics of an ideal training set, arguing that if an induction algorithm is to perform well in real-world applications, it must first be known to perform well under ideal conditions.

6.1 An example

6.1.1 The domain

The following example, taken from the world of ophthalmic optics, will be used to illustrate the procedures involved in rule induction.

An adult spectacle wearer enters an ophthalmic practice with a view to purchasing her first pair of contact lenses. She has had her eyes examined recently elsewhere and has brought her prescription with her. She understands that there are different types of contact lenses available, and that it is the optician's decision as to whether or not she is suitable for contact lens wear, and if so, which type she should be fitted with.

From the optician's point of view, this is a three-category[1] classification problem. His decision will be one of:

[1]It should be noted that this is a highly simplified example. In real life there are many types of contact lenses and many more factors affecting the decision as to which type, if any, to fit.

δ_1 : the patient should be fitted with hard contact lenses,

δ_2 : the patient should be fitted with soft contact lenses,

δ_3 : the patient should not be fitted with contact lenses.

In reaching his decision he must consider one or more of four[1] factors:

a : the age of the patient

1. young,
2. pre-presbyopic, or
3. presbyopic

b : her spectacle prescription

1. myope, or
2. hypermetrope

c : whether she is astigmatic

1. no, or
2. yes

d : her tear production rate

1. reduced, or
2. adequate

Table 6.1 shows the optician's decision for each combination of the four factors. However, the optician does not carry such a table around with him, either on his person or in his head. Instead, through his training and experience, he has learned to exercise his professional judgement in each individual case, and will make his decision almost instinctively. If questioned as to how he arrived at a particular decision, his answer is likely to be of the form

> This patient is not suitable for contact lens wear because her tear production rate is reduced.

	value of attribute				decision[2]
	a	b	c	d	δ
1	1	1	1	1	3
2	1	1	1	2	2
3	1	1	2	1	3
4	1	1	2	2	1
5	1	2	1	1	3
6	1	2	1	2	2
7	1	2	2	1	3
8	1	2	2	2	1
9	2	1	1	1	3
10	2	1	1	2	2
11	2	1	2	1	3
12	2	1	2	2	1
13	2	2	1	1	3
14	2	2	1	2	2
15	2	2	2	1	3
16	2	2	2	2	3
17	3	1	1	1	3
18	3	1	1	2	3
19	3	1	2	1	3
20	3	1	2	2	1
21	3	2	1	1	3
22	3	2	1	2	2
23	3	2	2	1	3
24	3	2	2	2	3

Table 6.1 Decision table for fitting contact lenses.

or

> This patient can only be fitted with hard contact lenses because she is astigmatic. As she is young and has an adequate tear production rate, hard lenses are not contraindicated.

Each explanation is a justification of a decision in terms of the values of relevant attributes, and is based on one or more 'rules of thumb', i.e.:

[2]The reader is asked not to be tempted to use this decision table to determine whether or not (s)he is suitable for contact lenses as there are many factors, not mentioned here, which may radically influence the decision.

if tear production rate is reduced
then do not fit contact lenses,

or

if the patient is astigmatic, and
the patient is young, and
the tear production rate is adequate
then fit hard contact lenses.

Although the optician is able to justify easily each individual decision, he would find it quite difficult to formalize his knowledge as a complete set of rules. ID3 seeks to establish this underlying set of rules, in the form of a decision tree, from examples of the optician's decisions. The algorithm is described in detail in section 7.1. Table 6.1 is used as the training set of instances; δ_1, δ_2 and δ_3 are the decisions or classifications; a, b, c and d are the attributes. Attribute a has three possible values (1, 2 and 3) and attributes b, c and d each have two possible values (1 and 2). Each instance is a description of a classification in terms of values of the four attributes. The following notation will be used in the remaining sections of this thesis:

α_x = attribute α has value x

δ_n = class has value n

$a_1 \& b_1 \& c_2 \& d_2 \rightarrow \delta_3$ = an instance in which attribute a has value 1, attribute b has value 1, attribute c has value 2 and attribute d has value 2, and which is classified as δ_3.

$b_2 \wedge c_1 \wedge d_2 \rightarrow \delta_2$ = a rule which states that the set of instances in which attribute b has value 2, attribute c has value 1 and attribute d has value 2 is classified as δ_2

6.1.2 The results

Applying the algorithm described in section 5.2[3] to the training set shown in table 6.1 produces the decision tree of figure 6.1. As can be seen, the

[3]The original ID3 algorithm was designed to deal with only two classes, but it is a trivial matter to adapt it to deal with any number of classes.

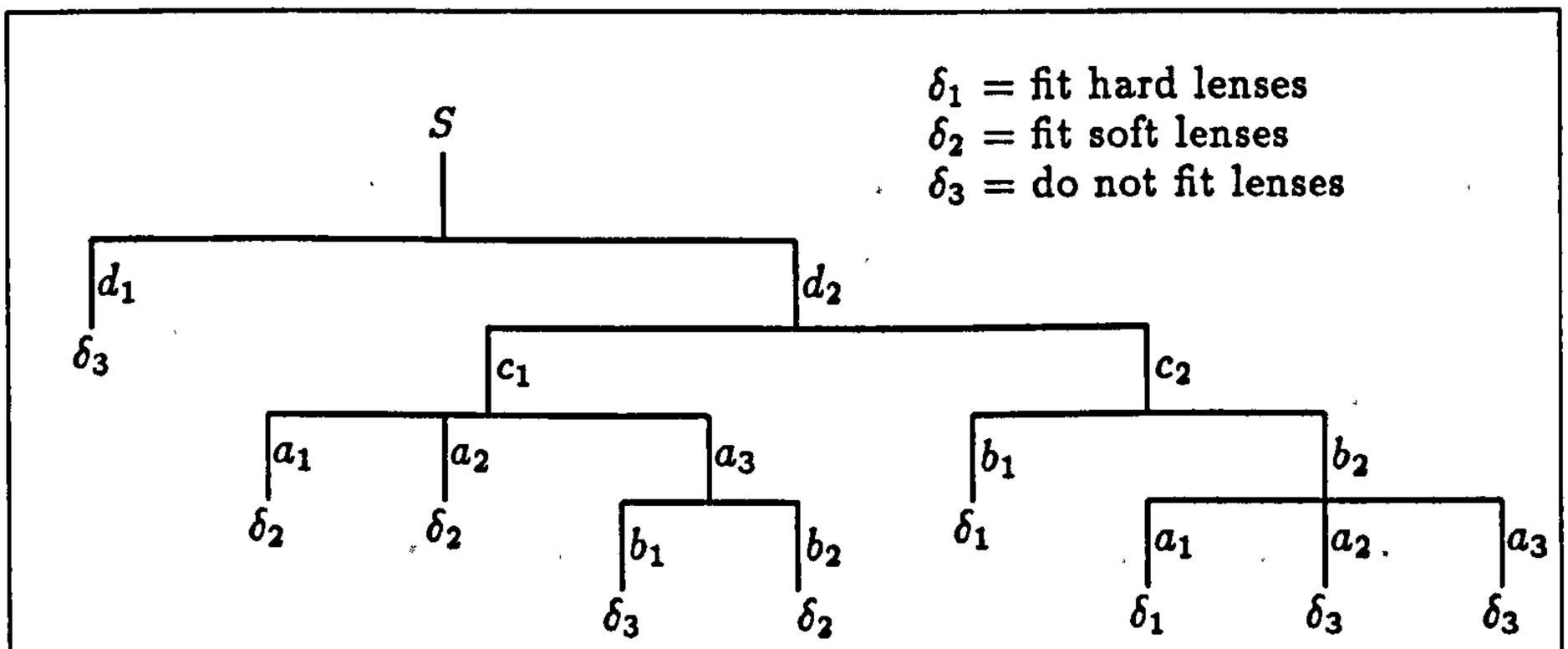


Figure 6.1 Decision tree produced by ID3

training set is divided first according to the values of attribute d, and then according to the values of attributes c, and a or b respectively. The resulting subsets are each of a single class. This is one of the simplest possible single decision trees which fully explain the data. Selecting the attributes in a different order does not reduce the number of nodes. For convenience, the decision tree can be rewritten as a set of individual rules:

1. $d_1 \rightarrow \delta_3$
2. $d_2 \wedge c_1 \wedge a_1 \rightarrow \delta_2$
3. $d_2 \wedge c_1 \wedge a_2 \rightarrow \delta_2$
4. $d_2 \wedge c_1 \wedge a_3 \wedge b_1 \rightarrow \delta_3$
5. $d_2 \wedge c_1 \wedge a_3 \wedge b_2 \rightarrow \delta_2$
6. $d_2 \wedge c_2 \wedge b_1 \rightarrow \delta_1$
7. $d_2 \wedge c_2 \wedge b_2 \wedge a_1 \rightarrow \delta_1$
8. $d_2 \wedge c_2 \wedge b_2 \wedge a_2 \rightarrow \delta_3$
9. $d_2 \wedge c_2 \wedge b_2 \wedge a_3 \rightarrow \delta_3$

There are nine rules, with a total of 30 terms — a considerable compression of the full decision table which contains $24 \times 4 = 96$ terms. This has been done extremely efficiently and entirely automatically. The decision tree which has been produced is simple and efficient to use as long as all the necessary data is available, and has the further advantage of being complete, i.e. all possible combinations of attributes and their values are represented.

6.2 The training set — necessary requirements

A training set to which ID3 is to be applied consists of a number of instances each of which is a description of a given classification or decision in terms of values for a fixed number of attributes. ID3 searches these instances and selects attribute-value pairs which appear to characterize particular classifications and to discriminate between two or more classifications. If the results are to be meaningful, the training set must meet certain requirements, namely:

- the set of attributes must be adequate,
- the classes must be specifiable in terms of attribute descriptions, and
- the classes must be mutually exclusive.

The following subsections briefly describe these requirements and the sorts of results which might be expected if they are *not* met.

6.2.1 The set of attributes must be adequate

The set of attributes is inadequate if there are two or more correct instances which have the same values for the attributes but are of different classes. In such cases, the algorithm tests each attribute in turn to try to distinguish between the classes, but fails to do so. The result is a decision tree which has one or more branches for which the class is indeterminate. In his original experiments [44], Quinlan chose to label the leaves of these branches *search*[4].

[4]This was later replaced by a probability that the instances corresponding to these leaves belong to a specified class [49].

For example, let the two instances

1. $x_1 \& y_2 \& z_3 \rightarrow \delta_1$

2. $x_1 \& y_2 \& z_3 \rightarrow \delta_2$

be present in the data. If the two instances are correct, this implies that a fourth attribute, say w, is needed to distinguish between the classes:

1. $x_1 \& y_2 \& z_3 \& w_1 \rightarrow \delta_1$

2. $x_1 \& y_2 \& z_3 \& w_2 \rightarrow \delta_2$

Omitting attribute w from the training set may have far reaching consequences. If it is the case that attribute w is necessary to distinguish between the classes only in the case where the value of x is 1, the value of y is 2 and the value of z is 3, its omission will be relevant only for a single branch. At the other extreme, if attribute w is the sole distinguishing attribute the result may be simply a message saying that the algorithm is unable to find a distinguishing attribute or it may be a tree in which some or all of the branches are labelled *search*. If the training set is incomplete it is possible for the algorithm to return a tree which distinguishes between classes in all cases (i.e. there are no *search* leaves) using the given attributes, even though in reality these are all irrelevant.

To illustrate the sort of results which can be expected when the set of attributes is inadequate, all references to attribute a were removed from the training set of table 6.1 and ID3 was applied to the remaining data. The decision tree of figure 6.2 was induced. This tree is similar to that of figure 6.1 except that two internal nodes referencing attribute a have been removed with the effect that the class is indeterminate at two leaves (B and E).

The experiment was then repeated with all references to attribute d removed from the training set. The result was the decision tree of figure 6.3. This tree is able to classify some instances of class δ_3 only, but these are classified without reference to attribute d, which was the attribute thought to be most important originally. No other instances can be classified.

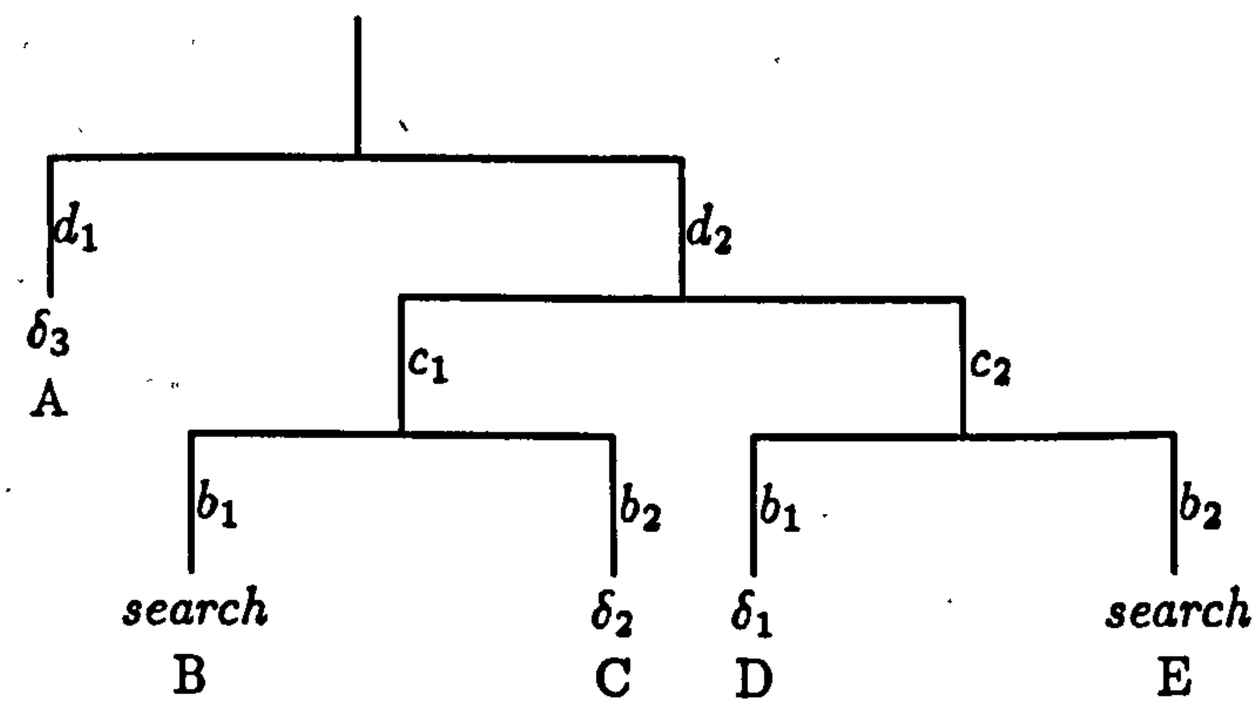


Figure 6.2 Decision tree produced when attribute *a* is missing

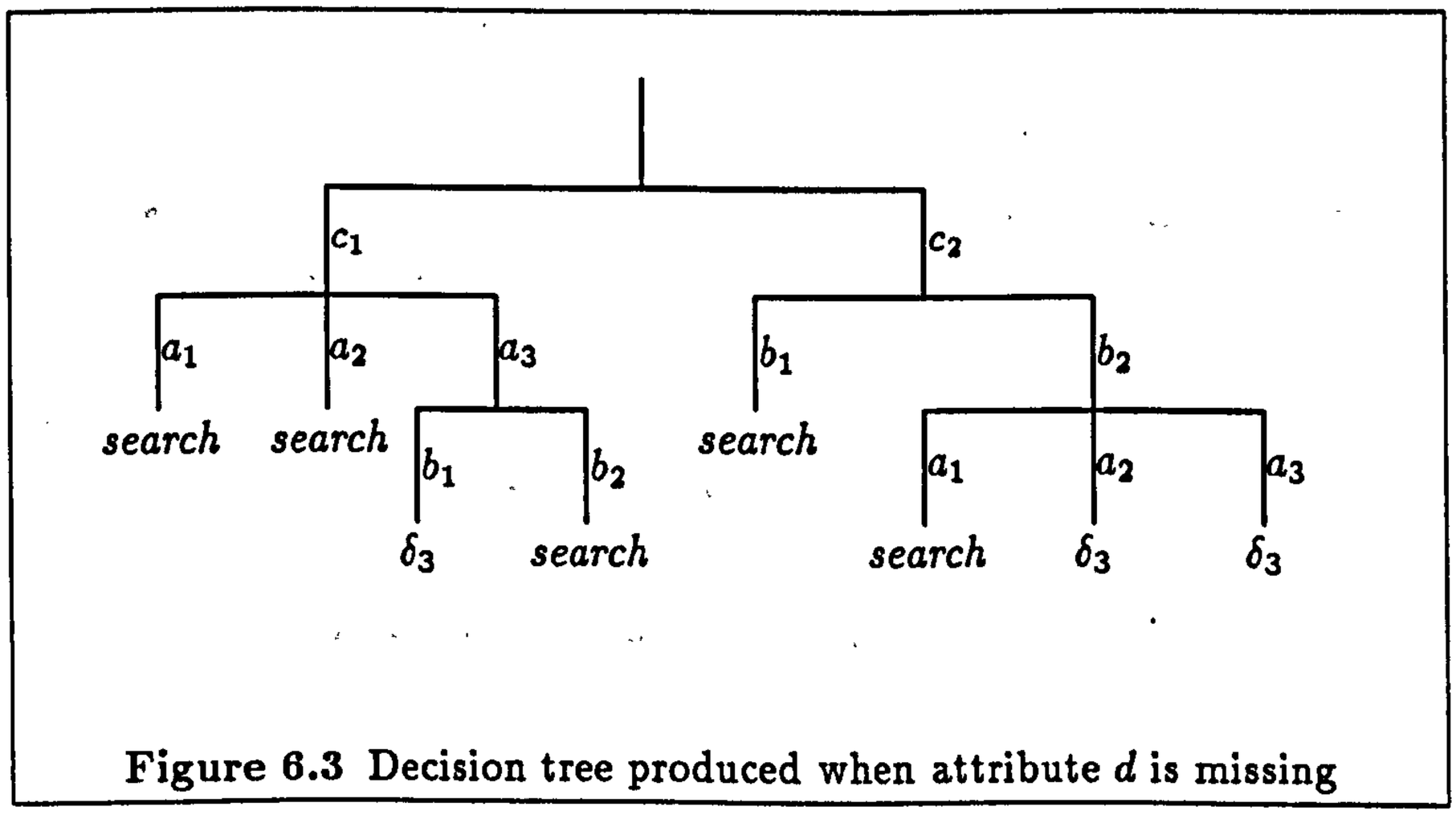


Figure 6.3 Decision tree produced when attribute *d* is missing

There is also a danger (which is not demonstrated by the above examples) when the set of attributes is inadequate that ID3 may discover apparent relationships between irrelevant attributes and class with the result that the induced decision tree may appear to classify some (or all) instances, but in reality be totally meaningless, and thus dangerously misleading.

6.2.2 The classes must be specifiable in terms of attribute descriptions

ID3 is unable to discover predicates or functions linking two or more attributes. Thus rules which describe structure, e.g. a is on top of b, or $a = b$, will not be induced.

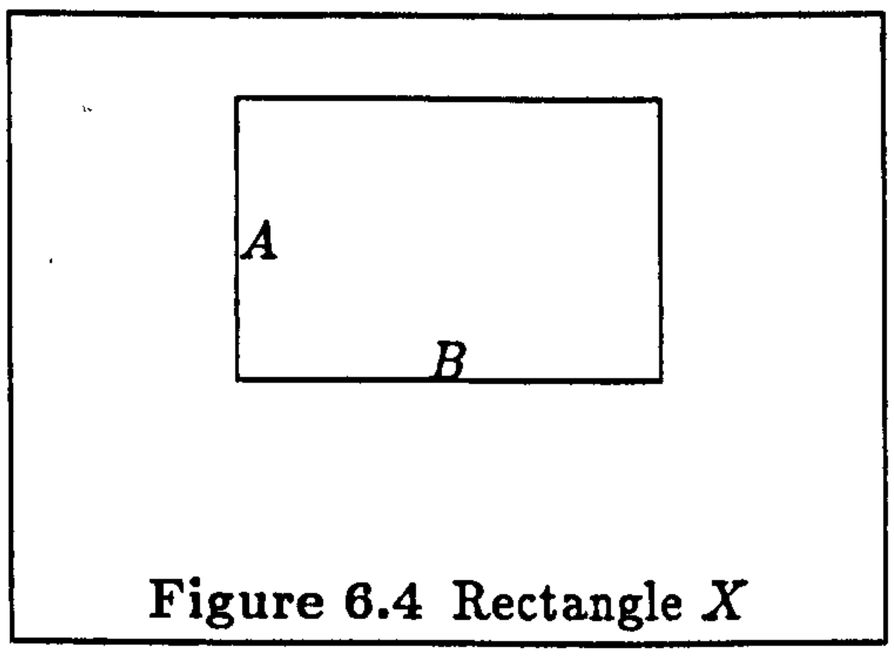


Figure 6.4 Rectangle X

For example, a square is defined to be an equilateral rectangle. Let A and B be two adjacent sides of rectangle X (fig. 6.4), and a and b be the lengths of A and B respectively, each with possible values 1, 2 and 3 units. Let the classifications be *square* and *not-square*. Given the training set shown in table 6.2, we wish to derive rules which would determine whether or not rectangle X is also a square.

The rules which we would like to derive are:

1. $a = b \rightarrow \delta_{square}$

2. $a \neq b \rightarrow \delta_{not-square}$

a	b	δ
1	1	*square*
1	2	*not-square*
1	3	*not-square*
2	1	*not-square*
2	2	*square*
2	3	*not-square*
3	1	*not-square*
3	2	*not-square*
3	3	*square*

Table 6.2 Training set for classifying rectangles

which can be represented by the decision tree of figure 6.5. However, as ID3 does not attempt to relate values of attributes to any values other than those of the class, the actual rules derived from the training set of table 6.2 (the decision tree is shown in figure 6.6) are:

1. $a_1 \wedge b_1 \rightarrow \delta_{square}$
2. $a_1 \wedge b_2 \rightarrow \delta_{not-square}$
3. $a_1 \wedge b_3 \rightarrow \delta_{not-square}$
4. $a_2 \wedge b_1 \rightarrow \delta_{not-square}$
5. $a_2 \wedge b_2 \rightarrow \delta_{square}$
6. $a_2 \wedge b_3 \rightarrow \delta_{not-square}$
7. $a_3 \wedge b_1 \rightarrow \delta_{not-square}$
8. $a_3 \wedge b_2 \rightarrow \delta_{not-square}$
9. $a_3 \wedge b_3 \rightarrow \delta_{square}$

These rules are clearly just a reproduction of the training set, and as such have no value even for data compression.

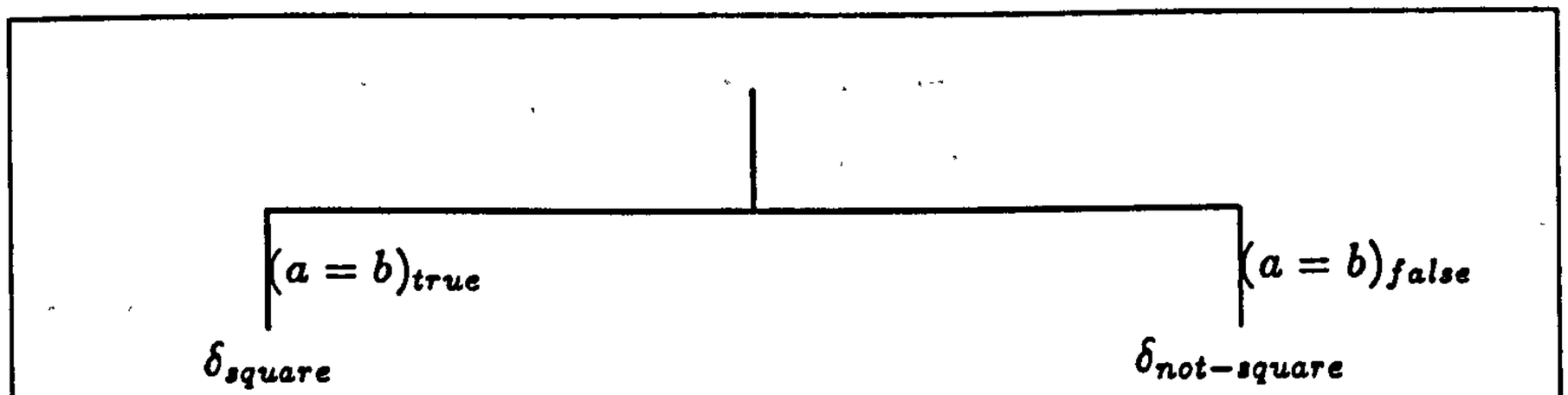


Figure 6.5 Decision tree for classifying rectangles (ideal)

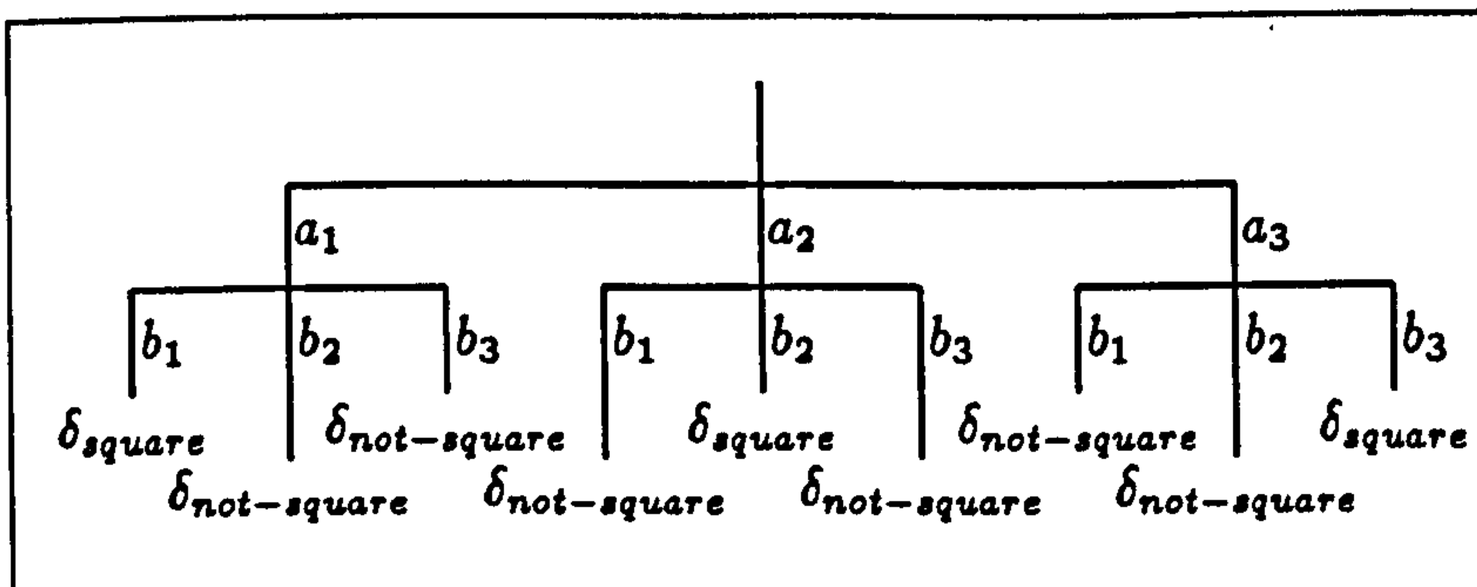


Figure 6.6 Decision tree for classifying rectangles (actual)

6.2.3 The classes must be mutually exclusive

Induction algorithms such as ID3 are based on finding attributes which distinguish between classes. Thus if the classes are not mutually exclusive there will be instances which can be classified correctly in more than one way, and the results of induction will be similar to results obtained when the set of attributes is inadequate (see subsection 6.2.1).

For example, let class δ_2 be a subset of class δ_1 and the following instances be correct:

1. $x_1 \& y_2 \& z_3 \rightarrow \delta_1$

2. $x_1 \& y_2 \& z_3 \rightarrow \delta_2$

Unlike the situation where the set of attributes is inadequate, in this case there is no fourth attribute which will distinguish between the classes.

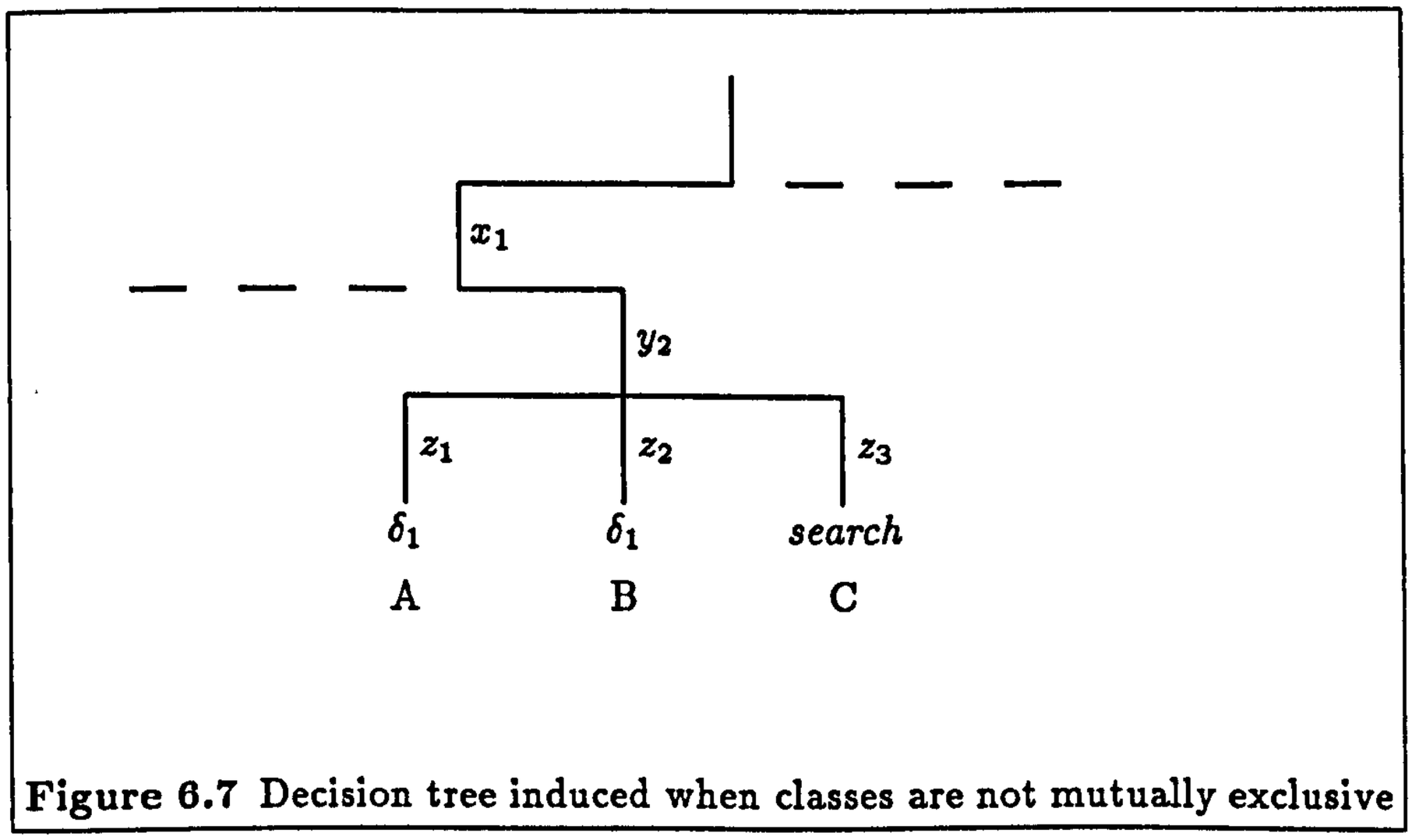


Figure 6.7 Decision tree induced when classes are not mutually exclusive

Let it be the case that for $x_1 \wedge y_2$ the class is δ_1 only but for $x_1 \wedge y_2 \wedge z_3$ the class is δ_2. Because δ_2 is a subset of δ_1, this latter instance can be classified correctly both ways. This situation cannot easily be represented as a decision tree. Nevertheless, ID3 would attempt to induce a tree. The result is shown in figure 6.7, in which branches A and B are too specific with

respect to attribute z and the class of instances corresponding to branch C is indeterminate. The results are similar if δ_1 and δ_2 are overlapping sets.

6.3 The training set — other characteristics

The training set of table 6.1 fulfills all three of the above conditions. It is also an 'ideal' training set in the sense that:

- it is complete
- the values of all attributes are discrete
- there are no duplicate instances
- there is no noise

These characteristics are not typical of most real-world training sets. In particular, it would be most unrealistic to expect a training set to be complete, i.e. to contain all possible instances[5]. If this were the case, rule induction would be useful only for data compression. No new information would be provided by the induced rules as classification of an instance could be determined in every case by means of a simple table look-up algorithm. The main value of rule induction is that rules induced from incomplete training sets can be used to predict the classification of new instances, i.e. instances not in the original training set. Induction from incomplete training sets is discussed in more detail in Chapter 9.

A requirement that the values of all attributes be discrete is more easily attained. In reality, attributes can have values which are discrete, i.e. the value is one, and only one, of a finite number of mutually-exclusive values, or continuous, i.e. the value is a point which lies within a range with fixed boundaries (which may or may not be ∞ and/or $-\infty$) but possibly an infinite number of points within the boundaries. Discrete values may be ordered, i.e. each value is a point which lies within a range with

[5]With a large number of multi-valued attributes, the total number of possible instances could soon add up to many millions or more.

type	attribute	examples of possible values
discrete (unordered)	sex	male, female
discrete (unordered)	shape	square, triangle, circle
discrete (ordered)	number of children	0, 1, 2, 5
discrete (ordered)	distance in squares from black king to knight	1, 2, 3, 7
continuous	distance in kms.	0.1, 576, 23.785
continuous	age in years	1.5, 81, $10\frac{11}{12}$

Table 6.3 Examples of different types of attribute value

fixed boundaries and a finite number of points within the boundaries, or unordered. Examples of attributes with discrete (unordered), discrete (ordered) and continuous values are given in table 6.3. Other types of attribute values, e.g. structured, can be found in [34].

In practice, the distinction between continuous and ordered discrete values can be ignored because continuous values must be measured to within a certain degree of accuracy and can thus be treated as (possibly a very large number of) ordered discrete values. Therefore, for the purposes of this project, continuous and ordered discrete values are treated equally and termed linear values. In practice also, linear values can often be grouped into meaningful ranges and treated in the same way as unordered discrete values. This has, in fact, been done with the contact lens data, in which attributes *a*, *b* and *d* all normally have linear values, but which for the purposes of this thesis have been grouped into meaningful ranges as described in section 6.1.1. Chapter 10 discusses the question of linear values in more detail.

Quinlan [46] distinguishes between two types of training set, the first consisting of instances which come from an existing database, e.g. a set of patient records in a medical domain, which may contain duplicate instances or omit uncommon ones, and the second consisting of instances prepared carefully by a domain expert to omit duplicate instances and include uncommon ones. He states that ID3 and ID3-like algorithms will deal with both kinds of training set 'in a satisfactory way'. However, the following

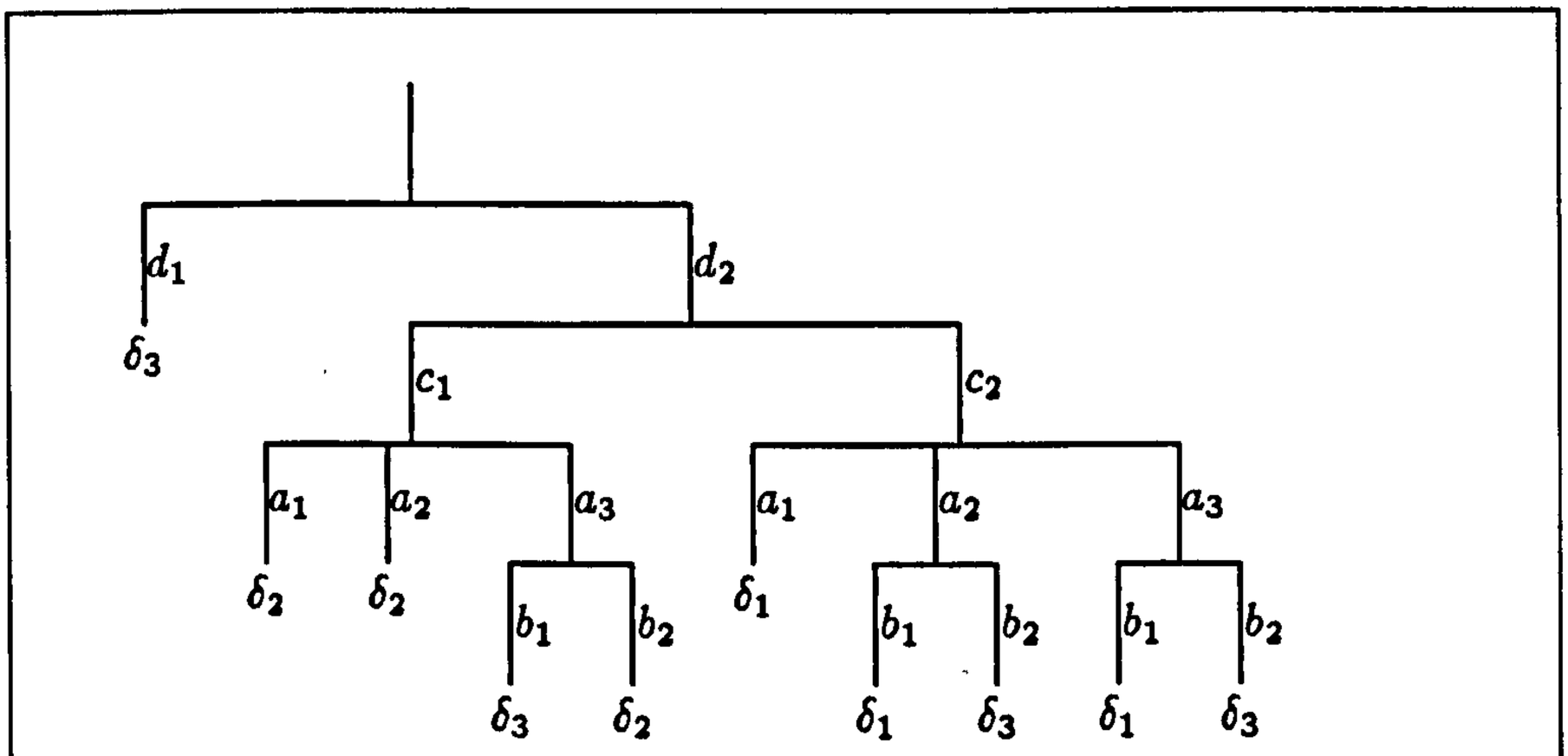


Figure 6.8 Instance no. 8 ($a_1 \& b_2 \& c_2 \& d_2 \rightarrow \delta_1$) duplicated 5 times

examples show that the inclusion of duplicate instances can easily affect the order in which attributes are selected, causing the induced decision trees to differ from each other depending on which instances are duplicated.

Each in turn of instances no. 8, no. 22 and no. 18 of the training set of table 6.1 were duplicated five times and ID3 was applied to each new training set of 29 instances. It was also applied to a training set of 34 instances in which instance no. 18 was duplicated ten times. The results are shown in figures 6.8, 6.9, 6.10 and 6.11, respectively. As can be seen, four different decision trees were induced and each of these was different from the original decision tree of figure 6.1. In fact, the original ID3 seems to be quite sensitive to duplicate instances — instance no. 22 only needs to be duplicated once or instance no. 8 twice for the induced decision trees to differ from the original.

The project reported in this thesis is concerned with examining ID3's induction strategy in detail, and using it as a model to develop a new algorithm which induces modular rules. As, in the initial stages of development, it is necessary to ensure that the algorithm itself is not a potential source of error, the training set used for development and testing must be totally

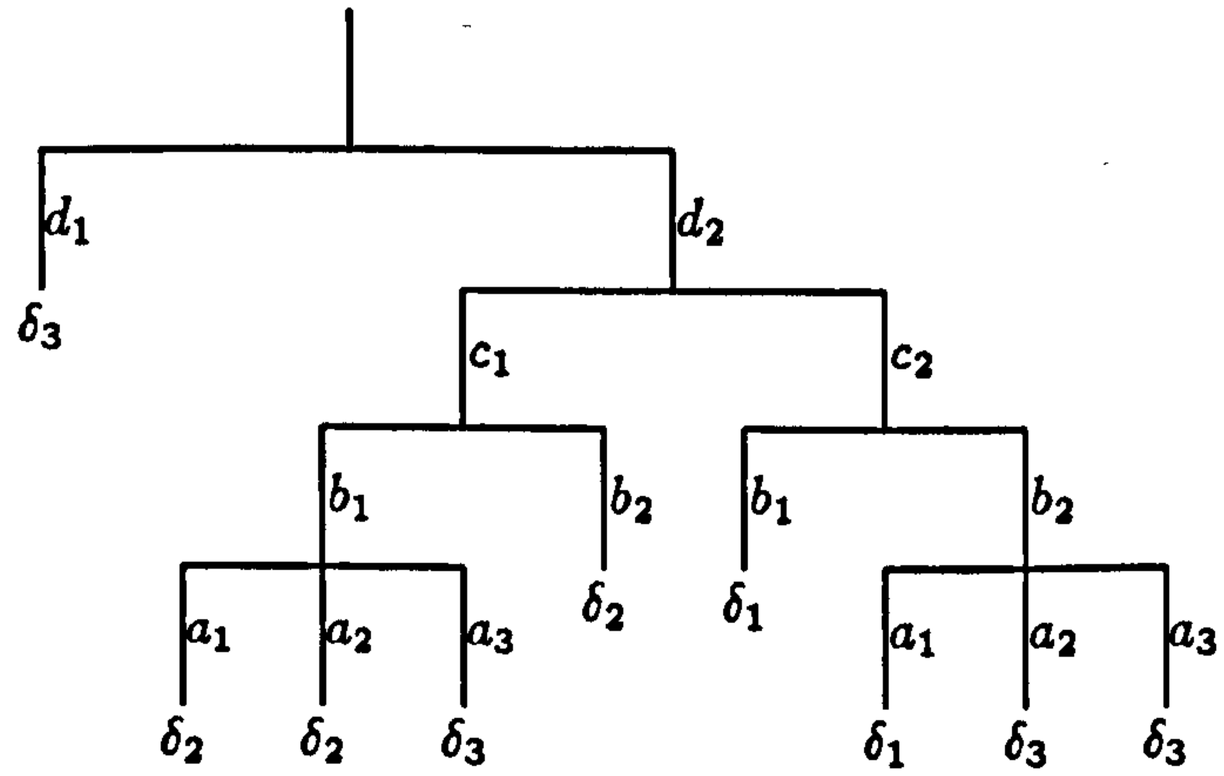


Figure 6.9 Instance no. 22 ($a_3 \& b_2 \& c_1 \& d_2 \rightarrow \delta_2$) duplicated 5 times

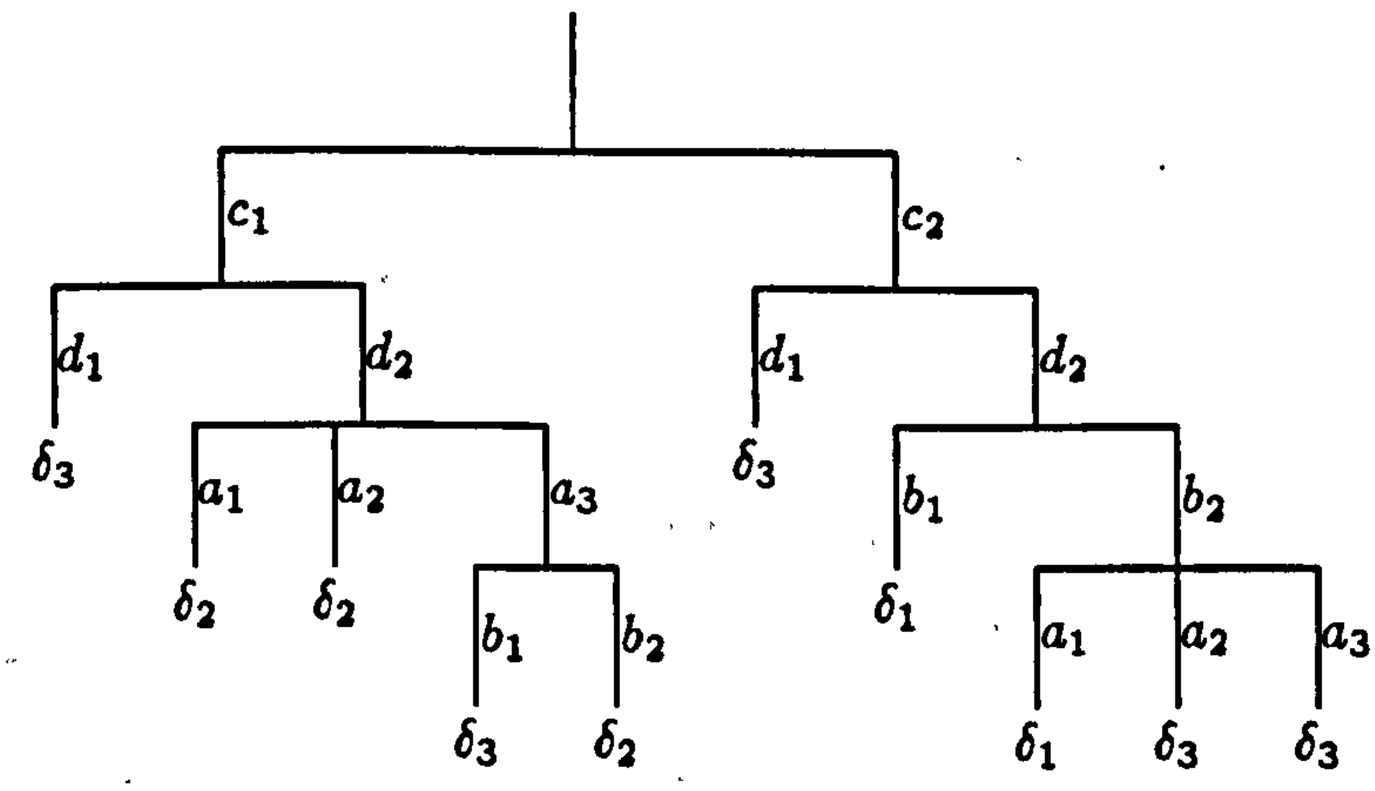


Figure 6.10 Instance no. 18 ($a_3 \& b_1 \& c_1 \& d_2 \rightarrow \delta_3$) duplicated 5 times

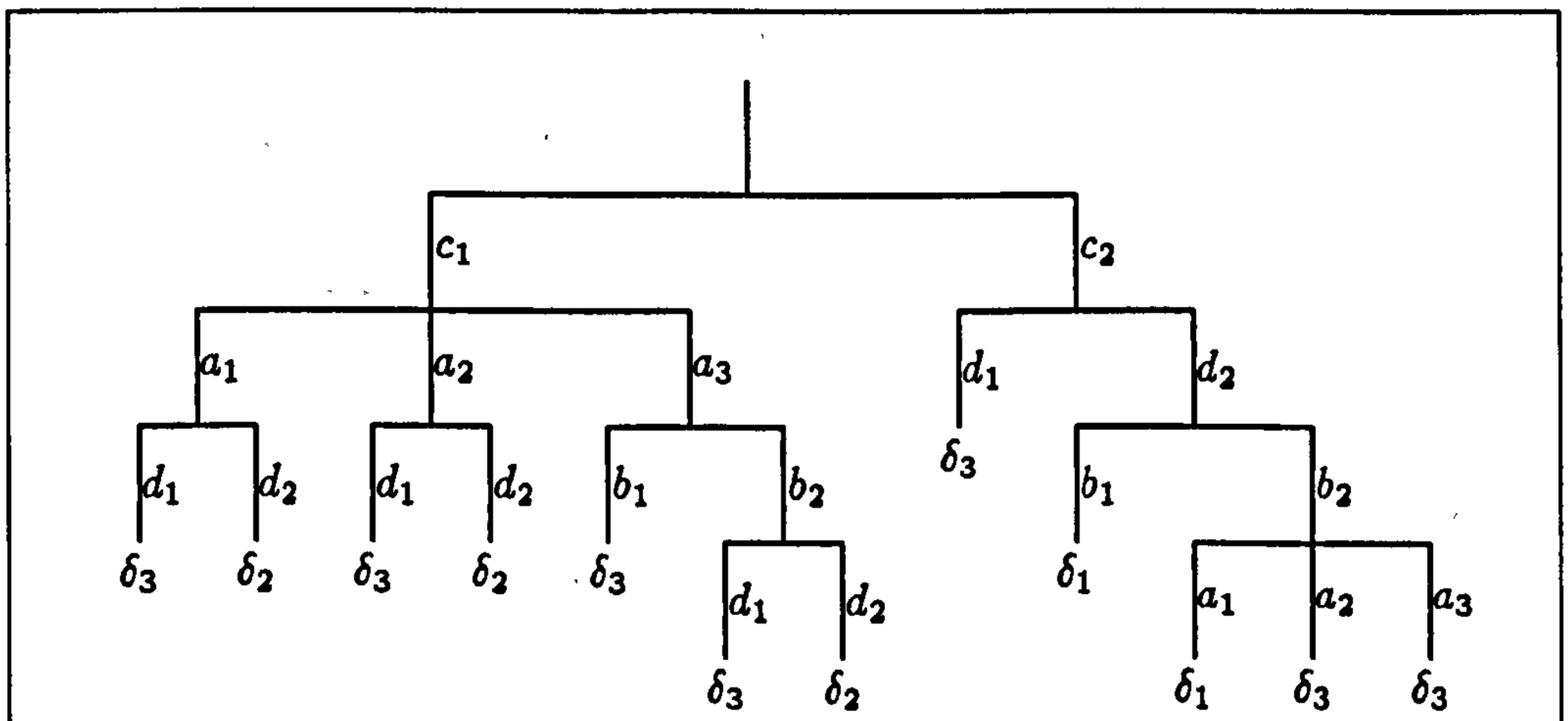


Figure 6.11 Instance no. 18 ($a_3 \& b_1 \& c_1 \& d_2 \rightarrow \delta_3$) duplicated 10 times

error-free. Only when the algorithm is known to perform well on such a training set, can it be tested on real-world training sets. The presence of noise in data, a problem which has been widely researched and documented by Quinlan and others [45,46,49], is a major source of error. An ideal training set should therefore be totally noise-free. The problems associated with noisy data are not addressed in this thesis.

Thus an ideal training set has the following characteristics:

- The set of attributes is adequate.
- The classes are specified in terms of attribute descriptions.
- The classes are mutually exclusive.
- The training set is complete.
- The values of all attributes are discrete.
- There are no duplicate instances.
- There is no noise.

Table 6.1 is an example of an ideal training set.

6.4 A perfect set of rules

A minimum requirement of any induction algorithm is that when applied to an ideal training set, it produces a rule or set of rules which classifies all possible instances correctly. One should, however, expect a little more than this. An ideal training set can itself be used by a simple table look-up algorithm to classify all instances correctly. In a perfect world an induction algorithm should ideally produce a perfect set of rules, i.e. a set of rules which not only classifies all possible instances correctly but which captures the underlying decision or classification strategy. The rules which are induced should be the rules which a domain expert uses when deciding how an instance should be classified. They should contain all the information necessary for classification and no redundant information.

However the world is not perfect. Incomplete training sets, duplicate instances and noise are potential sources of error or variability in the results. The aim of induction, therefore, should be to produce rules in which these errors are minimized. Only then can they be used with a reasonable degree of confidence to predict the class of unseen instances.

If the induction algorithm itself is a potential source of error, i.e. it produces rules which are less than perfect from an ideal training set, confidence in those rules will be reduced when it is applied to a training set taken from the real world. The training set of table 6.1 is an ideal training set as described in the previous section. It contains all the necessary information for a perfect set of rules to be deduced, and no redundant or misleading information. Section 6.5 describes why ID3, designed to induce decision trees, cannot always produce a perfect set of rules even from a training set which is ideal. In Chapter 7 the algorithm is examined in detail and used as a model for a new algorithm (called PRISM) which overcomes the problem.

6.5 Limitations of decision trees

One of the principal features of rule-based expert systems is that the modularity of the rules typically enables a knowledge base to be easily updated or

modified. It also provides a means for explanation. There is a requirement, therefore, that rules should be both modular and comprehensible, whether they are elicited from experts or automatically induced from examples.

Although ID3 has been proved to be computationally efficient [12,38,41], it produces its output in the form of a decision tree (e.g. figure 6.1). This decision tree representation of rules has a number of disadvantages. Firstly, decision trees are extremely difficult to manipulate — to extract information about any single classification it is necessary to examine the complete tree, a problem which is only partially resolved by trivially converting the tree into a set of individual rules, as the amount of information contained in some of these will often be more than can easily be assimilated. More importantly, there are rules that cannot easily be represented by trees.

Consider, for example, the following rule set:

$$\text{Rule 1} : a_1 \wedge b_1 \rightarrow \delta_1$$

$$\text{Rule 2} : c_1 \wedge d_1 \rightarrow \delta_1$$

Suppose that Rules 1 and 2 cover all instances of class δ_1 and all other instances are of class δ_2. These two rules cannot be represented by a single decision tree as the root node of the tree must split on a single attribute, and there is no attribute which is common to both rules. The simplest decision tree representation of the set of instances covered by these rules would necessarily add an extra term to one of the rules, which in turn would require at least one extra rule to cover instances excluded by the addition of that extra term. The complexity of the tree would depend on the number of possible values of the attributes selected for partitioning. For example, let the four attributes, a, b, c and d each have three possible values, 1, 2 and 3, and let attribute a be selected for partitioning at the root node. The simplest decision tree representation of Rules 1 and 2 above is shown in figure 6.12. The paths relating to class δ_1 can be listed as follows:

1. $a_1 \wedge b_1 \rightarrow \delta_1$

2. $a_1 \wedge b_2 \wedge c_1 \wedge d_1 \rightarrow \delta_1$

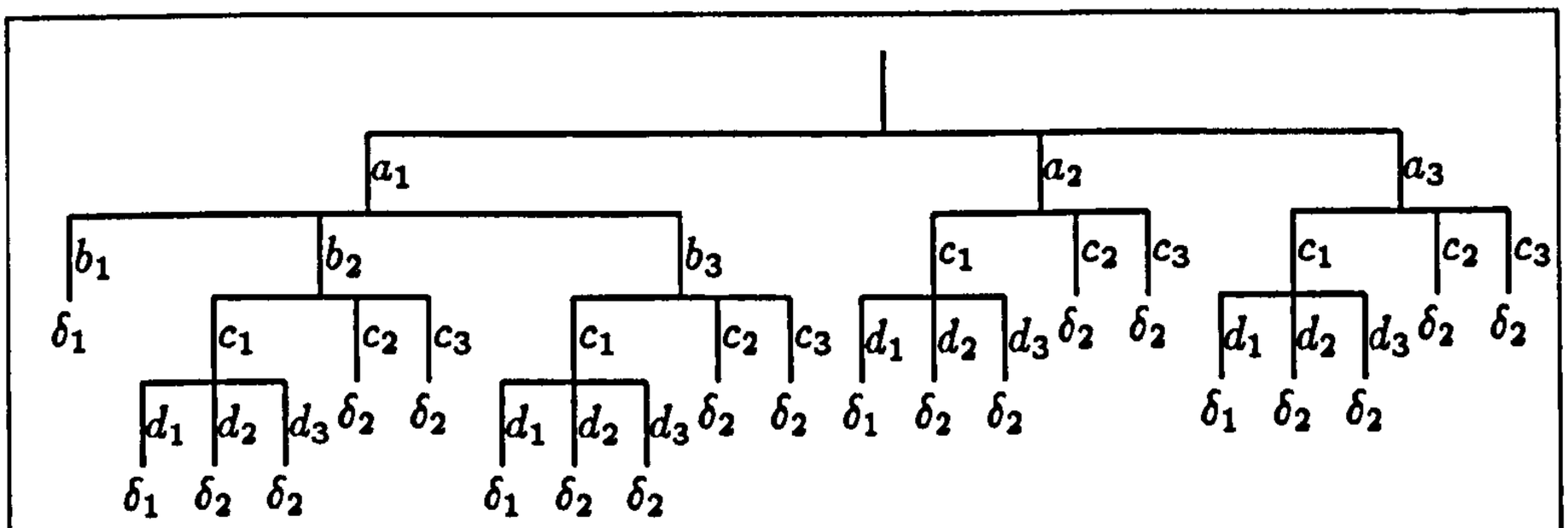


Figure 6.12 Decision tree representation of Rules 1 and 2

3. $a_1 \wedge b_3 \wedge c_1 \wedge d_1 \rightarrow \delta_1$

4. $a_2 \wedge c_1 \wedge d_1 \rightarrow \delta_1$

5. $a_3 \wedge c_1 \wedge d_1 \rightarrow \delta_1$

Clearly, the consequence of forcing a simple rule set into a decision tree representation is that the individual rules, when extracted from the tree, are often too specific (i.e. they reference attributes which are irrelevant). This makes them highly unsuitable for use in many domains, as is illustrated in the following example.

Suppose the decision tree in figure 6.1 was used as the knowledge base for an expert system advising on contact lens suitability, and suppose the patient requiring contact lenses was a presbyope with high hypermetropia and astigmatism (attributes a_3 & b_2 & c_2). The optician would know immediately from the age of the patient and her prescription that she was not a suitable candidate for contact lens wear (a decision taking about 30 seconds to make and costing the patient nothing). The expert system, however, would be unable to make a decision without the result of a tear production rate test (attribute d). This test is normally carried out as part of a contact lens consultation requiring a lot of time and payment of a fee. Having spent all this time and money, it would be quite understandable if the patient became upset or angry on finding out that the consultation had been, after

all, unnecessary. The consequences could be even more serious if the expert system was a medical one and attribute *d* involved surgery.

Clearly, a decision tree in its unmodified form is most unsuitable for some domains, not only because it can be incomprehensible, but because in many cases its use would demand irrelevant information to be supplied, information that could be costly to obtain. Attempts have been made at modifying the algorithm to avoid this problem by assigning a 'cost' to each attribute. Attempts have also been made [1,16] at converting decision trees into simple rule sets by identifying and removing redundant nodes, or by incorporating extra information which enables the user to focus on only relevant parts of the tree, but the problem is not an easy one to solve, particularly for very large and complex decision trees.

Although simplification of the trees is possible by identifying common branches or parts of branches, the combinatorial explosion in the number of comparisons that have to be made as the complexity increases makes this method only feasible for small trees. Also, parts of a branch may be matched in different ways, and the question then arises as to which is the better generalization to make. This would involve either asking the expert, or using another rule induction program to induce new rules from the old ones.

The research described in this thesis has as its goal an alternative approach, in which only the relevant parts of a tree are induced, i.e. relevant branches or parts of branches are induced individually. This research has resulted in a new induction algorithm, PRISM, which uses an information theoretic approach to selecting relevant attributes. The theory underlying the algorithm was developed by investigating in detail the process of attribute selection used by Quinlan's ID3 (described in section 7.1), and deciding how relevant attributes could be identified and separated from irrelevant ones (sections 7.2 and 7.3). PRISM is described in detail in Chapter 8.

Chapter 7

Information theoretic approaches to induction

Chapter 6 discussed some of the disadvantages of having a decision tree representation of decision or classification rules. Although a decision tree can be converted to a set of individual rules simply by listing each branch separately, many of these rules may still contain redundant terms. This chapter describes first how ID3 uses information theory for selecting attributes to partition the training set (section 7.1) and then how this process can be used as a model for an information theoretic approach to selecting non-redundant terms for modular rules (sections 7.2 and 7.3). The processes are described using the contact lens classification problem and ideal training set introduced in Chapter 6.

7.1 ID3's information theoretic approach

Originally, Quinlan chose to use a complexity estimate (see section 5.2) to select the 'best' attribute for partitioning. Later, this was abandoned in favour of a formula derived from information theory, for which the decision tree is thought of as a source of a message, and the amount of information conveyed by this message is related to the complexity of the tree. When the tree is being formed, at each node the attributes can be tested for expected information gain in the resulting tree if that attribute were selected for partitioning. That attribute is selected which minimizes entropy, thus

maximizing average information gain.

The procedure is described in detail in the following subsections.

7.1.1 Entropy

The training set can be thought of as a discrete information system, i.e. it contains a number of discrete messages (values of attributes) which impart some information about an event (classification). The entropy of a set of events has been defined as a measure of the 'freedom of choice' involved in the selection of the event, or the 'uncertainty' associated with this selection [20,27,51]. Let S be an ideal training set as described in Chapter 6. Because it is ideal, each instance in S is classified correctly and uniquely, i.e. there is no uncertainty about the classification. The entropy of S is 0. The entropy of a decision tree or rule set, which fully describes S is also 0, but in most cases the decision tree is a generalization of S, which implies that some information offered by the training set is redundant. ID3 tries to reduce this redundant information as much as possible (and thus find the least complex decision tree which fully describes the training set) by partitioning S into the smallest possible number of subsets, each of which can be described by a set of features (attribute-value pairs) whose entropy is 0.

If all that is known about the classifications is their probabilities of occurrence, $p(\delta_i\,;\, i = 1,2,3)$, then the entropy of the set of classifications,

$$H = -\sum_i p(\delta_i) \log_2 p(\delta_i) \text{ bits.} \tag{7.1}$$

For the contact lens classification problem,

$$H = -p(\delta_1)\log_2 p(\delta_1) - p(\delta_2)\log_2 p(\delta_2) - p(\delta_3)\log_2 p(\delta_3) \text{ bits.}$$

The probabilities of occurrence of each of the classifications are

$$\begin{aligned} p(\delta_1) &= 4/24 \\ p(\delta_2) &= 5/24 \\ p(\delta_3) &= 15/24 \end{aligned}$$

Thus,

$$\begin{aligned} H &= -\frac{4}{24}\log_2\left(\frac{4}{24}\right) - \frac{5}{24}\log_2\left(\frac{5}{24}\right) - \frac{15}{24}\log_2\left(\frac{15}{24}\right) \\ &= 0.4308 + 0.4715 + 0.4238 \\ &= 1.3261 \text{ bits.} \end{aligned} \tag{7.2}$$

The induction algorithm iteratively partitions the training set in such a way as to reduce this entropy by the maximum amount at each iteration, and continues until the entropy is 0.

7.1.2 Reducing entropy

If the training set, S, is divided according to the values of some attribute, α, then unless the classification, δ, is completely independent of α, the values will contain some information about δ. The total entropy of the subsets is known as the conditional entropy of S with known α, $H(S|\alpha)$. Let $p(\alpha_x)$ be the probability that attribute α has value x, and let $p(\delta_n \cap \alpha_x)$ be the probability that the classification is δ_n and the value of α is x. Then

$$H(S|\alpha) = H(S \cap \alpha) - H(\alpha) \tag{7.3}$$

where

$$H(S \cap \alpha) = -\sum_x \sum_n p(\delta_n \cap \alpha_x) \log_2 p(\delta_n \cap \alpha_x) \tag{7.4}$$

and

$$H(\alpha) = -\sum_x p(\alpha_x) \log_2 p(\alpha_x) \tag{7.5}$$

By performing this calculation for each attribute, it is possible to minimize the entropy of S by dividing it into subsets according to the values of that attribute for which $H(S|\alpha)$ is minimum.

The calculation can be simplified by using a frequency table, for example for attribute a:

no. of instances referencing	a_1	a_2	a_3	total
δ_1	2	1	1	4
δ_2	2	2	1	5
δ_3	4	5	6	15
total	8	8	8	24

$$
\begin{aligned}
H(S|a) &= H(S \cap a) - H(a) \\
&= -\sum_x \sum_n p(\delta_n \cap a_x) \log_2 p(\delta_n \cap a_x) + \sum_x p(a_x) \log_2 p(a_x) \\
&= -3 \times \frac{2}{24} \log_2 \left(\frac{2}{24}\right) - 3 \times \frac{1}{24} \log_2 \left(\frac{1}{24}\right) - \frac{4}{24} \log_2 \left(\frac{4}{24}\right) \\
&\quad -\frac{5}{24} \log_2 \left(\frac{5}{24}\right) - \frac{6}{24} \log_2 \left(\frac{6}{24}\right) + 3 \times \frac{8}{24} \log_2 \left(\frac{8}{24}\right) \\
&= \frac{1}{24}(3 \times 8 \log_2 8 - 3 \times 2 \log_2 2 - 3 \times \log_2 1 - 4 \log_2 4 \\
&\quad - 5 \log_2 5 - 6 \log_2 6) \\
&= 1.2867 \text{ bits.} \qquad (7.6)
\end{aligned}
$$

Similarly,

$$H(S|b) = 1.2867 \text{ bits.} \qquad (7.7)$$

$$H(S|c) = 0.9491 \text{ bits.} \qquad (7.8)$$

$$H(S|d) = 0.7773 \text{ bits.} \qquad (7.9)$$

Therefore, the entropy of S can be reduced by the greatest amount by dividing S according to the values of attribute d. Two subsets are formed, each of which is then further subdivided in the same way until the entropy of each subset is 0, i.e. all instances in the subset belong to the same classification. The final decision tree is shown in Figure 6.1.

7.2 The problem in focus

The above procedure evolved directly from Hunt's early work on concept learning and in particular from his CLS9 experiment (see section 5.1) in

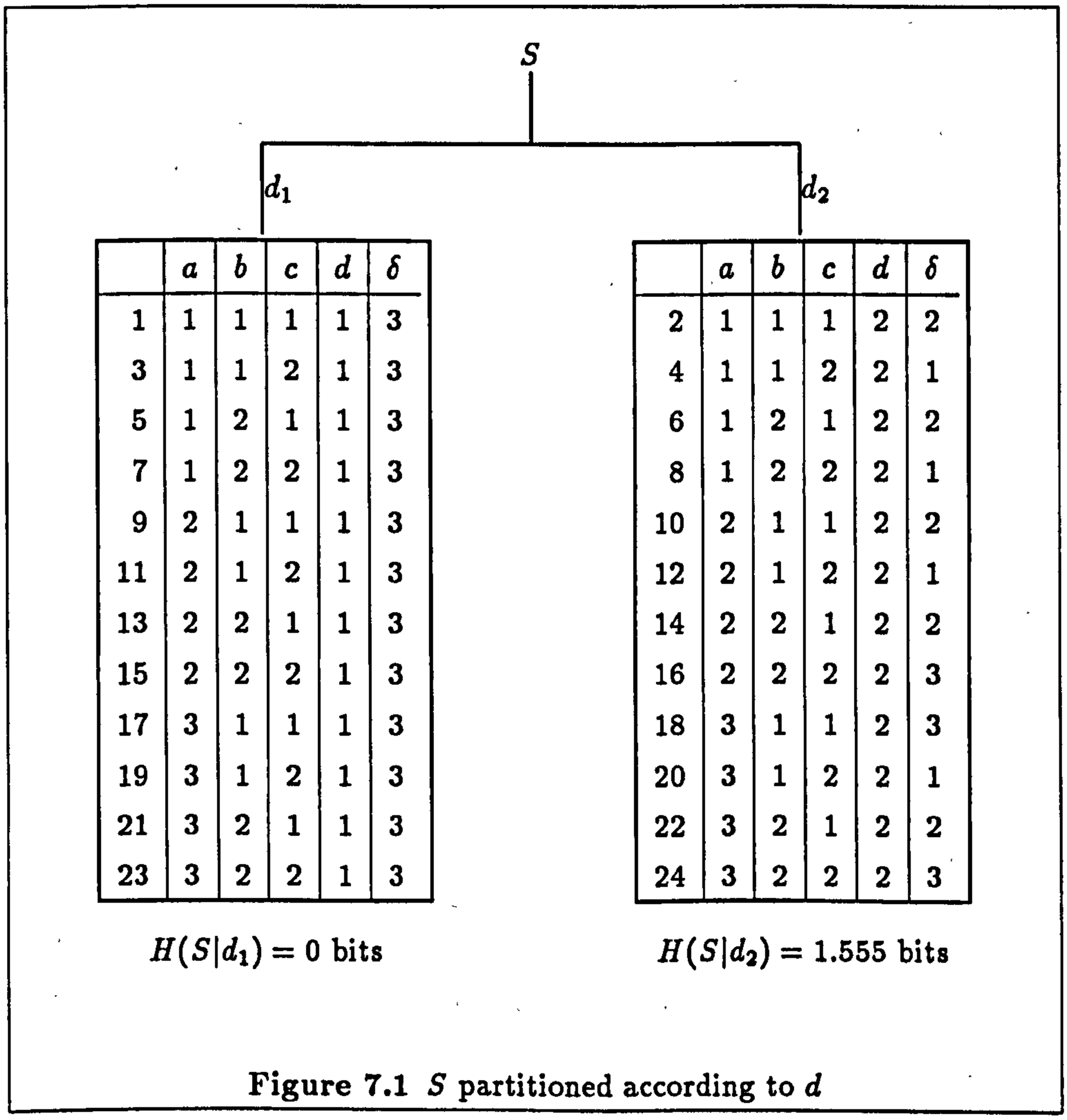
which he introduced a formula for defining the 'cost' of selecting an attribute for partitioning the tree. This cost was an average cost which assumed that all values of a particular attribute were equally relevant, an assumption which was valid for the preceding experiments in which all attributes had been binary. However, the main cause of the problem described in section 6.5 is either that an attribute is highly relevant to only one classification and irrelevant to the others, or that only one value of the attribute is relevant. For example, the attribute d in the contact lens problem is highly relevant to the classification δ_3, *if its value is 1*, and because of this, it is selected for partitioning the training set, for which all its values are used.

Figure 7.1 shows the decision tree after S has been partitioned according to the values of attribute d. It can be seen that although the entropy of the branch d_1 has been reduced to 0, the entropy of the branch d_2 has actually increased to 1.555 bits. Attribute d was chosen because ID3 minimizes the *average entropy* of the training set, or alternatively, it maximizes the *average* amount of information contributed by an attribute to the determination of *any* classification.

In order to eliminate the use of irrelevant values of attributes and attributes which are irrelevant to a classification, the algorithm needs to maximize the *actual* amount of information contributed by knowing the value of the attribute to the determination of a *specific* classification.

7.3 Induction of modular classification rules

The information theoretic approach to inducing decision trees described in section 7.1 can be modified fairly readily to enable the induction of modular classification rules, thus reducing the problems of incomprehensibility and irrelevance. Each induced rule comprises a conjunction of terms forming the premise and the classification which applies if the premise is satisfied. These rules can be induced by considering each possible term in turn and selecting the term (attribute-value pair) which maximizes information gain to partition the training set of instances. Thus emphasis is placed on calcu-



	a	b	c	d	δ
1	1	1	1	1	3
3	1	1	2	1	3
5	1	2	1	1	3
7	1	2	2	1	3
9	2	1	1	1	3
11	2	1	2	1	3
13	2	2	1	1	3
15	2	2	2	1	3
17	3	1	1	1	3
19	3	1	2	1	3
21	3	2	1	1	3
23	3	2	2	1	3

$H(S|d_1) = 0$ bits

	a	b	c	d	δ
2	1	1	1	2	2
4	1	1	2	2	1
6	1	2	1	2	2
8	1	2	2	2	1
10	2	1	1	2	2
12	2	1	2	2	1
14	2	2	1	2	2
16	2	2	2	2	3
18	3	1	1	2	3
20	3	1	2	2	1
22	3	2	1	2	2
24	3	2	2	2	3

$H(S|d_2) = 1.555$ bits

Figure 7.1 S partitioned according to d

lating the information contributed by an attribute-value pair to knowing a particular classification.

7.3.1 Calculating information content

As stated at the beginning of section 7.1.1, the values of attributes can be thought of as discrete messages in a discrete information system. Now, the amount of information about an event in a message i,

$$I(i) = \log_2 \left(\frac{\text{probability of event after the message is received}}{\text{probability of event before the message is received}} \right) \text{bits.}$$

The rest of this chapter describes how this definition (from standard information theory [20,27,51]) can be applied to the problem of inducing modular rules. The theory proposed here has been embodied in a new rule induction program, PRISM, which is described in Chapter 8.

The training set, S, contains 4 instances belonging to class δ_1, 5 belonging to class δ_2 and 15 to class δ_3. Therefore, the probability of an instance belonging to class δ_1, $p(\delta_1)$ is 4/24 and thus if the message i was δ_1 (i.e. the class is δ_1) then the amount of information received in this message,

$$I(\delta_1) = \log_2\left(\frac{1}{p(\delta_1)}\right) = -\log_2\left(\frac{4}{24}\right) = 2.585 \text{ bits.} \qquad (7.10)$$

Similarly, the amount of information received in the message δ_2,

$$I(\delta_2) = \log_2\left(\frac{1}{p(\delta_2)}\right) = -\log_2\left(\frac{5}{24}\right) = 2.263 \text{ bits.} \qquad (7.11)$$

and in the message δ_3,

$$I(\delta_3) = \log_2\left(\frac{1}{p(\delta_3)}\right) = -\log_2\left(\frac{15}{24}\right) = 0.678 \text{ bits.} \qquad (7.12)$$

Thus the lower the probability of occurrence of an event, the more information we receive if we are told that the event has occurred.

Now, if the message received was that attribute d has value 1, the amount of information received in this message about δ_3,

$$I(\delta_3|d_1) = \log_2\left(\frac{p(\delta_3|d_1)}{p(\delta_3)}\right) \text{ bits.} \qquad (7.13)$$

where $p(\delta_3|d_1)$ is the probability of δ_3 given that the value of d is 1.

For S, $p(\delta_3|d_1) = 1$, therefore

$$I(\delta_3|d_1) = \log_2\left(\frac{1}{p(\delta_3)}\right) = 0.678 \text{ bits.} \qquad (7.14)$$

Thus knowing that attribute d has value 1 contributes 0.678 bits of information to the belief that an instance belongs to class δ_3.

If, on the other hand, the message was that attribute d has value 2, then the amount of information received about δ_3,

$$I(\delta_3|d_2) = \log_2\left(\frac{p(\delta_3|d_2)}{p(\delta_3)}\right) = \log_2\left(\frac{3/12}{15/24}\right) = -1.322 \text{ bits.} \qquad (7.15)$$

The minus sign indicates that knowing that the value of d is 2 makes it less certain that an instance belongs to δ_3 than if the value of d was unknown. d_2 is therefore not a good choice for describing δ_3.

If an attribute-value pair, α_x, and a classification, δ_n, are mutually exclusive, $p(\delta_n|\alpha_x) = 0$ and $I(\delta_n|\alpha_x) = \log_2 0 = -\infty$. Thus knowing that the value of α is x indicates that the instance definitely does not belong to class δ_n.

If α_x and δ_n are completely independent, then $p(\delta_n|\alpha_x) = p(\delta_n)$ and $I(\delta_n|\alpha_x) = \log_2 1 = 0$, i.e. the fact α_x contributes no information to the belief that the class is δ_n.

7.3.2 Maximizing information gain

The task of an induction algorithm should be to find the attribute-value pair, α_x, which contributes the most information about a specified classification, δ_n, i.e. for which $I(\delta_n|\alpha_x)$ is maximum. Now,

$$I(\delta_n|\alpha_x) = \log_2 \left(\frac{p(\delta_n|\alpha_x)}{p(\delta_n)} \right) \text{bits.} \tag{7.16}$$

but $p(\delta_n)$ is the same for all α_x, and thus it is only necessary to find the α_x for which $p(\delta_n|\alpha_x)$ is maximum.

The values of $p(\delta_n|\alpha_x)$ for all α_x and $n = 1$ are listed in table 7.1a. There are two candidates for 'best' α_x. These are c_2 and d_2. For c_2, chosen arbitrarily, the information gain,

$$I(\delta_1|c_2) = \log_2 \left(\frac{p(\delta_1|c_2)}{p(\delta_1)} \right) = \log_2 \left(\frac{4/12}{4/24} \right) = 1 \text{ bit.} \tag{7.17}$$

Had d_2 been chosen, the information gain would also have been 1 bit. Repeating the process now on a subset of S which contains only those instances which have value 2 for attribute c, it can be seen from table 7.1b that $p(\delta_1|\alpha_x)$ has the highest value for d_2. The information gain (for this subset),

$$I(\delta_1|d_2) = \log_2 \left(\frac{p(\delta_1|d_2)}{p(\delta_1)} \right) = \log_2 \left(\frac{4/6}{4/12} \right) = 1 \text{ bit.} \tag{7.18}$$

α_x	$p(\delta_1\|\alpha_x)$		
a_1	$2/8$	$=$	0.25
a_2	$1/8$	$=$	0.125
a_3	$1/8$	$=$	0.125
b_1	$3/12$	$=$	0.25
b_2	$1/12$	$=$	0.083
c_1	0	$=$	0
c_2	$4/12$	$=$	0.333
d_1	0	$=$	0
d_2	$4/12$	$=$	0.333

Table 7.1a Selecting the first term

α_x	$p(\delta_1\|\alpha_x)$		
a_1	$2/4$	$=$	0.5
a_2	$1/4$	$=$	0.25
a_3	$1/4$	$=$	0.25
b_1	$3/6$	$=$	0.5
b_2	$1/6$	$=$	0.167
d_1	0	$=$	0
d_2	$4/6$	$=$	0.667

Table 7.1b Selecting the second term

α_x	$p(\delta_1\|\alpha_x)$		
a_1	$2/2$	$=$	1
a_2	$1/2$	$=$	0.5
a_3	$1/2$	$=$	0.5
b_1	$3/3$	$=$	1
b_2	$1/3$	$=$	0.333

Table 7.1c Selecting the third term

If the process is now repeated on the subset which contains only those instances which have value 2 for attribute c and value 2 for attribute d (table 7.1c), there is again a choice for 'best' α_x. Suppose the second of these, b_1, is selected[1]. Then

$$I(\delta_1|b_1) = \log_2\left(\frac{p(\delta_1|b_1)}{p(\delta_1)}\right) = \log_2\left(\frac{1}{4/6}\right) = 0.585 \text{ bits.} \qquad (7.19)$$

From equation 7.10, the information provided by the message δ_1 before any attributes are known = 2.585 bits.

The information provided by $c_2 = 1$ bit.

The information provided by d_2 when c_2 is known = 1 bit.

The information provided by b_1 when d_2 and c_2 are known = 0.585 bits.

Therefore, the information provided by $c_2 \wedge d_2 \wedge b_1$

$$= 1 + 1 + 0.585 = 2.585 \text{ bits.}$$

i.e. the message $c_2 \wedge d_2 \wedge b_1$ provides the same amount of information as the message δ_1.

Specialization of (i.e. adding more attribute-value pairs to) $c_2 \wedge d_2 \wedge b_1$ does not increase the information gain. All other attributes are irrelevant in this description as all instances containing $c_2 \& d_2 \& b_1$ belong to class δ_1 ($p(\delta_1|c_2 \wedge d_2 \wedge b_1) = 1$). The induced rule is therefore

$$c_2 \wedge d_2 \wedge b_1 \rightarrow \delta_1$$

and is known to be correct for S.

7.3.3 Modular rules

The decision tree at this stage of the induction process is shown in Figure 7.2. The algorithm has concentrated on building the shortest branch possible for the class δ_1. The remaining branches are not yet labelled, and the next step in the induction process is to identify the best rule for the set of instances which are not examples of the first rule. This is done by removing from S all instances containing $c_2 \& d_2 \& b_1$ and applying the algorithm to the remaining

[1]The reason for this choice is explained in section 8.4.1

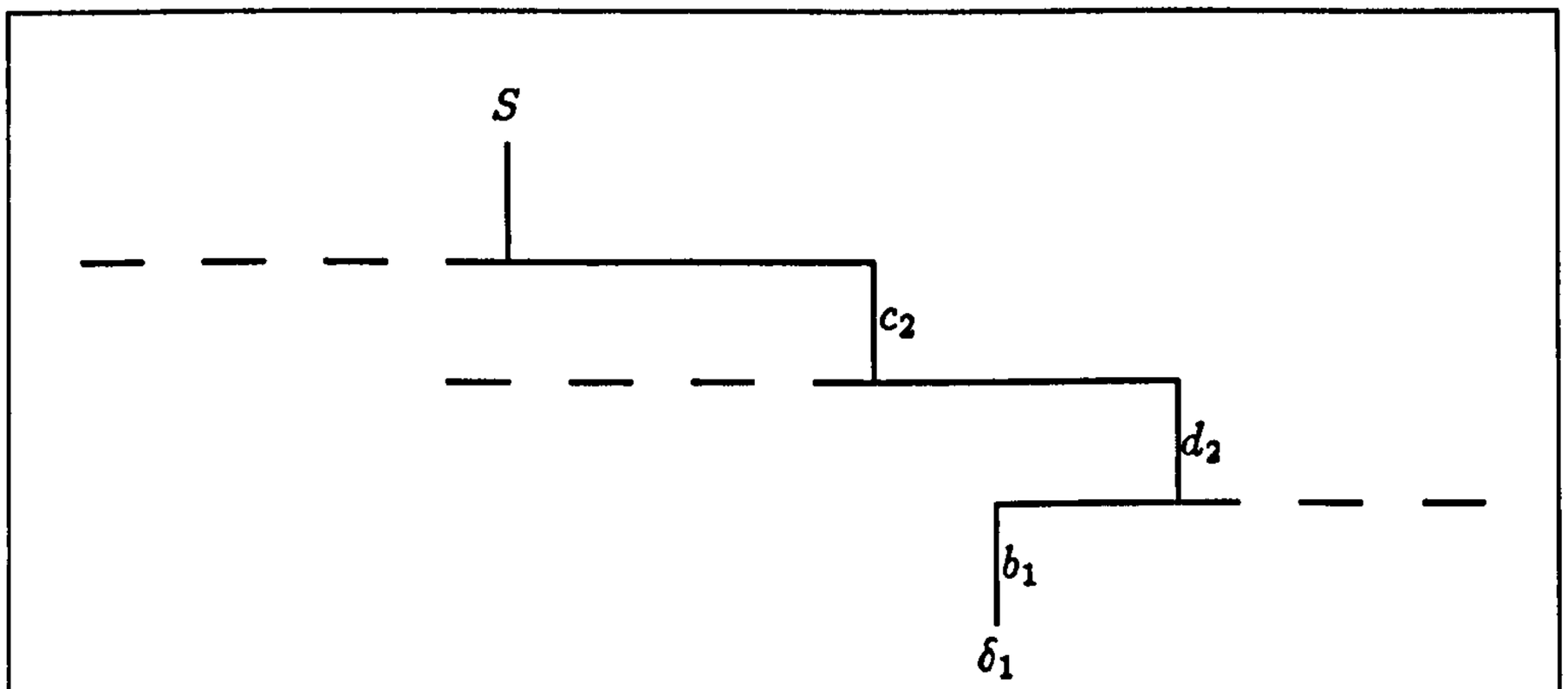


Figure 7.2 'Decision tree' after induction of the first rule

instances. If this is repeated until there are no instances of class δ_1 left in S, the result is not a single decision tree but a collection of individual branches. The whole process can then be repeated for each classification in turn, starting with the complete training set, S, each time.

The final output is an unordered collection of modular rules, each rule being as general as possible (but see section 8.4), thus ensuring that there are no redundant terms. The rule set for the optician's contact lens classification problem is as follows:

1. $c_2 \wedge d_2 \wedge b_1 \rightarrow \delta_1$

2. $a_1 \wedge c_2 \wedge d_2 \rightarrow \delta_1$

3. $c_1 \wedge d_2 \wedge b_2 \rightarrow \delta_2$

4. $c_1 \wedge d_2 \wedge a_1 \rightarrow \delta_2$

5. $c_1 \wedge d_2 \wedge a_2 \rightarrow \delta_2$

6. $d_1 \rightarrow \delta_3$

7. $b_2 \wedge c_2 \wedge a_2 \rightarrow \delta_3$

8. $b_2 \wedge c_2 \wedge a_3 \rightarrow \delta_3$

9. $a_3 \wedge b_1 \wedge c_1 \rightarrow \delta_3$

Although the number of rules in this set is the same as the number of leaf nodes in the decision tree (figure 6.1), five of the rules have had redundant terms removed. The presbyopic patient with high hypermetropia and astigmatism no longer needs to undergo an examination to be told that she is not suitable for contact lens wear (Rule 8).

Chapter 8

PRISM

The theory outlined in Chapter 7 has been embodied in a new rule induction program, PRISM. PRISM takes as input a training set entered as a file of ordered sets of attribute values, each set being terminated by a classification. Information about the attributes and classifications (e.g. name, number of possible values, list of possible values, etc.) is input from a separate file at the start of the program, and the results are output as individual rules for each of the classifications listed in terms of the described attributes.

8.1 The basic algorithm

The basic induction algorithm is essentially as described in section 7.3, namely:-
If the training set contains instances of more than one classification, then for each classification, δ_n, in turn:

Step 1: calculate the probability of occurrence, $p(\delta_n|\alpha_x)$, of the classification δ_n for each attribute-value pair α_x,

Step 2: select the α_x for which $p(\delta_n|\alpha_x)$ is a maximum and create a subset of the training set comprising all the instances which contain the selected α_x,

Step 3: repeat Steps 1 and 2 for this subset until it contains only instances of class δ_n. The induced rule is a conjunction of all

the attribute-value pairs used in creating the homogenous subset.

Step 4: remove all instances covered by this rule from the training set,

Step 5: repeat Steps 1 – 4 until all instances of class δ_n have been removed.

When the rules for one classification have been induced, the training set is restored to its initial state and the algorithm is applied again to induce a set of rules covering the next classification. As the classifications are considered separately, their order of presentation is immaterial. If all instances are of the same classification then that classification is returned as the rule, and the algorithm terminates.

8.2 The 'correctness' of rules

Given an ideal training set, as described in Chapter 6, the above algorithm produces a complete set of correct rules, i.e. a perfect set of rules, in most cases[1]. This is because PRISM concentrates on discovering the underlying rules which *cause* instances to be classified as δ_n by calculating $p(\delta_n|\alpha_x)$ for all α_x. The theory behind this can be explained in general terms as follows:

If the training set is complete and correct, the values of any attributes which are irrelevant to the class δ_n are equally distributed in the set of instances of class δ_n, i.e. $p(\alpha_1|\delta_n) = p(\alpha_2|\delta_n) = \cdots = p(\alpha_v|\delta_n)$ where v = number of possible values of attribute α. As a complete training set contains an equal distribution of the values of each attribute, i.e. $p(\alpha_1) = p(\alpha_2) = \cdots = p(\alpha_v)$, $p(\alpha_x|\delta_n) = p(\alpha_x)$ if α is irrelevant to δ_n. If on the other hand, α_x causes or partly causes an instance to be classified as δ_n, $p(\alpha_x|\delta_n)$ increases with respect to $p(\alpha_x)$ while $p(\alpha_y|\delta_n)(y \neq x)$ decreases with respect to $p(\alpha_y)$. It is possible for $p(\alpha_y|\delta_n)$ to decrease with respect to $p(\alpha_y)$ if α_y

[1]The exception to this is the case where all attributes are equally relevant (or irrelevant) to all classes and is described in section 8.4.

is a component of a rule describing δ_n, but only if α_x is a component of a rule or set of rules which covers a higher proportion of instances than that referencing α_y. In a complete training set, if the attributes are mutually exclusive, $p(\alpha_x|\delta_n)$ increases with respect to $p(\alpha_x)$ *only* if α_x is a component of a rule describing δ_n. If the attributes are not mutually exclusive, e.g. if say attribute β is partially dependent on attribute α such that if α has value 1, β can have only value 1, $p(\beta_1|\delta_n)$ will increase with respect to $p(\beta_1)$ if $p(\alpha_1|\delta_n)$ increases with respect to $p(\alpha_1)$, but $p(\beta_1|\delta_n)/p(\beta_1)$ will always be less than or equal to $p(\alpha_1|\delta_n)/p(\alpha_1)$, unless β_1 is independently relevant to δ_n.

Thus the α_x which has the highest value for $p(\alpha_x|\delta_n)/p(\alpha_x)$ *must* be a component of a rule describing δ_n. As $p(\alpha_x|\delta_n)/p(\alpha_x) = p(\delta_n|\alpha_x)/p(\delta_n)$ and $p(\delta_n)$ is constant, the α_x which has the highest value for $p(\delta_n|\alpha_x)$ must be a component of a rule describing δ_n.

8.3 PRISM compared with ID3

Although the basic induction algorithm used by PRISM is based on techniques employed by ID3, it is quite unlike ID3 in many respects. The major difference is that PRISM concentrates on finding only relevant values of attributes, while ID3 is concerned with finding the attribute which is most relevant overall, even though some values of that attribute may be irrelevant. All other differences between the two algorithms stem from this. ID3 divides a training set into homogenous subsets without reference to the class of this subset, whereas PRISM must identify subsets of a specific class. This has the disadvantage of slightly increased computational effort, but the advantage of an output in the form of modular rules rather than a decision tree.

This section demonstrates the performance of PRISM on a training set containing a large number of examples. The training set is provided by the King-Knight-King-Rook chess end-game on which Quinlan performed his original experiments [44]. The problem is to find a rule set which will

determine for each configuration of the four pieces, whether knight's side is lost two-ply in a black-to-move situation. Quinlan tackled the problem in stages, by first placing severe constraints on the number of allowable configurations of the pieces, and then gradually relaxing these constraints until he could apply his algorithm successfully to the original unrestricted problem. He identified a total of seven problems of increasing complexity. The training set described below is provided by the third of these problems.

There are seven attributes:

a: distance from black king to knight, values 1, 2 or 3,

b: distance from black king to rook, values 1, 2 or 3,

c: distance from white king to knight, values 1, 2 or 3,

d: distance from white king to rook, values 1, 2 or 3,

e: black king, knight, rook in line, values t or f,

f: rook bears on black king, values t or f,

g: rook bears on knight, values t or f.

There are two possible classifications – lost and safe, and the training set consists of 647 instances[2]. The decision tree produced by ID3 is shown in figure 8.1. It has 20 branches, and if these are trivially converted into separate rules, there are a total of 105 terms. In contrast, the rule set produced by PRISM has 15 rules and 48 terms:

1. $e_f \rightarrow$ safe

2. $f_f \rightarrow$ safe

3. $g_f \rightarrow$ safe

4. $b_1 \wedge d_2 \rightarrow$ safe

5. $b_1 \wedge d_3 \rightarrow$ safe

6. $a_1 \wedge c_2 \rightarrow$ safe

[2]There is one combination of the seven attributes (a_1 & b_1 & c_1 & d_1 & e_t & f_t & g_t) which is illegal and therefore not included in the training set.

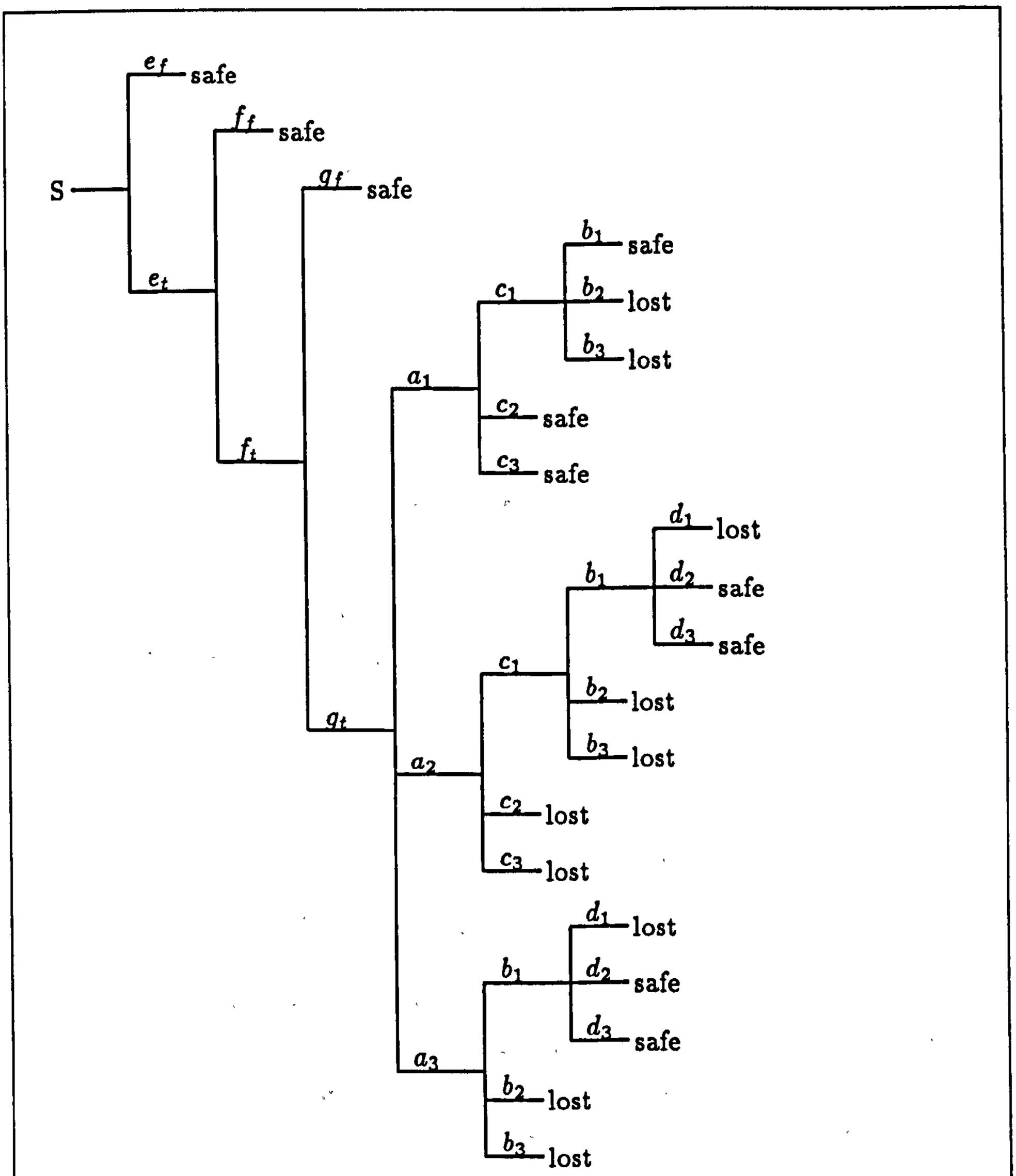


Figure 8.1 Decision tree for Quinlan's third problem

7. $a_2 \wedge c_2 \rightarrow$ safe

8. $a_1 \wedge c_3 \rightarrow$ safe

9. $a_2 \wedge c_3 \rightarrow$ safe

10. $a_3 \wedge b_2 \wedge e_t \wedge f_t \wedge g_t \rightarrow$ lost

11. $b_3 \wedge c_1 \wedge e_t \wedge f_t \wedge g_t \rightarrow$ lost

12. $a_3 \wedge b_3 \wedge e_t \wedge f_t \wedge g_t \rightarrow$ lost

13. $b_2 \wedge c_1 \wedge e_t \wedge f_t \wedge g_t \rightarrow$ lost

14. $a_3 \wedge b_1 \wedge d_1 \wedge e_t \wedge f_t \wedge g_t \rightarrow$ lost

15. $a_2 \wedge b_1 \wedge c_1 \wedge d_1 \wedge e_t \wedge f_t \wedge g_t \rightarrow$ lost

Both the decision tree and the above rule set classify all 647 instances correctly, but an expert system using the decision tree as its knowledge base would require significantly more tests to be performed.

There is also one less obvious difference between the outputs, which is that the decision tree would classify the illegal instance (a_1 & b_1 & c_1 & d_1 & e_t & f_t & g_t) as safe, whereas the rule set produced by PRISM is unable to classify it.

8.4 The use of heuristics

ID3 and PRISM are similar in that they both employ an information theoretic approach to discovering disjunctive rules by grouping together sets of instances with similar features. Consequently, they both encounter similar difficulties in certain circumstances. In particular, there is the problem of which attribute or attribute-value pair to choose when the results of the respective calculations indicate that there are two or more which are equal. In ID3 the choice is immaterial because the objective is to reduce entropy at the maximal rate and this is achieved equally well whichever attribute is chosen. On the other hand, if the wrong choice is made in PRISM, the

result is that an irrelevant attribute-value pair may be chosen. Fortunately, this situation can often be avoided by incorporating some heuristics in the basic algorithm.

8.4.1 Opting for generality I

If there are two or more rules describing a classification, PRISM tries to induce the most general rule first. The rationale behind this is that the more general a rule is then the less likely it is to reference an irrelevant attribute. Thus where there is a choice of attribute-value pairs, PRISM selects that attribute-value pair which has the highest frequency of occurrence in the set of instances being considered. Referring back to table 7.1c in section 7.3.2 (selection of a third term for the first rule for class δ_1), it can be seen that the attribute-value pairs a_1 and b_1 both offer an equal information gain. PRISM selects b_1 because the resulting rule covers three instances, whereas the rule resulting from the selection of a_1 would only cover two instances. Thus the rule $c_2 \wedge d_2 \wedge b_1 \rightarrow \delta_1$ is more general than $c_2 \wedge d_2 \wedge a_1 \rightarrow \delta_1$. In this particular case, both rules are in fact equally correct, and so the order in which they are induced does not really matter, but opting for generality in this way has the advantage of reducing computational effort when there is a significant difference in the number of instances covered by each of the rules. Its true value, however, is realized when the training set is an incomplete one and there is a possibility that one potential rule is a specialization of another. In this situation PRISM *must* select the more general.

8.4.2 Opting for generality II

When both the information gain offered by two or more attribute-value pairs is the same and the numbers of instances referencing them is the same, PRISM selects the first. This is the only time that the order of input of the attributes affects the induction process, and in these cases it is still possible for an irrelevant attribute-value pair to be selected. To illustrate how PRISM copes with this situation, suppose there are four attributes, a, b, c and d, each having three possible values, 1, 2 and 3, and the rules to be

induced for class δ_1 are:

$$\text{Rule } 1 : c_1 \wedge d_1 \rightarrow \delta_1$$

$$\text{Rule } 2 : c_2 \wedge d_2 \rightarrow \delta_1$$

$$\text{Rule } 3 : c_3 \wedge d_3 \rightarrow \delta_1$$

Thus, attributes a and b are irrelevant to δ_1, whereas all values of attributes c and d are equally relevant. If the training set is complete, $p(\delta_1|\alpha_x)$ is the same for all α_x and PRISM selects a_1. The subset containing only instances which have value 1 for attribute a also presents the same problem: $p(\delta_1|\alpha_x)$ is equal for all α_x, so b_1 is selected, and so on. The result is the following set of rules:

$$\text{Rule } 1 : a_1 \wedge b_1 \wedge c_1 \wedge d_1 \rightarrow \delta_1$$

$$\text{Rule } 2 : a_2 \wedge b_1 \wedge c_1 \wedge d_1 \rightarrow \delta_1$$

$$\text{Rule } 3 : a_3 \wedge b_1 \wedge c_1 \wedge d_1 \rightarrow \delta_1$$

$$\text{Rule } 4 : b_2 \wedge a_1 \wedge c_1 \wedge d_1 \rightarrow \delta_1$$

$$\text{Rule } 5 : b_3 \wedge a_1 \wedge c_1 \wedge d_1 \rightarrow \delta_1$$

At this stage $p(\delta_1|\alpha_x)$ is greater for c_2, c_3, d_2 and d_3 than for any other attribute-value pair, so the next two rules are induced correctly:

$$\text{Rule } 6 : c_2 \wedge d_2 \rightarrow \delta_1$$

$$\text{Rule } 7 : c_3 \wedge d_3 \rightarrow \delta_1$$

The remaining instances all have value 1 for attribute c and value 1 for attribute d, so the final rule is

$$\text{Rule } 8 : c_1 \wedge d_1 \rightarrow \delta_1$$

Rules 1–5 are all specializations of Rule 8. To avoid this happening, PRISM first induces all rules for a classification and then selects the most general of these on the basis of i) the rule which covers the maximum number of instances, and ii) the rule which references the fewest attributes. The instances covered by this rule are removed from the training set, and PRISM

goes on to induce the remaining rules in the same way. For the above example, the result is that Rules 6 and 7 are induced first, and then Rule 8. These three rules account for all instances of class δ_1, so Rules 1–5 are discarded.

Although this iterative procedure is quite costly in terms of computational effort, it ensures (at least for a complete training set) that the induced rules are maximally general.

8.5 The training set — necessary requirements

Section 6.2 described some characteristics of a training set which must be present for ID3 to perform successfully, namely:

- the set of attributes must be adequate
- the classes must be specifiable in terms of attribute descriptions
- the classes must be mutually exclusive

The same characteristics are necessary for PRISM to perform successfully and are described again below, this time with reference to PRISM.

8.5.1 The set of attributes must be adequate

ID3 allows instances for which the attributes are found to be inadequate to remain unclassified. PRISM, however, attempts to find a set of rules which covers all instances in the training set. If the training set contains a pair of contradictory instances, i.e. a pair of instances which have the same values for the attributes but are of different classes, the result is that a pair of contradictory rules is induced. For example, all references to attribute a were again removed from the training set of table 6.1. PRISM was applied to the remaining data, resulting in the following rules:

1. $b_1 \wedge c_2 \wedge d_2 \rightarrow \delta_1$
2. $b_2 \wedge c_2 \wedge d_2 \rightarrow \delta_1$

3. $b_2 \wedge c_1 \wedge d_2 \rightarrow \delta_2$

4. $b_1 \wedge c_1 \wedge d_2 \rightarrow \delta_2$

5. $d_1 \rightarrow \delta_3$

6. $b_2 \wedge c_2 \wedge d_2 \rightarrow \delta_3$

7. $b_1 \wedge c_1 \wedge d_2 \rightarrow \delta_3$

These results are very similar to the decision tree of figure 6.2, except that branch B (unclassified) has been replaced by rules 4 and 7 above and branch E has been replaced by rules 2 and 6. The only rules which were induced correctly[3] (rules 1, 3 and 5) were those which do not need to reference attribute a.

PRISM uses all attributes in an attempt to discriminate between contradictory instances and, like ID3, may discover apparent relationships between irrelevant attributes and class, resulting in incorrect or misleading rules. Thus if PRISM is to be used successfully, the set of attributes *must* be adequate. At the very least, all contradictory instances must be removed from the training set.

8.5.2 The classes must be specifiable in terms of attribute descriptions

Like ID3, PRISM can only discover rules in terms of attribute descriptions. If applied to the complete training set for classifying rectangles (table 6.2, page 85) or any other similar domain, the result may often be a set of rules which is just a reproduction of the instances comprising the training set. PRISM does not and should not be expected to induce structural descriptions or relations between two or more attributes. Thus it is not suitable for a domain characterized by such features.

[3] when compared with the rules induced from the complete and correct training set

8.5.3 The classes must be mutually exclusive

Both ID3 and PRISM are based on finding attributes which distinguish between classes. If the classes are not mutually exclusive and there are instances which can be classified correctly in more than one way, the results of induction are similar to results obtained when the set of attributes is inadequate, as explained in sections 6.2.3 and 8.5.1.

8.6 Duplicate instances

The sensitivity of ID3 to duplicate instances was described in section 6.3, where it was shown that different decision trees could be induced simply by duplicating certain instances in a complete training set. The training set used was that of table 6.1 on page 78, from which four new training sets were created, in which:

1. Instance no. 8 was duplicated five times.

2. Instance no. 22 was duplicated five times.

3. Instance no. 18 was duplicated five times.

4. Instance no. 18 was duplicated ten times.

These same training sets were used to assess the sensitivity of PRISM to duplicate instances. It was found that in each case PRISM induced the same set of rules, and these rules were identical to those (listed on page 107) induced from the original training set. The only effect of duplicate instances is that the rules may be induced in a different order. This is because PRISM tends to induce the most general rules first, i.e. those covering the most instances. Thus for the training set of table 6.1, rule 1 ($b_1 \wedge c_2 \wedge d_2 \rightarrow \delta_1$) covers three instances and rule 2 ($a_1 \wedge c_2 \wedge d_2 \rightarrow \delta_1$) covers two instances. Rule 1 is induced first. By duplicating instance no. 8 ($a_1 \& b_2 \& c_2 \& d_2 \rightarrow \delta_1$) five times, the number of instances covered by rule 2 increases to seven, causing rule 2 to be induced first. However, as the rules are modular their

order is irrelevant. Thus the inclusion of duplicate instances has no effect on the results (apart from an increase in computation to achieve those results).

This is true in all cases. PRISM induces its rules by iteratively selecting the attribute-value pair α_x which has the highest value for $p(\delta_n|\alpha_x)$ (see sections 7.3 and 8.2). When an instance of class δ_n is duplicated, although it may no longer be true that $p(\alpha_x|\delta_n) = p(\alpha_x)$ if α is irrelevant to δ_n, the instance which is duplicated must be covered by a rule describing δ_n. As the number of instances containing α_x is increased by the same amount for all α_x which comprise the duplicated instance, it is still true that the α_x which has the highest value for $p(\alpha_x|\delta_n)/p(\alpha_x)$, and therefore the highest value for $p(\delta_n|\alpha_x)$, is a component of a rule describing δ_n.

Chapter 9

Induction from incomplete training sets

When PRISM is applied to a complete and correct training set, the resulting set of rules can confidently be expected to be complete and correct. When the training set is incomplete, this confidence is reduced. The smaller the proportion of instances in the training set, the more likely it is that the rule set will contain errors. Errors in the induction process arise for a number of reasons which can be best explained by example. The training set used for this purpose is described in section 9.1. Section 9.2 lists the rules induced by PRISM from about 20% of this training set and an analysis of how and why errors arise is given in section 9.3. PRISM has been enhanced to enable some of these errors to be reduced — this is described in detail in section 9.4. Section 9.5 gives a summary of the complete induction procedure and finally, section 9.6 contains an analysis of the performance of PRISM, compared with ID3, with particular reference to predictive power, i.e. the ability to induce rules which correctly classify unseen instances.

9.1 The training set

The training set (table 9.1) used for most of the experiments described in this chapter is an extension of the training set described in section 6.1.1 (table 6.1). Attribute *b*, spectacle prescription, has been modified to have three possible values: 1: myopia, 2: high hypermetropia and 3: low hypermetropia, and a fifth attribute, *e*: tear break-up time, has been added.

	value of attribute					decision [1]		value of attribute					decision [1]
	a	b	c	d	e	δ		a	b	c	d	e	δ
1	1	1	1	1	1	3	28	1	3	1	2	1	3
2	1	1	1	1	2	3	29	1	3	1	2	2	3
3	1	1	1	1	3	3	30	1	3	1	2	3	3
4	1	1	1	2	1	3	31	1	3	2	1	1	3
5	1	1	1	2	2	2	32	1	3	2	1	2	3
6	1	1	1	2	3	2	33	1	3	2	1	3	3
7	1	1	2	1	1	3	34	1	3	2	2	1	3
8	1	1	2	1	2	3	35	1	3	2	2	2	3
9	1	1	2	1	3	3	36	1	3	2	2	3	3
10	1	1	2	2	1	3	37	2	1	1	1	1	3
11	1	1	2	2	2	2	38	2	1	1	1	2	3
12	1	1	2	2	3	1	39	2	1	1	1	3	3
13	1	2	1	1	1	3	40	2	1	1	2	1	3
14	1	2	1	1	2	3	41	2	1	1	2	2	3
15	1	2	1	1	3	3	42	2	1	1	2	3	2
16	1	2	1	2	1	3	43	2	1	2	1	1	3
17	1	2	1	2	2	2	44	2	1	2	1	2	3
18	1	2	1	2	3	2	45	2	1	2	1	3	3
19	1	2	2	1	1	3	46	2	1	2	2	1	3
20	1	2	2	1	2	3	47	2	1	2	2	2	3
21	1	2	2	1	3	3	48	2	1	2	2	3	1
22	1	2	2	2	1	3	49	2	2	1	1	1	3
23	1	2	2	2	2	2	50	2	2	1	1	2	3
24	1	2	2	2	3	1	51	2	2	1	1	3	3
25	1	3	1	1	1	3	52	2	2	1	2	1	3
26	1	3	1	1	2	3	53	2	2	1	2	2	3
27	1	3	1	1	3	3	54	2	2	1	2	3	2

Table 9.1 Decision table for fitting contact lenses (part 1)

[1]The reader is asked not to be tempted to use this decision table to determine whether or not (s)he is suitable for contact lens wear as there are many factors, not mentioned here, which may radically influence the decision.

	value of attribute					decision[1]		value of attribute					decision[1]
	a	b	c	d	e	δ		a	b	c	d	e	δ
55	2	2	2	1	1	3	82	3	1	2	2	1	3
56	2	2	2	1	2	3	83	3	1	2	2	2	3
57	2	2	2	1	3	3	84	3	1	2	2	3	1
58	2	2	2	2	1	3	85	3	2	1	1	1	3
59	2	2	2	2	2	3	86	3	2	1	1	2	3
60	2	2	2	2	3	3	87	3	2	1	1	3	3
61	2	3	1	1	1	3	88	3	2	1	2	1	3
62	2	3	1	1	2	3	89	3	2	1	2	2	3
63	2	3	1	1	3	3	90	3	2	1	2	3	2
64	2	3	1	2	1	3	91	3	2	2	1	1	3
65	2	3	1	2	2	3	92	3	2	2	1	2	3
66	2	3	1	2	3	3	93	3	2	2	1	3	3
67	2	3	2	1	1	3	94	3	2	2	2	1	3
68	2	3	2	1	2	3	95	3	2	2	2	2	3
69	2	3	2	1	3	3	96	3	2	2	2	3	3
70	2	3	2	2	1	3	97	3	3	1	1	1	3
71	2	3	2	2	2	3	98	3	3	1	1	2	3
72	2	3	2	2	3	3	99	3	3	1	1	3	3
73	3	1	1	1	1	3	100	3	3	1	2	1	3
74	3	1	1	1	2	3	101	3	3	1	2	2	3
75	3	1	1	1	3	3	102	3	3	1	2	3	3
76	3	1	1	2	1	3	103	3	3	2	1	1	3
77	3	1	1	2	2	3	104	3	3	2	1	2	3
78	3	1	1	2	3	3	105	3	3	2	1	3	3
79	3	1	2	1	1	3	106	3	3	2	2	1	3
80	3	1	2	1	2	3	107	3	3	2	2	2	3
81	3	1	2	1	3	3	108	3	3	2	2	3	3

Table 9.1 Decision table for fitting contact lenses (part 2)

A full description of the data is reproduced below.

Class (decision) :

δ_1 : the patient should be fitted with hard contact lenses

δ_2 : the patient should be fitted with soft contact lenses

δ_3 : the patient should not be fitted with contact lenses

Attributes :

a : the age of the patient

1. young
2. pre-presbyopic
3. presbyopic

b : spectacle prescription

1. myopia
2. high hypermetropia
3. low hypermetropia

c : astigmatic

1. no
2. yes

d : tear production rate

1. reduced
2. normal

e : tear break-up time

1. $\leq$ 5 secs.
2. $>$ 5 secs., $\leq$ 10 secs.
3. $>$ 10 secs.

The rules induced by applying PRISM to a complete and correct training set are:

1. $b_1 \wedge c_2 \wedge d_2 \wedge e_3 \rightarrow \delta_1$

2. $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_3 \rightarrow \delta_1$

3. $b_2 \wedge c_1 \wedge d_2 \wedge e_3 \rightarrow \delta_2$

4. $a_1 \wedge b_1 \wedge d_2 \wedge e_2 \rightarrow \delta_2$

5. $a_1 \wedge b_2 \wedge d_2 \wedge e_2 \rightarrow \delta_2$

6. $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_3 \rightarrow \delta_2$

7. $a_2 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_3 \rightarrow \delta_2$

8. $d_1 \rightarrow \delta_3$

9. $b_3 \rightarrow \delta_3$

10. $e_1 \rightarrow \delta_3$

11. $a_3 \wedge e_2 \rightarrow \delta_3$

12. $a_2 \wedge e_2 \rightarrow \delta_3$

13. $a_3 \wedge b_2 \wedge c_2 \rightarrow \delta_3$

14. $a_3 \wedge b_1 \wedge c_1 \rightarrow \delta_3$

15. $a_2 \wedge b_2 \wedge c_2 \rightarrow \delta_3$

The decision tree induced by ID3 from the same training set is shown in figure 9.1.

9.2 PRISM applied to an incomplete training set

21 instances (just under 20%) were selected at random from the training set of table 9.1 and PRISM was applied to this new training set (shown in table 9.2). The following nine rules were induced:

A $a_1 \wedge b_1 \wedge d_2 \rightarrow \delta_1$

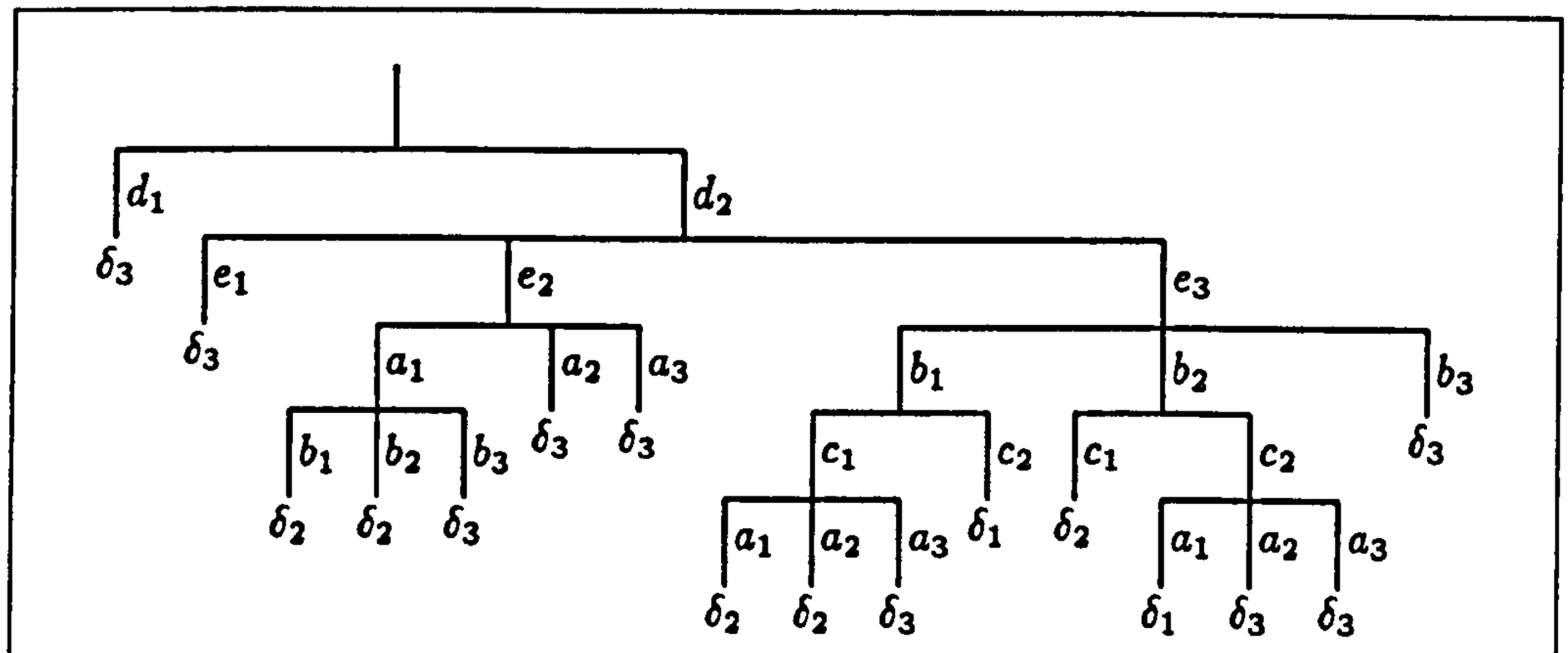


Figure 9.1 Decision tree for contact lens fitting problem

B $b_2 \wedge c_1 \wedge d_2 \wedge e_3 \rightarrow \delta_2$

C $a_1 \wedge b_2 \rightarrow \delta_2$

D $d_1 \rightarrow \delta_3$

E $e_1 \rightarrow \delta_3$

F $b_3 \rightarrow \delta_3$

G $a_2 \wedge c_2 \rightarrow \delta_3$

H $c_1 \wedge e_2 \rightarrow \delta_3$

I $a_3 \wedge b_1 \rightarrow \delta_3$

When these are compared with the complete set of correct rules (listed in section 9.1), it can be seen that rules B, D, E and F have been induced correctly (rules 3, 8, 10 and 9, respectively), rules C, G and I are generalizations of rules 5, 15 and 14 respectively and rules 2, 4, 6, 7 and 13 of the correct set have not been induced at all. Rule A is an incorrect version of rule 1 and rule H is an incorrect version of the conjunction of rules 11 and 12.

Section 9.3 analyses in detail how and why some of these errors occur.

	a	b	c	d	e	δ
9	1	1	2	1	3	3
12	1	1	2	2	3	1
23	1	2	2	2	2	2
30	1	3	1	2	3	3
41	2	1	1	2	2	3
43	2	1	2	1	1	3
47	2	1	2	2	2	3
50	2	2	1	1	2	3
51	2	2	1	1	3	3
52	2	2	1	2	1	3
54	2	2	1	2	3	2
59	2	2	2	2	2	3
60	2	2	2	2	3	3
63	2	3	1	1	3	3
78	3	1	1	2	3	3
82	3	1	2	2	1	3
85	3	2	1	1	1	3
89	3	2	1	2	2	3
90	3	2	1	2	3	2
103	3	3	2	1	1	3
105	3	3	2	1	3	3

Table 9.2 Incomplete training set

9.3 Analysis

9.3.1 Failure to induce a rule

For a rule to be induced, the training set must contain at least one instance which it covers uniquely. For example, PRISM failed to induce rule 13 ($a_3 \wedge b_2 \wedge c_2 \rightarrow \delta_3$). There are six possible instances of this rule, five of which are also covered by at least one other rule (rule 8, 10 or 11). It is the sixth instance (instance no. 96 in table 9.1) which must be included in the training set for rule 13 to be induced. In fact, none of these six instances is included in the partial training set, table 9.2.

Rules 2, 4, 6 and 7 cover only five instances in total, all uniquely. As none of these five instances is included in the training set, these rules cannot be induced.

α_x	$f(\delta_1\|\alpha_x)$	$p(\delta_1\|\alpha_x)$
a_1	0.25	0.056
a_2	0	0.028
a_3	0	0.028
b_1	0.143	0.083
b_2	0	0.028
b_3	0	0
c_1	0	0
c_2	0.1	0.074
d_1	0	0
d_2	0.077	0.074
e_1	0	0
e_2	0	0
e_3	0.1	0.111

Table 9.3 Relative frequency f vs. probability p for a small training set

9.3.2 Over-specialization

Theoretically, the induction algorithm is based on finding the α_x for which $p(\delta_1|\alpha_x)$ is a maximum. In practice, for an incomplete training set, the true probability of occurrence p is unknown, and is approximated by relative frequency, $f(\delta_1|\alpha_x)$. This approximation of p introduces errors in the estimation of information gain of each α_x, which become significant for small training sets, resulting in the selection of an irrelevant attribute-value pair as the best representative of δ_1.

Pure over-specialization does not occur with the training set of table 9.2[2], but the principle can be demonstrated by examining how rule A ($a_1 \wedge b_1 \wedge d_2 \rightarrow \delta_1$) is induced from this training set. Rule A is an incorrect version of rule 1 on page 125 ($b_1 \wedge c_2 \wedge d_2 \wedge e_3 \rightarrow \delta_1$). It is incorrect because it is too general with respect to attributes c and e, but it has also been over-specialized with respect to attribute a, i.e. a_1 is an unwanted term. The reason for the selection of a_1 becomes clear when the values of p and f for each α_x are compared (see table 9.3). It can be seen from table 9.3 that the value of $p(\delta_1|a_1)$ is somewhat smaller than the value of $p(\delta_1|e_3)$, (e_3 is the term which would have been selected if the training set were complete),

[2] An example of a training set in which it does occur is given in [13].

but as the distribution of the values of a is inaccurately represented in the training set, $f(\delta_1|a_1)$ is artificially high, thus leading to the selection of a_1 as 'best' attribute-value pair.

Statistical problems associated with approximating probability by relative frequency of occurrence cannot easily be avoided in PRISM, ID3 or other similar induction algorithm. As the training set becomes smaller, any rules induced from it become less reliable. PRISM informs the user of how many instances are covered by each rule, but does not specify which rules, if any, may be unreliable.

9.3.3 Over-generalization and ambiguity in induced rules

An induced rule may be too general if there are no counter-examples to it in the training set. For example, rule I above ($a_3 \wedge b_1 \rightarrow \delta_3$) is a generalization of the correct rule, rule 14 ($a_3 \wedge b_1 \wedge c_1 \rightarrow \delta_3$). As there is only one possible counter-example to rule I (instance no. 84), and this instance has not been included in the training set, there is no evidence that rule I should be specialized. Similarly, rule G ($a_2 \wedge c_2 \rightarrow \delta_3$) is a generalization of the correct rule, rule 15 ($a_2 \wedge b_2 \wedge c_2 \rightarrow \delta_3$). Again, there is only one possible counter-example (instance no. 48) which has not been included in the training set. So again, there is no evidence that rule G should be specialized.

The situation is somewhat different with rule C ($a_1 \wedge b_2 \rightarrow \delta_2$) which is a generalization of the correct rule, rule 5 ($a_1 \wedge b_2 \wedge d_2 \wedge e_2 \rightarrow \delta_2$). There are eight possible counter-examples to rule C, none of which has been included in the training set. However, rules D ($d_1 \rightarrow \delta_3$), E ($e_1 \rightarrow \delta_3$) and H ($c_1 \wedge e_2 \rightarrow \delta_3$), induced later on in the induction process, all clash with, i.e. contradict, rule C, which implies that some specialization may be necessary. Clashes, whether or not produced by over-generalization, can result in ambiguity in the final rule set. For example, an instance in which attribute a has value 1, attribute b has value 2 and attribute d has value 1 ($a_1 \& b_2 \& d_1$) would be classified as δ_2 by rule C and as δ_3 by rule D of the above rule set.

These clashes do not occur in decision trees produced by ID3 because there is always at least one common attribute, e.g. at the root node of the

tree, whose value is specified in all rules (branches). If this attribute has the same value in rules for different classes, other attributes are selected until one is found whose value differs for different classes. It is exactly this process which causes over-specialization in ID3. Attributes which are deemed necessary for discrimination along one branch are included, often unnecessarily along other branches.

Any attempts by PRISM to remove clashes between rules by specialization could have similar consequences. There is no guarantee that an attribute chosen for specialization is a relevant one. For example, the clash between rules C ($a_1 \wedge b_2 \rightarrow \delta_2$) and D ($d_1 \rightarrow \delta_3$) could be resolved by specialization in one of a number of different ways:

a : rule C can be specialized to C1 : $a_1 \wedge b_2 \wedge d_2 \rightarrow \delta_2$,
b : rule D can be specialized to D1 : $a_2 \wedge d_1 \rightarrow \delta_3$ and
D2 : $a_3 \wedge d_1 \rightarrow \delta_3$,
c : rule D can be specialized to D1 : $b_1 \wedge d_1 \rightarrow \delta_3$ and
D2 : $b_3 \wedge d_1 \rightarrow \delta_3$,
d : attribute c can be introduced into both rules,
e : attribute e can be introduced into both rules,
f : any combination of the above.

There is no significant evidence in the training set that any one of these choices is more correct than the others. Nevertheless, if ambiguity is to be removed a choice must be made.

Section 9.4 describes in detail a simple procedure which has been incorporated in PRISM to check consistency and specialize where necessary.

9.4 Specialization of over-general rules

PRISM checks consistency by comparing each induced rule with all rules previously induced for other classes. If there is at least one attribute whose value differs for the two rules being compared there is no contradiction, i.e. the rules are consistent with one another, otherwise a clash occurs and specialization becomes necessary.

Specialization is an iterative procedure:

1. **The rules to be specialized are selected.** One rule is chosen from each pair of contradictory rules, the choice being made on the basis of lesser generality. Thus the rule which potentially covers the fewer instances is selected. For example, rule C above ($a_1 \wedge b_2 \rightarrow \delta_2$) covers 12 possible instances, whereas rule D ($d_1 \rightarrow \delta_3$) covers 54 possible instances. There are six instances which are covered by both rules. These are instances containing the terms $a_1 \& b_2 \& d_1$. The purpose of specialization is to remove the ambiguity in classifying these six instances, but the cost of specializing the wrong rule can be high — the more general the rule, the higher the cost. For example, if rule C is correct, six of the instances which it covers are classified correctly (as δ_2 by rule C) and six are classified ambiguously (as δ_2 by rule C and δ_3 by rule D). Specializing rule C by adding the term d_2 (rule C becomes $a_1 \wedge b_2 \wedge d_2 \rightarrow \delta_2$) causes the instances which were previously classified ambiguously now to be classified incorrectly (by rule D). The other six instances are still classified correctly. If on the other hand, rule D is correct, 48 instances are classified correctly (as δ_3 by rule D) and six ambiguously (as δ_3 by rule D and δ_2 by rule C). Rule D can be specialized with respect to attribute a or with respect to attribute b. Unless it is specialized with respect to both attributes separately (replacing rule D by four rules — $a_2 \wedge d_1 \rightarrow \delta_3$, $a_3 \wedge d_1 \rightarrow \delta_3$, $b_1 \wedge d_1 \rightarrow \delta_3$, $b_3 \wedge d_1 \rightarrow \delta_3$) there will be a set of instances which can no longer be classified. If rule D is specialized with respect to attribute a only ($a_2 \wedge d_1 \rightarrow \delta_3, a_3 \wedge d_1 \rightarrow \delta_3$), the six ambiguously classified instances will now be classified incorrectly (by rule C), 36 of the previously correctly classified instances (those covered by the two new rules) will remain correctly classified, but the remaining 12 instances which were correctly classified ($a_1 \& b_{1 \vee 3} \& d_1$) will now be unclassified. The same applies if rule D is specialized with respect to attribute b only. Thus in both cases, specializing the wrong rule causes the six instances which were classified ambiguously to be classified incorrectly, but selecting the more general rule for specialization (by adding a single term) also reduces the number of correctly classified instances. The risk of introducing these new errors is minimized

by selecting the less general rule for specialization. For the training set of table 9.2, rules A and C are selected for specialization.

2. An attribute to be used for specialization is selected. When all rules have been induced and checked for consistency, each rule which has been marked for specialization is specialized by adding one or more terms. The attributes to be used are chosen by examining the frequency of occurrence of available attributes in the rules which clash with the rule to be specialized. Thus rule C ($a_1 \wedge b_2 \rightarrow \delta_2$) clashes with rules D ($d_1 \rightarrow \delta_3$), E ($e_1 \rightarrow \delta_3$) and H ($c_1 \wedge e_2 \rightarrow \delta_3$). Attributes c, d and e are available for specialization. Attibutes c and d occur once each in rules D, E and H, whilst attribute e occurs twice. Thus rule C is specialized with respect to attribute e. Selecting attributes on this basis allows clashes to be removed at a maximal rate, i.e. selecting attribute e has the potential of removing two clashes, whereas selecting attribute c or d would only allow at most one clash to be removed.

3. The attribute value is selected. The attribute value to be used for specialization is selected by examining which values of the relevant attribute occur in the set of instances which are covered uniquely by the rule to be specialized. Rule C covers only one instance uniquely — instance no. 23 in the training set of table 9.2. Attribute e in this instance has value 2. Thus rule C is specialized by adding the term e_2 to its premise. Rule C becomes $a_1 \wedge b_2 \wedge e_2 \rightarrow \delta_2$. Had the set of uniquely covered instances contained more than one value for e, rule C would have been duplicated for each new value. Thus had there been three uniquely covered instances, with values for attribute e of 2, 1 and 3 respectively, rule C would have been specialized as above, and two new rules — $a_1 \wedge b_2 \wedge e_1 \rightarrow \delta_2$ and $a_1 \wedge b_2 \wedge e_3 \rightarrow \delta_2$ — would have been added. The new rules would be then checked further for consistency separately.

4. Rule C is now checked for consistency with rules D, E and H. It is found that the clash with rule E has been removed, but the clashes with rules D and H remain. Thus rule C needs to be specialized further. Steps 2 and 3 above are repeated. The available attributes are c and d, occurring

once each in rules D and H. Attribute c is chosen (arbitrarily); its value in instance no. 23 is 2. Thus the term c_2 is added to rule C, removing the clash with rule H. Finally, steps 2 and 3 are repeated again, and the term d_2 is added to rule C to remove the clash with rule D. Rule C has therefore been specialized to $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_2 \rightarrow \delta_2$. Three new terms have had to be added in this case to remove clashes with three rules. Rule A ($a_1 \wedge b_1 \wedge d_1 \rightarrow \delta_1$) however, only needs the addition of one extra term, e_3, to remove clashes with two rules, rules E and H.

The final rule set for the training set of table 9.2 is:

A $a_1 \wedge b_1 \wedge d_2 \wedge e_3 \rightarrow \delta_1$

B $b_2 \wedge c_1 \wedge d_2 \wedge e_3 \rightarrow \delta_2$

C $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_2 \rightarrow \delta_2$

D $d_1 \rightarrow \delta_3$

E $e_1 \rightarrow \delta_3$

F $b_3 \rightarrow \delta_3$

G $a_2 \wedge c_2 \rightarrow \delta_3$

H $c_1 \wedge e_2 \rightarrow \delta_3$

I $a_3 \wedge b_1 \rightarrow \delta_3$

There is no ambiguity in this rule set, and its performance has been significantly improved. If this rule set is tested on the complete set of instances (table 9.1) it is found that 98 instances are classified correctly, five are classified incorrectly and five are unclassified, whereas the original rule set (listed on pages 125—126) classified 88 instances correctly and six incorrectly, three instances being unclassified and 11 classified ambiguously. So specialization has resulted in an increase in the number of correctly classified instances and a decrease in the number of incorrectly classified instances, as well as the removal of any ambiguity.

A summary of the complete algorithm is given in section 9.5 and section 9.6 describes a series of experiments which was performed to test how well PRISM's induced rules can predict the class of unseen instances in general and to compare this with the performance of ID3.

9.5 Summary of the induction procedure

PRISM uses the basic algorithm described in section 8.1 for inducing individual rules. This algorithm, reproduced here for completeness, proceeds as follows:

Step A: calculate the probability of occurrence, $p(\delta_n|\alpha_x)$, of the classification δ_n for each attribute-value pair α_x,

Step B: select the α_x for which $p(\delta_n|\alpha_x)$ is a maximum and create a subset of the training set comprising all the instances which contain the selected α_x,

Step C: repeat Steps A and B for this subset until it contains only instances of class δ_n. The induced rule is a conjunction of all the attribute-value pairs used in creating the homogenous subset.

Step D: remove all instances covered by this rule from the training set,

Step E: repeat Steps A – D until all instances of class δ_n have been removed.

The complete induction procedure is summarized below.

Step 1: Use the basic algorithm to induce a set of rules for the first class δ_1.

Step 2: Select the most general rule from this set and add it to the final rule set.

Step 3: Check this rule for consistency with other rules in the final rule set (if there are any). If a clash occurs, select one rule for specialization.

Step 4: Remove all instances covered by this rule from the training set.

Step 5: Repeat steps 1 — 4 until there are no instances of class δ_1 left in the training set.

Step 6: Restore the training set to its initial state.

Step 7: Repeat steps 1 — 6 for each class $\delta_2 \cdots \delta_n$ in turn.

Step 8: Specialize each rule identified as being too general in step 3 by iteratively selecting and adding new terms until no clashes occur.

A full listing of the program is given in Appendix A.

9.6 Predictive power

If one defines the predictive power of an induced rule set as its ability to classify instances correctly, then as can be expected, predictive power depends on the relative size of the training set from which the rules were induced. In order to determine the relationship between the size of a training set and the predictive power of a rule set induced from it, a fixed number of instances were selected at random from the complete data set shown in table 9.1, a set of rules was induced from these instances using PRISM, and then the induced rules were tested on the full set of instances to calculate a) the percentage of instances which were classified correctly, b) the percentage of instances which were classified incorrectly, and c) the percentage of instances which could not be classified. This was repeated one hundred times each for training sets containing 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80% and 90% of the complete data set, and the results averaged for each size of training set. These results are shown in table 9.4. Table 9.4 also shows

the average number of rules induced and the average total number of terms comprising these rules.

The experiment was then repeated using ID3 (adapted for more than two classes); the results are shown in table 9.5.

The results relating to correct and incorrect classification from both tables 9.4 and 9.5 are shown in figure 9.2, in which graph A shows the percentage of instances classified correctly by rules induced using PRISM and graph B shows the percentage of instances classified correctly by the decision tree induced using ID3. Graphs C and D show the percentage of instances classified incorrectly by PRISM's rule set and ID3's decision tree, respectively.

These results show that although the numbers of correctly classified instances is similar for both PRISM and ID3, the numbers of incorrectly classified instances differ significantly. ID3 regularly classifies more instances incorrectly than does PRISM. Tables 9.4 and 9.5 also show the average number of rules (or branches in a decision tree) induced from each size of training set and the total number of terms comprising these rules[3]. These are displayed in figure 9.3, which shows that in general ID3's rules are considerably more specific than those induced by PRISM. Thus PRISM has reduced over-specialization without sacrificing predictive power. Furthermore, performance has been improved because the probability of incorrect classification is lower. An expert system using a decision tree induced by ID3 requires on average more tests to be performed than does one using PRISM's rule set, but the probability of the decision being correct is similar and the probability of the decision being incorrect is greater.

The reason for this is simply that ID3 uses a decision tree representation. For example, ID3 was applied to the incomplete training set of table 9.2, The branches of the induced decision tree, shown in figure 9.4, are listed below as separate rules:

[3]To determine the number of terms in a decision tree, the tree is first trivially converted into a set of individual rules, and the total number of terms in those rules counted.

% of complete data set				average number of rules	average number of terms
size of training set	classified correctly	classified incorrectly	not classified		
5	75.07	15.06	9.86	1.90	2.00
10	76.00	12.36	11.64	3.27	5.27
20	84.76	7.28	7.95	5.97	13.21
30	88.28	5.67	6.05	7.95	19.78
40	90.94	3.60	5.46	9.56	26.22
50	92.65	3.36	3.99	10.66	30.63
60	95.03	2.16	2.81	12.16	36.24
70	96.22	1.64	2.14	13.05	39.55
80	97.53	0.94	1.53	13.76	42.40
90	98.81	0.56	0.62	14.52	45.34

Table 9.4 Results of experiment to test predictive power of rules induced by PRISM from incomplete training sets

% of complete data set				average number of rules	average number of terms
size of training set	classified correctly	classified incorrectly	not classified		
5	76.31	20.69	3.00	1.95	1.80
10	76.95	18.90	4.15	3.67	5.89
20	81.54	15.38	3.08	6.72	15.78
30	85.74	11.57	2.69	9.10	25.25
40	89.48	8.79	1.73	11.24	34.53
50	91.31	7.13	1.56	12.51	40.72
60	94.09	4.68	1.23	13.92	47.55
70	96.10	2.83	1.06	14.62	52.59
80	98.06	1.20	0.74	15.23	56.45
90	99.16	0.49	0.35	15.65	59.88

Table 9.5 Results of experiment to test predictive power of decision trees induced by ID3 from incomplete training sets

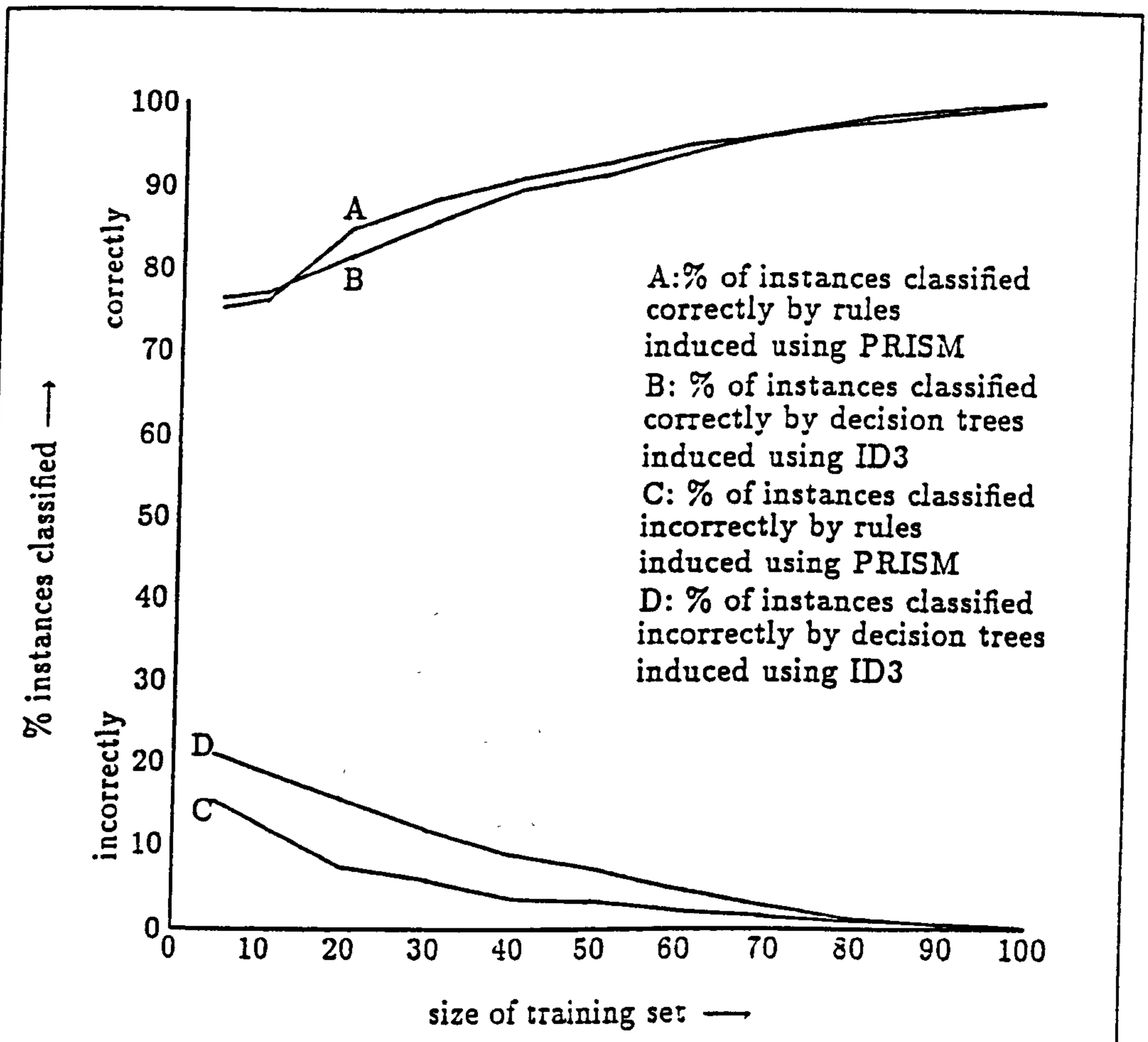


Figure 9.2 Correct and incorrect classification of instances

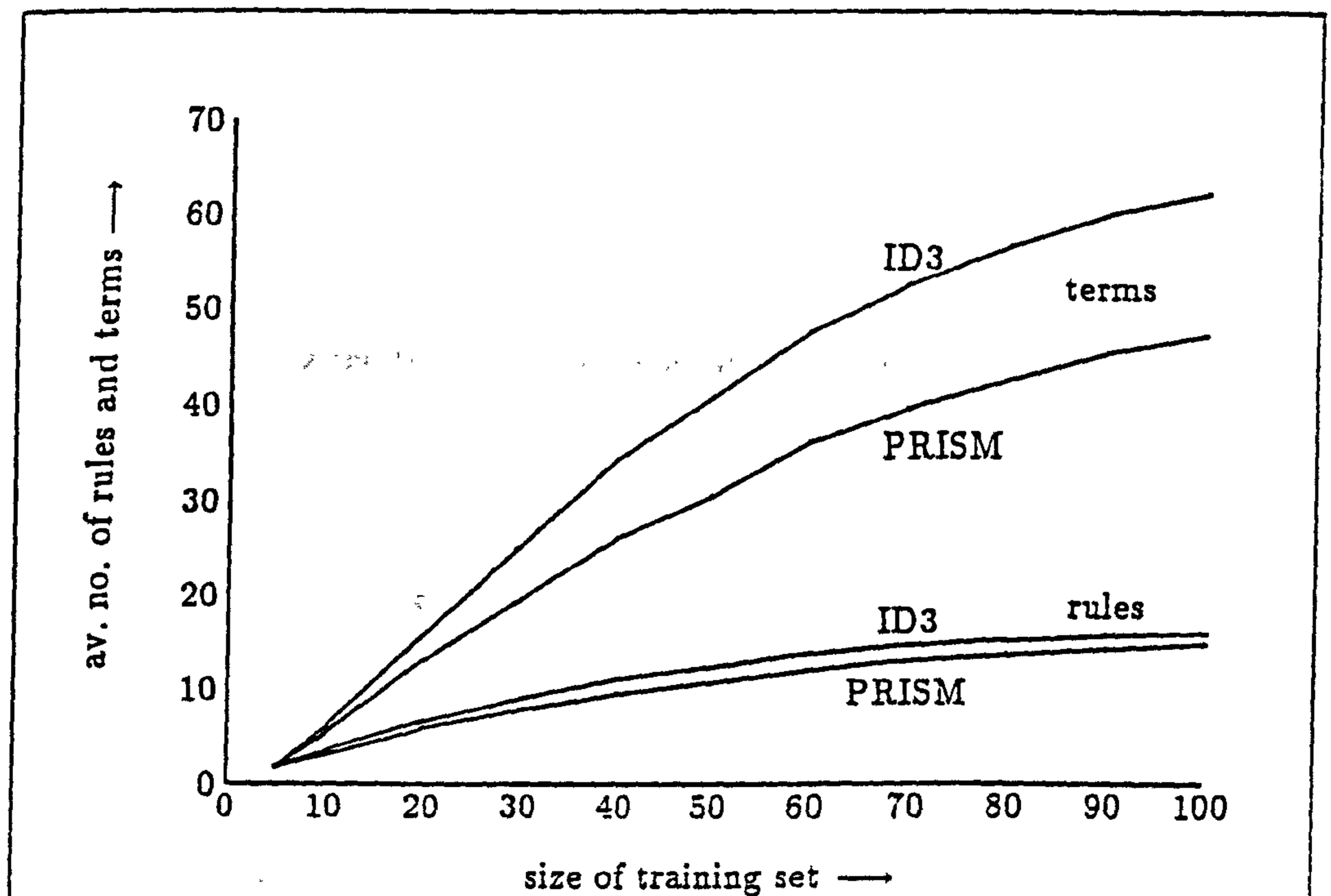


Figure 9.3 Number of rules and terms induced

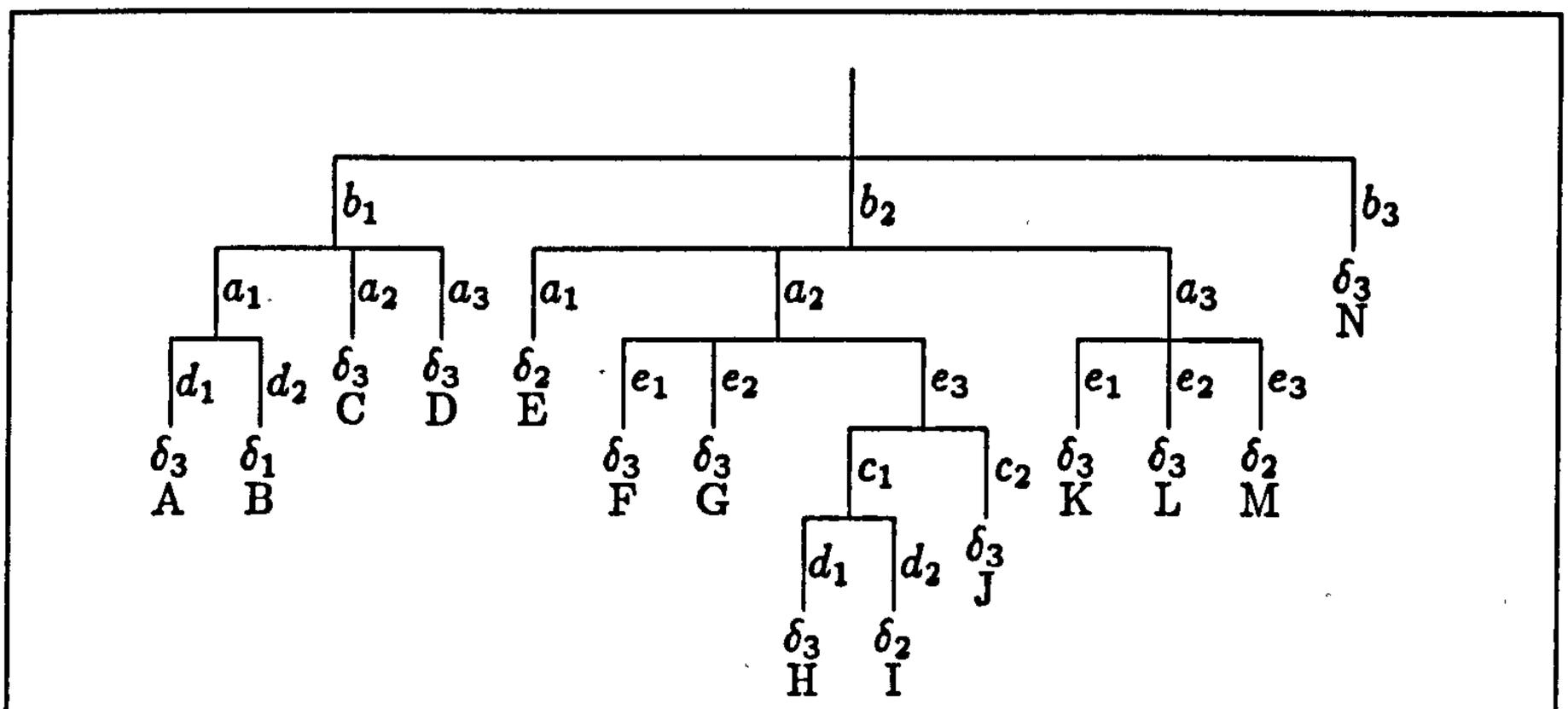


Figure 9.4 Decision tree for incomplete training set

A $a_1 \wedge b_1 \wedge d_1 \rightarrow \delta_3$

B $a_1 \wedge b_1 \wedge d_2 \rightarrow \delta_1$

C $a_2 \wedge b_1 \rightarrow \delta_3$

D $a_3 \wedge b_1 \rightarrow \delta_3$

E $a_1 \wedge b_2 \rightarrow \delta_2$

F $a_2 \wedge b_2 \wedge e_1 \rightarrow \delta_3$

G $a_2 \wedge b_2 \wedge e_2 \rightarrow \delta_3$

H $a_2 \wedge b_2 \wedge c_1 \wedge d_1 \wedge e_3 \rightarrow \delta_3$

I $a_2 \wedge b_2 \wedge c_1 \wedge d_2 \wedge e_3 \rightarrow \delta_2$

J $a_2 \wedge b_2 \wedge c_2 \wedge e_3 \rightarrow \delta_3$

K $a_3 \wedge b_2 \wedge e_1 \rightarrow \delta_3$

L $a_3 \wedge b_2 \wedge e_2 \rightarrow \delta_3$

M $a_3 \wedge b_2 \wedge e_3 \rightarrow \delta_2$

N $b_3 \rightarrow \delta_3$

Consider the set of instances which have value 1 for attribute *e*. These instances are covered by the rule $e_1 \rightarrow \delta_3$ and should therefore all be classified as δ_3. PRISM discovers this underlying rule (rule E on page 133) and thus classifies the instances correctly. ID3 does not discover the rule $e_1 \rightarrow \delta_3$. The instances are classified correctly only if they are covered by some other rule which classifies them as δ_3, or if attribute *e* has been chosen (correctly) for specialization after other attributes have been tried (incorrectly). Branches F and K of the decision tree illustrate this. Both of these branches reference attributes *a* and *b*, i.e. they are over-specialized. This will always be the case with the contact lens data set because the underlying rule set contains three (equally general) rules, $d_1 \rightarrow \delta_3$, $e_1 \rightarrow \delta_3$ and $b_3 \rightarrow \delta_3$. ID3 can only discover one of these rules without over-specialization. In the current example the rule $b_3 \rightarrow \delta_3$ has been induced correctly (branch N).

Predictive power, as defined at the beginning of this section, depends on the size of the training set. It also depends on the generality and number of actual rules governing the data. The experiment described above was repeated using Quinlan's chess data (see section 8.3). The results are shown in tables 9.6 and 9.7 and figures 9.5 and 9.6, which show the percentages of instances classified correctly and incorrectly, and the average numbers of rules and terms induced.

By comparing figures 9.2 and 9.5 it can be seen that for the same relative size of training set, a rule set induced for the chess data is more accurate than one induced for the contact lens data. One of the reasons for this is that the rules underlying the chess data tend to be on average more general than those underlying the contact lens data (15 rules cover 647 instances in the former case and 108 instances in the latter case). The more general a rule the easier it is to induce it, i.e. fewer instances are needed, and once induced, it classifies more instances correctly. Thus, to classify 90% of the instances correctly requires an initial training set of 40% — 45% for the

% of complete data set				average number of rules	average number of terms
size of training set	classified correctly	classified incorrectly	not classified		
5	89.34	3.87	6.79	4.80	9.68
10	92.79	2.67	4.54	6.97	16.58
20	95.65	1.72	2.62	10.14	30.27
30	97.04	0.93	2.03	12.63	40.53
40	98.07	0.48	1.44	13.83	43.76
50	99.01	0.20	0.77	14.89	47.68
60	99.46	0.12	0.42	14.95	47.10
70	99.71	0.04	0.24	15.11	47.22
80	99.85	0.02	0.13	15.13	47.62
90	99.94	0.00	0.06	14.90	46.85

Table 9.6 Results of experiment to test predictive power of rules induced by PRISM using chess data

% of complete data set				average number of rules	average number of terms
size of training set	classified correctly	classified incorrectly	not classified		
5	88.87	9.24	1.89	5.55	13.12
10	92.67	6.24	1.09	7.82	23.71
20	96.57	2.81	0.63	10.73	43.36
30	97.83	1.58	0.59	13.42	61.75
40	98.55	0.99	0.46	15.22	72.91
50	99.11	0.53	0.36	16.87	83.90
60	99.37	0.42	0.21	18.17	92.37
70	99.64	0.18	0.17	19.02	98.10
80	99.90	0.06	0.04	19.64	102.43
90	99.92	0.05	0.03	20.09	105.80

Table 9.7 Results of experiment to test predictive power of decision trees induced by ID3 using chess data

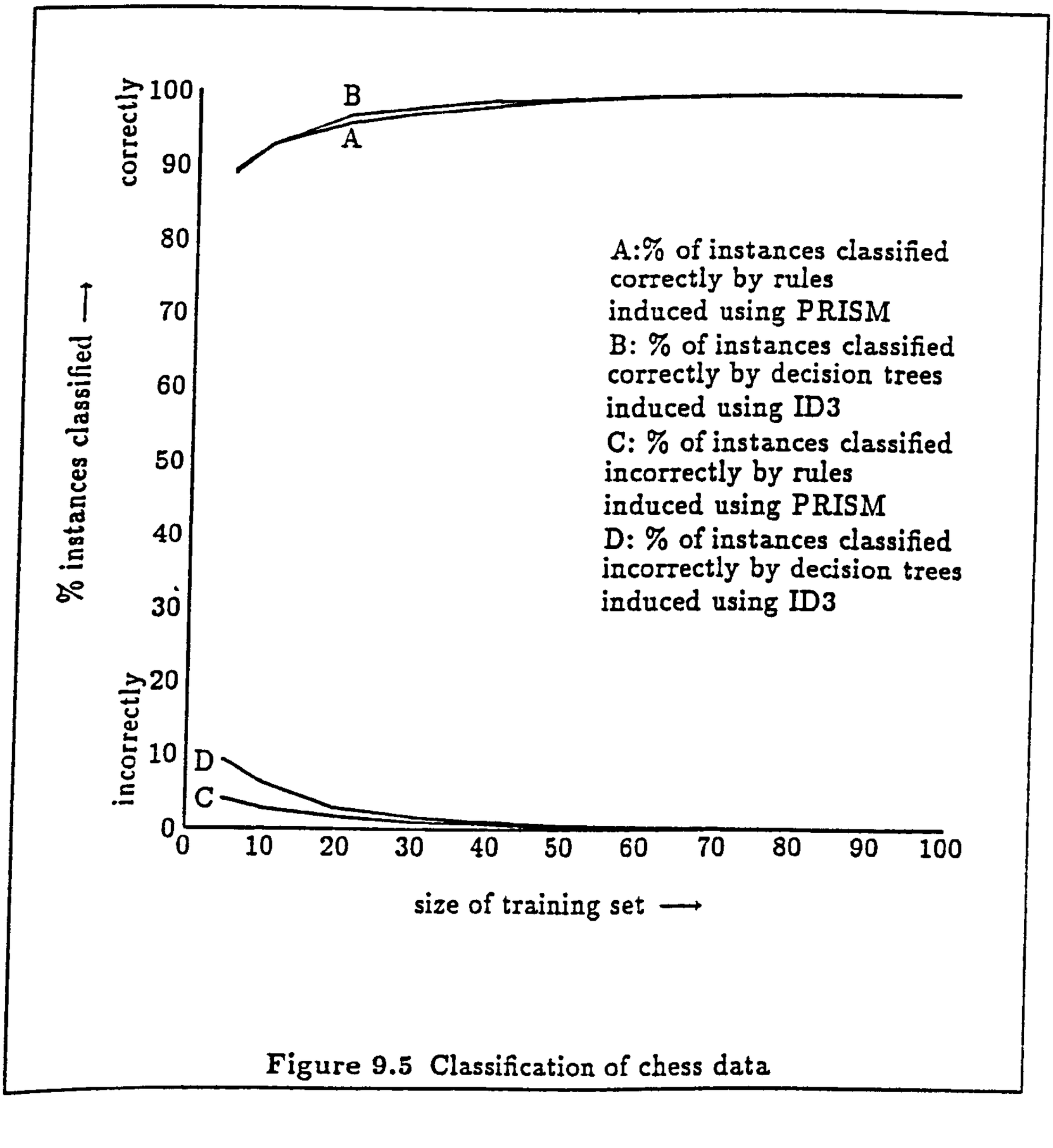


Figure 9.5 Classification of chess data

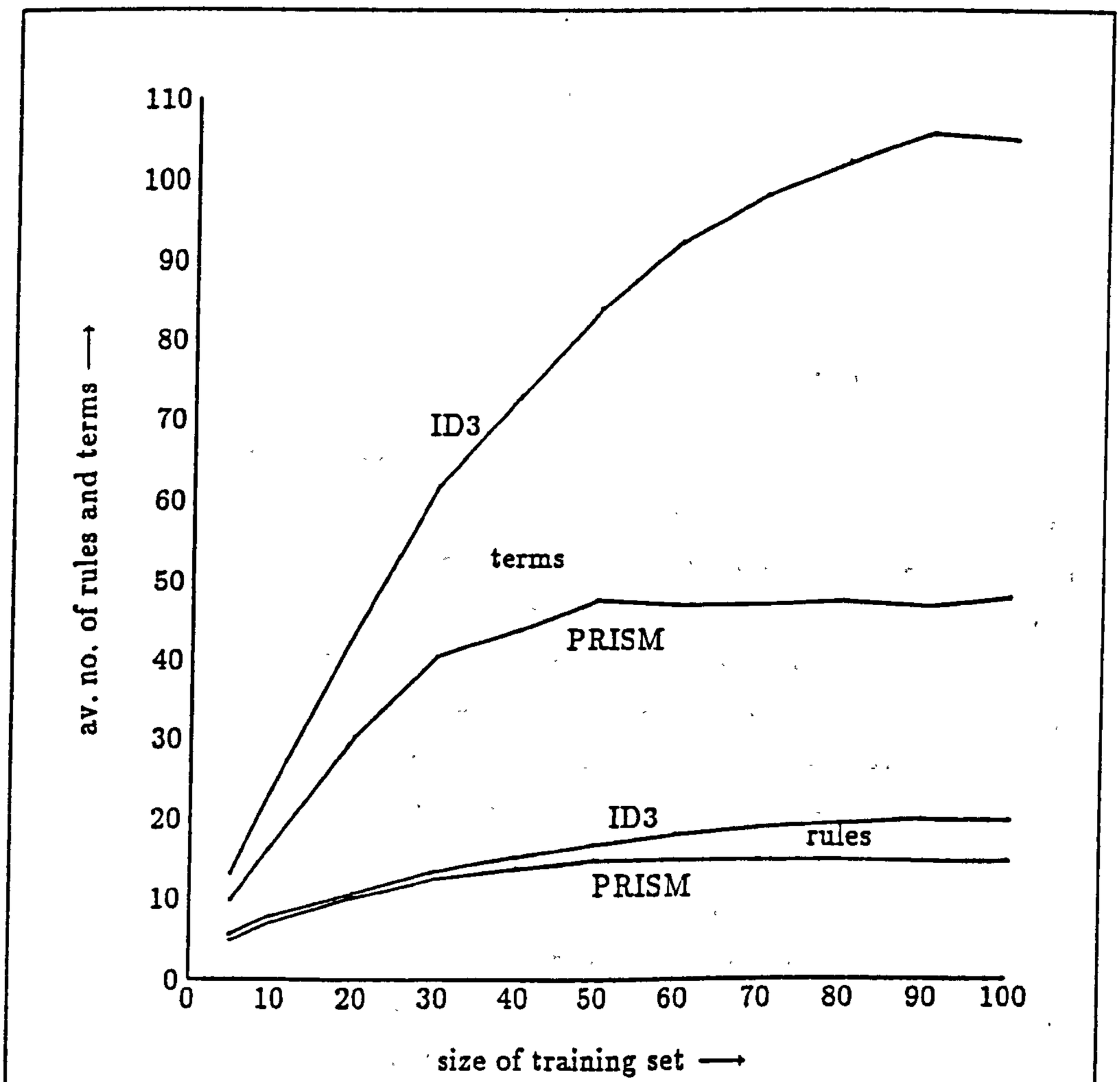


Figure 9.6 Number of rules and terms induced (chess data)

size of training set	% of instances classified correctly	
	contact lens data	chess data
5	73.76	88.78
10	73.33	91.99
20	80.95	94.56
30	83.24	95.77
40	84.90	96.78
50	85.28	98.02
60	87.58	98.65
70	87.40	99.03
80	87.65	99.25
90	88.10	99.40

Table 9.8 Predictive power of PRISM's rules

contact lens data, but less than 10% of the chess data.

The graphs of figures 9.2 and 9.5 show results which include instances which are present in the training set. As all of these instances are always classified uniquely and correctly, it must be expected that predictive power increases with the size of the training set. However, if one defines predictive power as the ability of an induced rule set to classify *new* instances correctly, this is not so obvious. The results of tables 9.4 and 9.6 were used to calculate the percentages of new instances which were classified correctly in each case. These are shown in table 9.8 and figure 9.7, which shows the percentages of new instances classified correctly by PRISM's rule set, Graph A being the results for the contact lens data and graph B the results for the chess data. These results show that predictive power does in fact increase with the size of the training set in each case.

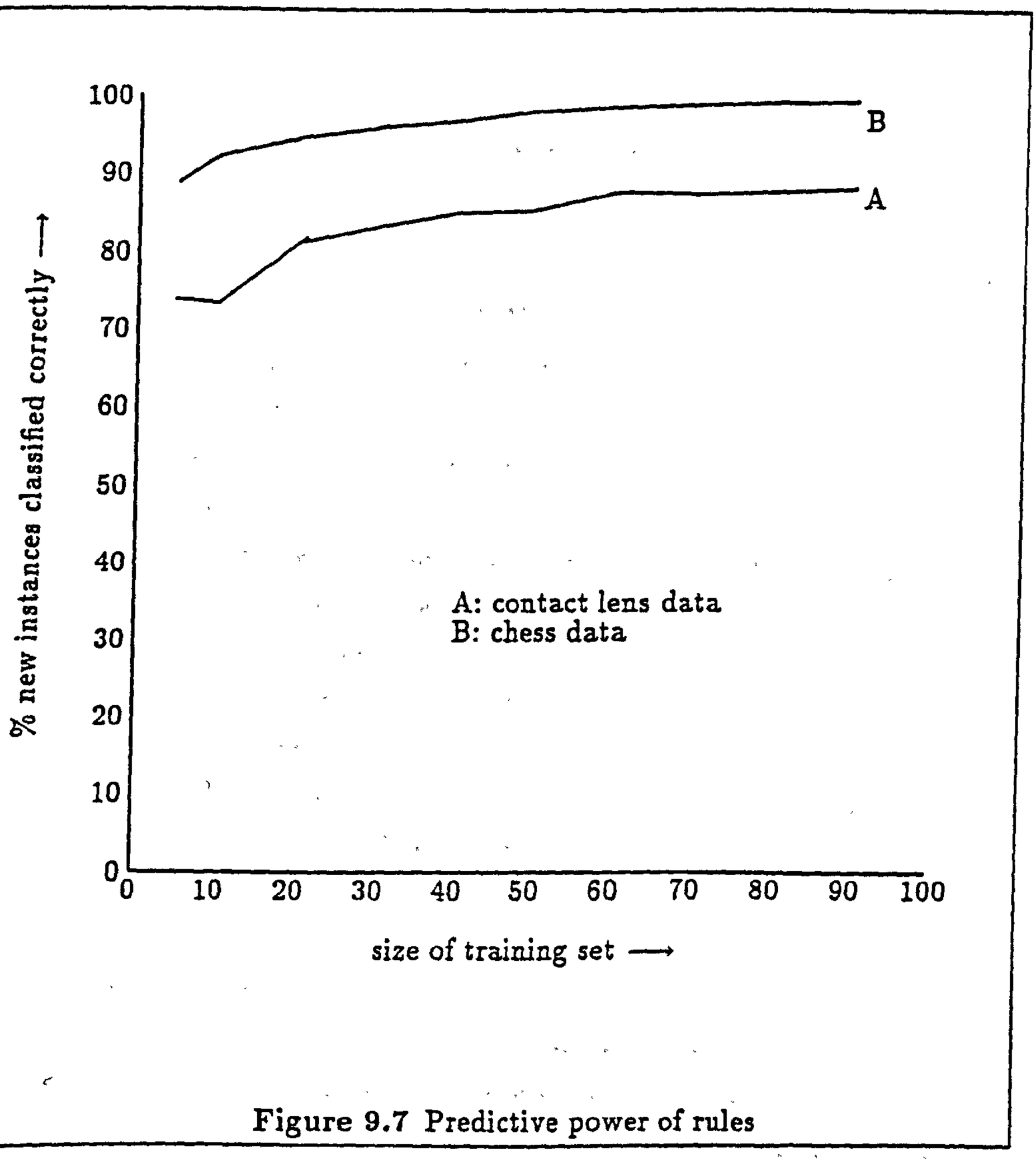


Figure 9.7 Predictive power of rules

Chapter 10

Attributes with linear values

The underlying theory on which PRISM, like ID3, is based was developed under the assumption that all attributes have discrete values. This chapter describes a series of experiments performed to assess ways in which PRISM might be modified to enable it to deal with attributes with linear values, as defined in Chapter 6.

The experiments were performed using the contact lens data detailed in section 9.1, modified to allow the values, V_i, of attribute e, tear break-up time, measured to the nearest second, to be given as actual measurements in seconds within the range $0 < V_i \leq 20$. The values of attributes a, b, c and d are discrete, as described in section 9.1. The new complete training set consists of $3 \times 3 \times 2 \times 2 \times 20 = 720$ instances. When PRISM was applied to it, 80 rules were induced. These were the same as those induced from table 9.1, listed on page 125, except where a rule references attribute e, a separate rule was induced for each value of e within the relevant range. Thus 10 rules were induced in place of rule 1 on page 125, one for each value of attribute e in group 3 ($10 < V_i \leq 20$); 10 rules were induced in place of each of rules 2 and 3, 5 rules in place of each of rules 4 and 5, etc.

Selecting single values of attribute e instead of groups of values is a form of specialization. With an incomplete training set in which the frequency of occurrence of each value of e is likely to be greatly reduced (compared with a training set in which the values are grouped into ranges and treated as discrete) there is less likelihood of counter-examples. Consequently, attribute e

is more discriminating than and is likely to be selected in preference to other (discrete) attributes.

Table 10.1 is an incomplete training set containing 144 (20%) instances selected at random from the complete set. The rules listed in table 10.2 are those induced by PRISM when the values of attribute e are grouped into the appropriate ranges and treated as discrete. In comparison, the following rules are those induced by PRISM from the training set of table 10.1 without grouping the values of e:

1 $a_1 \wedge b_1 \wedge d_2 \wedge e_{20} \rightarrow \delta_1$
2 $a_1 \wedge b_1 \wedge d_2 \wedge e_{15} \rightarrow \delta_1$
3 $a_1 \wedge b_2 \wedge d_2 \wedge e_{18} \rightarrow \delta_1$
4 $a_1 \wedge b_2 \wedge d_2 \wedge e_{11} \rightarrow \delta_1$
5 $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{12} \rightarrow \delta_1$
6 $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{14} \rightarrow \delta_1$
7 $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{16} \rightarrow \delta_1$
8 $a_2 \wedge b_1 \wedge d_2 \wedge e_{18} \rightarrow \delta_1$
9 $a_2 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{17} \rightarrow \delta_1$
10 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{16} \rightarrow \delta_2$
11 $b_2 \wedge d_2 \wedge e_{17} \rightarrow \delta_2$
12 $a_1 \wedge b_2 \wedge d_2 \wedge e_6 \rightarrow \delta_2$
13 $b_2 \wedge d_2 \wedge e_{13} \rightarrow \delta_2$
14 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{12} \rightarrow \delta_2$
15 $b_2 \wedge d_2 \wedge e_{14} \rightarrow \delta_2$
16 $a_1 \wedge b_2 \wedge d_2 \wedge e_7 \rightarrow \delta_2$
17 $a_1 \wedge b_2 \wedge d_2 \wedge e_{12} \rightarrow \delta_2$
18 $a_1 \wedge b_1 \wedge d_2 \wedge e_{11} \rightarrow \delta_2$
19 $a_1 \wedge b_2 \wedge d_2 \wedge e_9 \rightarrow \delta_2$
20 $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_8 \rightarrow \delta_2$
21 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{14} \rightarrow \delta_2$
22 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{10} \rightarrow \delta_2$
23 $a_1 \wedge b_1 \wedge d_2 \wedge e_{17} \rightarrow \delta_2$
24 $d_1 \rightarrow \delta_3$
25 $b_3 \rightarrow \delta_3$
26 $a_3 \wedge b_1 \rightarrow \delta_3$
27 $e_4 \rightarrow \delta_3$
28 $e_2 \rightarrow \delta_3$
29 $e_3 \rightarrow \delta_3$
30 $e_{19} \rightarrow \delta_3$
31 $c_1 \wedge e_8 \rightarrow \delta_3$
32 $a_2 \wedge e_7 \rightarrow \delta_3$
33 $a_2 \wedge e_9 \rightarrow \delta_3$
34 $a_2 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{16} \rightarrow \delta_2$

As expected, most of these rules are over-specialized with respect to attribute e. This causes over-generalization with respect to other attributes leading to clashes which can only be resolved by further specialization.

	a	b	c	d	e	δ		a	b	c	d	e	δ
1	1	1	1	1	8	3	37	1	2	2	2	11	1
2	1	1	1	1	16	3	38	1	2	2	2	18	1
3	1	1	1	1	17	3	39	1	3	1	1	4	3
4	1	1	1	1	18	3	40	1	3	1	1	10	3
5	1	1	1	2	10	2	41	1	3	1	1	13	3
6	1	1	1	2	11	2	42	1	3	1	1	20	3
7	1	1	1	2	12	2	43	1	3	1	2	4	3
8	1	1	1	2	14	2	44	1	3	1	2	9	3
9	1	1	1	2	16	2	45	1	3	1	2	10	3
10	1	1	1	2	17	2	46	1	3	1	2	12	3
11	1	1	2	1	7	3	47	1	3	1	2	14	3
12	1	1	2	1	10	3	48	1	3	1	2	15	3
13	1	1	2	1	14	3	49	1	3	2	1	1	3
14	1	1	2	1	15	3	50	1	3	2	1	2	3
15	1	1	2	1	19	3	51	1	3	2	1	9	3
16	1	1	2	2	12	1	52	1	3	2	1	11	3
17	1	1	2	2	14	1	53	1	3	2	1	12	3
18	1	1	2	2	15	1	54	1	3	2	2	9	3
19	1	1	2	2	16	1	55	1	3	2	2	17	3
20	1	1	2	2	20	1	56	2	1	1	1	3	3
21	1	2	1	1	1	3	57	2	1	1	1	11	3
22	1	2	1	1	14	3	58	2	1	1	1	13	3
23	1	2	1	1	17	3	59	2	1	1	1	14	3
24	1	2	1	1	20	3	60	2	1	1	1	18	3
25	1	2	1	2	4	3	61	2	1	1	1	19	3
26	1	2	1	2	6	2	62	2	1	1	2	2	3
27	1	2	1	2	12	2	63	2	1	1	2	4	3
28	1	2	1	2	14	2	64	2	1	1	2	8	3
29	1	2	2	1	9	3	65	2	1	1	2	9	3
30	1	2	2	1	19	3	66	2	1	1	2	16	2
31	1	2	2	1	20	3	67	2	1	2	1	10	3
32	1	2	2	2	3	3	68	2	1	2	2	17	1
33	1	2	2	2	4	3	69	2	1	2	2	18	1
34	1	2	2	2	7	2	70	2	2	1	1	3	3
35	1	2	2	2	8	2	71	2	2	1	1	10	3
36	1	2	2	2	9	2	72	2	2	1	1	11	3

Table 10.1 Incomplete training set with linear values for e (part 1)

	a	b	c	d	e	δ		a	b	c	d	e	δ
73	2	2	1	1	16	3	109	3	1	2	2	7	3
74	2	2	1	2	3	3	110	3	1	2	2	8	3
75	2	2	1	2	7	3	111	3	1	2	2	10	3
76	2	2	1	2	8	3	112	3	2	1	1	1	3
77	2	2	1	2	13	2	113	3	2	1	1	5	3
78	2	2	1	2	17	2	114	3	2	1	1	6	3
79	2	2	2	1	1	3	115	3	2	1	1	18	3
80	2	2	2	1	2	3	116	3	2	1	1	20	3
81	2	2	2	1	18	3	117	3	2	1	2	2	3
82	2	2	2	2	7	3	118	3	2	1	2	4	3
83	2	3	1	2	3	3	119	3	2	1	2	8	3
84	2	3	1	2	6	3	120	3	2	1	2	17	2
85	2	3	1	2	8	3	121	3	2	2	1	2	3
86	2	3	1	2	9	3	122	3	2	2	1	5	3
87	2	3	1	2	13	3	123	3	2	2	1	12	3
88	2	3	1	2	17	3	124	3	2	2	1	16	3
89	2	3	2	1	6	3	125	3	2	2	1	18	3
90	2	3	2	1	8	3	126	3	2	2	1	19	3
91	2	3	2	1	20	3	127	3	2	2	2	2	3
92	2	3	2	2	5	3	128	3	2	2	2	19	3
93	2	3	2	2	10	3	129	3	3	1	1	1	3
94	2	3	2	2	17	3	130	3	3	1	1	3	3
95	2	3	2	2	19	3	131	3	3	1	1	4	3
96	3	1	1	1	2	3	132	3	3	1	1	8	3
97	3	1	1	1	4	3	133	3	3	1	1	11	3
98	3	1	1	1	6	3	134	3	3	1	1	16	3
99	3	1	1	1	13	3	135	3	3	1	2	3	3
100	3	1	1	1	17	3	136	3	3	1	2	12	3
101	3	1	1	1	18	3	137	3	3	1	2	13	3
102	3	1	1	1	19	3	138	3	3	2	1	1	3
103	3	1	1	2	2	3	139	3	3	2	1	3	3
104	3	1	1	2	4	3	140	3	3	2	2	2	3
105	3	1	1	2	9	3	141	3	3	2	2	5	3
106	3	1	1	2	11	3	142	3	3	2	2	13	3
107	3	1	1	2	17	3	143	3	3	2	2	15	3
108	3	1	2	1	19	3	144	3	3	2	2	18	3

Table 10.1 Incomplete training set with linear values for e (part 2)

1	$a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_1$
2	$a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_1$
3	$a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{6-10} \rightarrow \delta_2$
4	$b_2 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$
5	$a_1 \wedge b_2 \wedge d_2 \wedge e_{6-10} \rightarrow \delta_2$
6	$a_2 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$
7	$d_1 \rightarrow \delta_3$
8	$b_3 \rightarrow \delta_3$
9	$e_{1-5} \rightarrow \delta_3$
10	$a_3 \wedge c_2 \rightarrow \delta_3$
11	$a_3 \wedge b_1 \rightarrow \delta_3$
12	$a_2 \wedge e_{6-10} \rightarrow \delta_3$
13	$a_3 \wedge e_{6-10} \rightarrow \delta_3$
14	$a_2 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_1$
15	$a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$

Table 10.2 Rules induced by PRISM when values of e are discrete groups

To try to remedy this, three different modifications of PRISM were tried. In the first, the values of attribute e were divided into a number of equal ranges. The results are described in section 10.1 below. The second modification enabled PRISM to handle linear values in a similar way to ACLS and ASSISTANT (see sections 5.3 and 5.4, and [42] and [30]). This is described in section 10.2, and section 10.3 describes the third modification which enabled PRISM to select a 'best' range of values in terms of information gain.

10.1 Values of e divided into equal ranges

The values of attribute e were divided first into ten equal ranges: 1–2 secs. incl., 3–4 secs. incl., $\cdots$, 19–20 secs. incl. PRISM induced 29 rules:

1	$a_2 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{17-18} \rightarrow \delta_1$	7	$a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{9-10} \rightarrow \delta_2$
2	$a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{11-12} \rightarrow \delta_1$	8	$a_1 \wedge b_2 \wedge d_2 \wedge e_{13-14} \rightarrow \delta_2$
3	$a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{15-16} \rightarrow \delta_1$	9	$a_1 \wedge b_2 \wedge d_2 \wedge e_{7-8} \rightarrow \delta_2$
4	$a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{19-20} \rightarrow \delta_1$	10	$b_2 \wedge c_1 \wedge d_2 \wedge e_{17-18} \rightarrow \delta_2$
5	$a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{13-14} \rightarrow \delta_1$	11	$a_1 \wedge b_2 \wedge d_2 \wedge e_{5-6} \rightarrow \delta_2$
6	$a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_{17-18} \rightarrow \delta_1$	12	$a_2 \wedge b_1 \wedge d_2 \wedge e_{15-16} \rightarrow \delta_2$

13 $a_1 \wedge b_2 \wedge d_2 \wedge e_{9-10} \rightarrow \delta_2$

14 $a_1 \wedge b_2 \wedge c_1 \wedge d_2 \wedge e_{11-12} \rightarrow \delta_2$

15 $d_1 \rightarrow \delta_3$

16 $b_3 \rightarrow \delta_3$

17 $a_3 \wedge c_2 \rightarrow \delta_3$

18 $e_{3-4} \rightarrow \delta_3$

19 $a_3 \wedge b_1 \rightarrow \delta_3$

20 $e_{1-2} \rightarrow \delta_3$

21 $a_2 \wedge e_{7-8} \rightarrow \delta_3$

22 $a_2 \wedge e_{9-10} \rightarrow \delta_3$

23 $a_3 \wedge e_{7-8} \rightarrow \delta_3$

24 $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_{11-12} \rightarrow \delta_1$

25 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{11-12} \rightarrow \delta_2$

26 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{13-14} \rightarrow \delta_2$

27 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{15-16} \rightarrow \delta_2$

28 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{17-18} \rightarrow \delta_2$

29 $a_2 \wedge b_2 \wedge d_2 \wedge e_{13-14} \rightarrow \delta_1$

As can be seen, many of these rules are still highly specialized; in particular with reference to attribute *e*. Because each range is small (it contains only two points) and because the training set is incomplete (it contains 40% of the possible number of instances) *e* is a highly discriminating attribute, making it more likely to be selected than other (correct) attributes.

The values of attribute *e* were then divided into five equal ranges: 1–4 secs. incl., 5–8 secs. incl., $\cdots$, 17–20 secs. incl. The training set contained about 57% of the total number of possible instances with some duplicates. There were also two clashing instances. These are instances no. 36 and no. 37 from table 10.1:

$$36.\ a_1 \& b_2 \& c_2 \& d_2 \& e_9 \rightarrow \delta_2$$
$$37.\ a_1 \& b_2 \& c_2 \& d_2 \& e_{11} \rightarrow \delta_1$$

Both of these instances have values for attribute *e* which fall into the same range (9–12 secs. incl.), which violates the condition that for PRISM to work the attributes must be adequate. The consequences of this violation are that PRISM is unable to discriminate between the two instances and thus reproduces them as two maximally specific but contradictory rules. If instances no. 36 and no. 37 and other duplicate instances are removed from the training set, the following 22 rules are induced:

1 $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{9-12} \rightarrow \delta_1$

2 $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{17-20} \rightarrow \delta_1$

3 $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_{17-20} \rightarrow \delta_1$

4 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{9-12} \rightarrow \delta_2$

5 $a_1 \wedge b_2 \wedge d_2 \wedge e_{13-16} \rightarrow \delta_2$

6 $a_1 \wedge b_2 \wedge d_2 \wedge e_{5-8} \rightarrow \delta_2$

7 $b_2 \wedge c_1 \wedge d_2 \wedge e_{17-18} \rightarrow \delta_2$

8 $a_1 \wedge b_2 \wedge d_2 \wedge e_{9-12} \rightarrow \delta_2$

9 $a_2 \wedge b_1 \wedge d_2 \wedge e_{13-16} \rightarrow \delta_2$

10 $d_1 \rightarrow \delta_3$

11 $b_3 \rightarrow \delta_3$

12 $e_{1-4} \rightarrow \delta_3$

13 $a_3 \wedge c_2 \rightarrow \delta_3$

14 $a_3 \wedge b_1 \rightarrow \delta_3$

15 $a_3 \wedge e_{5-8} \rightarrow \delta_3$

16 $a_2 \wedge e_{5-8} \rightarrow \delta_3$

17 $a_2 \wedge e_{9-12} \rightarrow \delta_3$

18 $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{13-16} \rightarrow \delta_1$

19 $a_2 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{17-20} \rightarrow \delta_1$

20 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{13-16} \rightarrow \delta_2$

21 $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{17-20} \rightarrow \delta_2$

22 $a_2 \wedge b_2 \wedge d_2 \wedge e_{13-16} \rightarrow \delta_1$

Although many of the rules are still over-specialized with respect to attribute *e*, there are fewer rules and less over-specialization in general. As the ranges into which the values of *e* are divided become larger and fewer in number, so the chance of selecting the wrong attributes decreases.

The values of attribute *e* were then divided into two equal ranges: 1–10 secs. incl. and 11–20 secs. incl. which resulted in an increase in the number of clashing instances. The following 12 rules were induced from a training set from which all clashes and duplicates had been removed:

1. $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_1$

2. $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \rightarrow \delta_1$

3. $b_2 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$

4. $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \rightarrow \delta_2$

5. $a_2 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$

6. $d_1 \rightarrow \delta_3$

7. $b_3 \rightarrow \delta_3$

8. $a_3 \wedge e_{1-10} \rightarrow \delta_3$

9. $a_2 \wedge e_{1-10} \rightarrow \delta_3$

10. $a_3 \wedge c_2 \rightarrow \delta_3$

11. $a_3 \wedge b_1 \rightarrow \delta_3$

12. $a_2 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_1$

Because e is no longer such a discriminating attribute, the quality of most of the rules has improved. For the same reason, some other rules are too general with respect to attribute e. Furthermore, some of the improvement in quality can be attributed to the fact that the range 11–20 secs. incl. happens to coincide with one of the ranges into which the values of e should be divided.

Thus dividing the linear values of an attribute into too many small ranges results in gross over-specialization. Increasing the size of the ranges reduces over-specialization up to the point where one or more range(s) coincide(s) with the correct range(s), beyond which the attributes become inadequate causing clashing instances and contradictory rules, which again results in over-specialization.

Attribute e ideally should be divided into three ranges, 1–5 secs. incl., 6–10 secs. incl. and 11–20 secs. incl. The method of dividing linear values into equal ranges will never allow these true ranges to be discovered because they are *not* equal. Even in cases where the true ranges are equal, the exact number of ranges is unknown unless specified by a domain expert, in which case the values can be made discrete prior to induction.

For these reasons the method of dividing linear values into equal ranges was found to be unsatisfactory and was abandoned.

10.2 Iterative binary split

ACLS [42] and ASSISTANT [30] are enhanced versions of ID3 which allow attributes to have linear values (see sections 5.3 and 5.4). The entropy for such attributes is calculated for each value of the attribute contained in the training set, by performing a binary split of the training set at that value. The value for which entropy is minimized is selected for comparison with other attributes. Thus if the training set contained all 20 possible values for attribute e, for each value $V_i (V_i = 1 \ldots 20)$ entropy would be calculated

for a training set divided into two subsets, one with values of e less than V_i and the other with values greater than or equal to V_i. Entropy would thus be calculated 20 times, and the value for which it was minimized would be selected. The attribute could then be selected again for a further binary split if necessary and this could be repeated many times.

A similar approach was tried with PRISM. For each value V_i of attribute e, the training set (table 10.1) was divided into two subsets, one with values of e less than V_i and the other with values of e greater than or equal to V_i. The probabilities $p(\delta_n|e_{<V_i})$ and $p(\delta_n|e_{\geq V_i})$ were calculated for each V_i, the maximum being selected for comparison with all other $p(\delta_n|\alpha_x)(\alpha = a, b, c, d)$. The following rule set was induced:

1. $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \wedge e_{\geq 11} \rightarrow \delta_1$
2. $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_{\geq 18} \rightarrow \delta_1$
3. $a_1 \wedge b_2 \wedge d_2 \wedge e_{>9} \wedge e_{<12} \rightarrow \delta_1$
4. $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{\geq 5} \rightarrow \delta_2$
5. $b_2 \wedge c_1 \wedge d_2 \wedge e_{\geq 12} \rightarrow \delta_2$
6. $a_1 \wedge b_2 \wedge d_2 \wedge e_{\geq 5} \wedge e_{<10} \rightarrow \delta_2$
7. $a_2 \wedge b_1 \wedge d_2 \wedge e_{\geq 16} \wedge e_{<17} \rightarrow \delta_2$
8. $d_1 \rightarrow \delta_3$
9. $b_3 \rightarrow \delta_3$
10. $e_{<5} \rightarrow \delta_3$
11. $a_3 \wedge b_1 \rightarrow \delta_3$
12. $a_2 \wedge e_{<10} \rightarrow \delta_3$
13. $a_3 \wedge b_2 \wedge c_2 \wedge e_{\geq 19} \rightarrow \delta_3$
14. $a_3 \wedge e_{<9} \rightarrow \delta_3$

By comparing these rules with those of table 10.2, it can be seen that many of the rules are the same or very similar. However, rules induced by PRISM are intended to be as general as possible in the sense that each rule should reference the least number of attributes necessary for classification. A rule is specialized only when it is too general to discriminate between classes. With a training set in which one or more attributes have linear values, there are two ways in which specialization can be achieved — another attribute can be selected for inclusion in the rule, or an attribute with linear values can have its range reduced. PRISM tends to choose the latter of these two alternatives when the training set is incomplete because of the lack of counter-examples, which in many cases results in over-specialization with respect to attributes with linear values. An iterative binary split as described above does not prevent this from happening. For example, rules 2, 3, 7 and 13 above each reference attribute *e*, but the selected ranges for *e* are much smaller than they should be, making *e* a highly discriminating attribute.

10.3 Range selection

PRISM constructs rules by identifying the attribute-value pairs, α_x, which are relevant to a class, δ_n, i.e. for which the information gain, $I(\delta_n|\alpha_x)$, is positive (see sections 7.3.1 and 7.3.2). The theory was developed for attributes which have discrete values, but can be extended to apply to attributes with linear values. Information gain is positive when $p(\delta_n|\alpha_x) > p(\delta_n)$, thus if α has linear values, it is necessary to find the range R for which all $p(\delta_n|\alpha_i)(i \in R) > p(\delta_n)$. There may be more than one such range, in which case it is necessary to find the range for which $p(\delta_n|\alpha_R)$ is maximum. The process is straightforward if the training set is complete — $p(\delta_n|\alpha_i)$ is calculated for all α_i and if it is greater than $p(\delta_n)$, i is included in the range. For example, table 10.3 shows $p(\delta_3|e_i)$ for all i for the complete training set of 720 instances, for which $p(\delta_3) = 0.847$. Thus if the value i of attribute e lies within the range 1–10 secs. incl., $p(\delta_3|e_i) > p(\delta_3)$. Therefore, the range of values of e which are relevant to δ_3 is 1–10 secs. incl., and $p(\delta_3|e_{1-10})$ is

$p(\delta_3\|e_1)$	=	1.0
$p(\delta_3\|e_2)$	=	1.0
$p(\delta_3\|e_3)$	=	1.0
$p(\delta_3\|e_4)$	=	1.0
$p(\delta_3\|e_5)$	=	1.0
$p(\delta_3\|e_6)$	=	0.889
$p(\delta_3\|e_7)$	=	0.889
$p(\delta_3\|e_8)$	=	0.889
$p(\delta_3\|e_9)$	=	0.889
$p(\delta_3\|e_{10})$	=	0.889
$p(\delta_3\|e_{11})$	=	0.75
$p(\delta_3\|e_{12})$	=	0.75
$p(\delta_3\|e_{13})$	=	0.75
$p(\delta_3\|e_{14})$	=	0.75
$p(\delta_3\|e_{15})$	=	0.75
$p(\delta_3\|e_{16})$	=	0.75
$p(\delta_3\|e_{17})$	=	0.75
$p(\delta_3\|e_{18})$	=	0.75
$p(\delta_3\|e_{19})$	=	0.75
$p(\delta_3\|e_{20})$	=	0.75

Table 10.3 $p(\delta_3|e_i)(i = 1 \ldots 20)$ for a complete training set

calculated (0.944) for comparison with $p(\delta_3|\alpha_x)$ for all $\alpha(\alpha = a, b, c, d)$.

When the training set is incomplete, however, this process can be highly inaccurate, as $p(\delta_3|e_i) = 0$ if there are no instances of class δ_3 in the training set which have the specific value i for attribute e. If i lies within a relevant range R and the training set contains instances with other values of e (say j and k) within the same range, then it is likely that $p(\delta_3|e_j) > p(\delta_3)$ and $p(\delta_3|e_k) > p(\delta_3)$ resulting in (at least) two highly specific ranges for e.

Therefore, $p(\delta_3|e_i)$ is not calculated for each individual i. Instead, a range r of fixed size is selected, and $p(\delta_3|e_r)$ is calculated for all possible e_r or for as many as is practical. A domain expert must supply the lower and upper limits of the total range of possible values of e (lim_l and lim_u respectively) and the accuracy (Ac) to within which the measurements are taken. For attribute e of the contact lens data, $lim_l = 1$ sec., $lim_u = 20$ secs. and $Ac = 1$ sec. A range of one-fifth of the total range (i.e. 4 secs.) is

$p(\delta_3\|e_{1-4})$	=	1.0
$p(\delta_3\|e_{2-5})$	=	1.0
$p(\delta_3\|e_{3-6})$	=	0.962
$p(\delta_3\|e_{4-7})$	=	0.913
$p(\delta_3\|e_{5-8})$	=	0.870
$p(\delta_3\|e_{6-9})$	=	0.852
$p(\delta_3\|e_{7-10})$	=	0.867
$p(\delta_3\|e_{8-11})$	=	0.844
$p(\delta_3\|e_{9-12})$	=	0.767
$p(\delta_3\|e_{10-13})$	=	0.759
$p(\delta_3\|e_{11-14})$	=	0.679
$p(\delta_3\|e_{12-15})$	=	0.68
$p(\delta_3\|e_{13-16})$	=	0.68
$p(\delta_3\|e_{14-17})$	=	0.621
$p(\delta_3\|e_{15-18})$	=	0.677
$p(\delta_3\|e_{16-19})$	=	0.743
$p(\delta_3\|e_{17-20})$	=	0.794

Table 10.4 $p(\delta_3|e_r)(r = 1-4, 2-5, \cdots, 17-20)$ for an incomplete training set.

selected and $p(\delta_3|e_r)$ calculated for all possible e_r, i.e. for r = 1–4 secs. incl., 2–5 secs. incl., $\cdots$, 17–20 secs. incl. in turn. The lower limit of a relevant range R lies within the first r for which $p(\delta_3|e_r) \geq p(\delta_3)$. This is recorded and $p(\delta_3|e_r)$ continues to be calculated for each successive r until $p(\delta_3|e_r) < p(\delta_3)$. The upper limit of the relevant range lies within the preceding r. The actual limits of R are taken to be the mid-point of each respective r, or lim_l and/or lim_u if r is the first or last range, respectively. $p(\delta_3|e_R)$ is calculated and then $p(\delta_3|e_r)$ for all remaining r in search of further relevant ranges. The maximum $p(\delta_3|e_R)$ is selected for comparison with $p(\delta_3|\alpha_x)$ for attributes a, b, c and d. Table 10.4 shows the values of $p(\delta_3|e_r)$ for all r when the above algorithm is applied to the incomplete training set of table 10.1, for which $p(\delta_3) = 0.826$. The relevant range R is 1–9 secs. incl. for which $p(\delta_3|e_R) = 0.937$.

Once attribute e has been selected for inclusion in a rule and its relevant range R has been determined, the limits of R are fixed and any future calculations of $p(\delta_n|e_r)$ take place within these already established limits. This

is because once a range has been selected, a rule induced and the appropriate instances removed from the training set, any range r which contains a limiting value of R is biased away from δ_n. The problem is compounded by an increase in error when approximating probability by relative frequency of occurrence as the training set becomes smaller. Therefore, limits selected early on in the induction process for each class are most likely to be correct. Thus if a range limit has been established at 10 secs., $p(\delta_3|e_r)$ is calculated for the ranges 1–4, $\cdots$, 7–10 and again for the ranges 11–14, $\cdots$, 17–20. This tends to prevent a second limit being selected at 8 secs., 9 secs., 11 secs. or 12 secs., and so stabilizes internal limits to a certain degree.

When a value is measured to a high degree of accuracy, there may be many possible ranges, r, for attribute α. In such cases it would not be practical to calculate $p(\delta_n|\alpha_r)$ for all r, so it is necessary to determine first by how much the lower limit of r should be increased to find the next r. The number of possible values $= (lim_u - lim_l) \div Ac$ is calculated. If it is greater than 50, r is increased by $(lim_u - lim_l)/50$, rounded to the given accuracy, otherwise r is increased by the minimum amount (the given degree of accuracy) each time. The size of r is fixed at $(lim_u - lim_l)/5$, with a minimum of 1.

Thus the algorithm for selecting ranges is as follows:

Step 1 Calculate the number of possible values, the size, s of r and the amount, inc, by which r is to be increased at each step.

Step 2 For the first pair of fixed limits, b_l and b_u (for the first rule for each class, $b_l = lim_l$ and $b_u = lim_u$), calculate $p(\delta_n|\alpha_r)$ for all $\alpha_r(r = b_l \cdots (b_l+s), (b_l+inc) \cdots (b_l+s+inc), (b_l+2*inc) \cdots (b_l+s+2*inc), \cdots, (b_u - s) \cdots b_u)$.

Step 3 Establish the lower limit R_l of a relevant range at the mid-point of the first r for which $p(\delta_n|\alpha_r) \geq p(\delta_n)$ or at b_l if $r = b_l \cdots (b_l+s)$.

Step 4 Establish the upper limit R_u of the range at the mid-point of the last r before $p(\delta_n|\alpha_r) < p(\delta_n)$ or at b_u if $r = (b_u - s) \cdots b_u$.

Step 5 Calculate $p(\delta_n|\alpha_R)$.

Step 6 Repeat steps 3, 4 and 5 until $r = (b_u - s) \cdots b_u$.

Step 7 Repeat steps 2–6 for each pair of fixed limits b_u and b_l.

Step 8 Select the R for which $p(\delta_n|\alpha_R)$ is maximum.

If α_R is selected for inclusion in a rule the limits of R are fixed as internal limits for any subsequent calculations.

Using this algorithm PRISM induced the following set of rules from the complete training set of 720 instances:

1. $b_1 \wedge c_2 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_1$
2. $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_1$
3. $b_2 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$
4. $a_1 \wedge b_1 \wedge d_2 \wedge e_{6-10} \rightarrow \delta_2$
5. $a_1 \wedge b_2 \wedge d_2 \wedge e_{6-10} \rightarrow \delta_2$
6. $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$
7. $a_2 \wedge b_1 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$
8. $d_1 \rightarrow \delta_3$
9. $b_3 \rightarrow \delta_3$
10. $e_{1-5} \rightarrow \delta_3$
11. $a_3 \wedge e_{6-10} \rightarrow \delta_3$
12. $a_2 \wedge e_{6-10} \rightarrow \delta_3$
13. $a_3 \wedge b_2 \wedge c_2 \rightarrow \delta_3$
14. $a_3 \wedge b_1 \wedge c_1 \rightarrow \delta_3$
15. $a_2 \wedge b_2 \wedge c_2 \rightarrow \delta_3$

These rules match those induced from table 9.1 precisely.

The following rules are those induced when the algorithm was applied to the incomplete training set of table 10.1:

1. $b_1 \wedge c_2 \wedge d_2 \wedge e_{13-20} \rightarrow \delta_1$

2. $a_1 \wedge b_1 \wedge c_2 \wedge d_2 \rightarrow \delta_1$

3. $a_1 \wedge b_2 \wedge d_2 \wedge e_{17-20} \rightarrow \delta_1$

4. $a_1 \wedge b_2 \wedge c_2 \wedge d_2 \wedge e_{8-12} \rightarrow \delta_1$

5. $a_1 \wedge b_1 \wedge c_1 \wedge d_2 \rightarrow \delta_2$

6. $b_2 \wedge c_1 \wedge d_2 \wedge e_{11-20} \rightarrow \delta_2$

7. $a_1 \wedge b_2 \wedge d_2 \wedge e_{5-10} \rightarrow \delta_2$

8. $a_2 \wedge b_1 \wedge c_1 \wedge e_{15-17} \rightarrow \delta_2$

9. $d_1 \rightarrow \delta_3$

10. $b_3 \rightarrow \delta_3$

11. $e_{1-5} \rightarrow \delta_3$

12. $a_3 \wedge b_1 \rightarrow \delta_3$

13. $a_2 \wedge e_{6-9} \rightarrow \delta_3$

14. $a_3 \wedge c_2 \rightarrow \delta_3$

15. $a_3 \wedge e_{6-9} \rightarrow \delta_3$

These rules were induced without employing a specialization procedure to maintain consistency, and contain inaccuracies, only some of which can be corrected by specialization. Other inaccuracies can be attributed to the ad hoc method of selecting boundaries for the ranges of attribute *e*, combined with an increase in error when approximating probabilities by relative frequencies of occurrence when small ranges are considered. For example,

whilst inducing rule 1 above, the lower limit of the relevant range R of attribute e was found to lie within the range 11–14 secs. incl. The mid-point of this range was selected and R was fixed at 13–20 secs. incl. The resulting rule ($b_1 \wedge c_2 \wedge d_2 \wedge e_{13-20} \rightarrow \delta_1$) covers instances nos. 17, 18, 19, 20, 68 and 69 of table 10.1, but excludes instance no. 16 in which the value of attribute e is 12. This in turn causes rule 2 ($a_1 \wedge b_1 \wedge c_2 \wedge d_2 \rightarrow \delta_1$) to be induced. Had R been fixed at 11–20 secs. incl. or 12–20 secs. incl., rule 1 would have been slightly more accurate and rule 2 would never have been induced.

The size s of each range r was fixed at one-fifth of the possible range by trial and error using the contact lens data set. The smaller s becomes the more specific are the ranges of e in induced rules because of the increased likelihood that statistical errors are included. On the other hand, the larger the range the smaller the difference between successive $p(\delta_n|\alpha_r)$ and the more likely it is that a true limit does not lie at exactly the mid-point of r. Furthermore, if there are any true ranges which are smaller than r the attributes become inadequate. This is illustrated by rule 4 above. Rule 4 was induced to cover instance no. 37 in table 10.1, in which the value of attribute e is 11. Because 13 secs. was fixed as an internal limit whilst inducing rule 1, and because the size s of r is fixed at 4 secs., the relevant range of attribute e for rule 4 was found to be 8–12 secs. incl. This was the smallest range which could be found to include 11 secs., i.e. the value of e in instance no. 37. The result is that rule 4 also covers instances nos. 35 and 36, and classifies them incorrectly. Had the internal limit been fixed at 11 secs. instead of at 13 secs., the relevant range for e would have been found to be 11–14 secs. incl., making rule 4 considerably more accurate. Alternatively, had s been fixed at 2 secs., the relevant range for e would have been 10–12 secs. incl. and instances nos. 35 and 36 would not have been classified incorrectly.

For these reasons, the method of range selection described in this subsection was not thought to be sufficiently accurate or robust to be incorporated into PRISM.

10.4 Summary

This chapter has described three ways in which PRISM might be modified to enable it to handle linear values.

The first method was to divide the values into a fixed number of equal ranges. However, this was found to be unsatisfactory because unequal ranges could never be discovered. Even in cases where the true ranges are equal, the exact number of ranges is unknown unless specified by a domain expert, in which case the values can be made discrete prior to induction.

The second potential method of handling linear values was to use an iterative binary split as employed by ACLS [42] and ASSISTANT [30]. This method was found to be unsatisfactory because some values, particularly those near the limits of the total possible range or those which occur infrequently, tend to be highly discriminatory, causing PRISM to select linear-valued attributes with highly specific ranges in preference to discrete-valued attributes.

The third, and probably most promising, potential method of handling linear values was to use a window of fixed size to scan sequentially the entire range of values in order to determine a range R for which $p(\delta_n|\alpha_R)$ is maximum. However, the procedure contained an ad hoc method of determining the precise limits of R, causing slight inaccuracies which tended to be propagated throughout other rules. Thus, this method was thought to be not yet sufficiently robust to be incorporated into PRISM.

Consequently, PRISM currently can handle only discrete values. If it is to be applied to a training set containing linear-valued attributes, the appropriate ranges of these attributes must first be specified by a domain expert, and the values converted to discrete groups prior to induction.

Chapter 11

Conclusions

11.1 Discussion

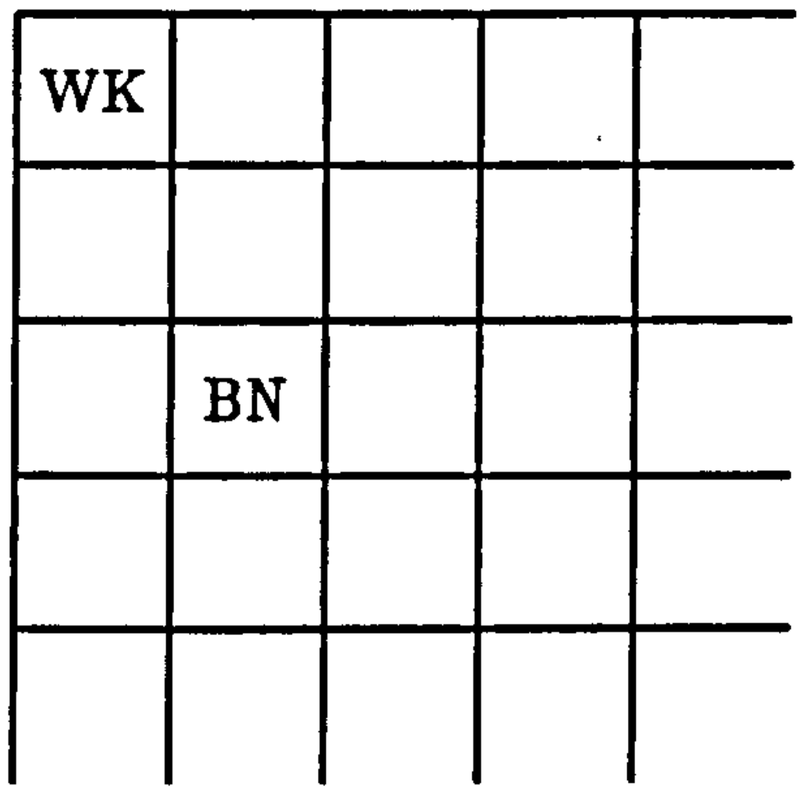
The process of building the knowledge base for an expert system can be long and tedious, both for the knowledge engineer who is building the system and for the domain expert supplying the knowledge. Even if the knowledge engineer and domain expert are one and the same person, a complex domain makes the task non-trivial. If the expert underestimates the effort required to build the knowledge base, although initially he may be most willing, in time he may lose his enthusiasm and become reluctant to continue. Much recent research has been aimed at easing the elicitation process, but often the expert is still expected to provide all the details, to essentially spell out all he knows about the domain. A computer program which learns from examples, thereby releasing the expert from much of the time-consuming task of iterative refinement of the knowledge base, may be able to ease this burden.

Arguably, the most popular and successful algorithm to address this problem is Quinlan's ID3 [44,46,47,48], which induces classification rules in the form of decision trees and is used as the basis of many modern commercially available induction programs. Another program which is claimed to have had reasonable success is Michalski's AQ11 (described in Chapter 4 and [35,36]) which induces modular classification rules. Programs such as these offer an alternative to the currently popular but laborious method

of rule elicitation by interviewing or task and protocol analysis and then iteratively building, testing and refining a knowledge base until it appears to perform satisfactorily. Although automatically induced rules may not be totally accurate, the process of debugging a set of approximate rules is generally much easier for a domain expert than it is for him to conceive even approximate rules. Thus with their promise to minimize inefficient use of human resources in the construction of knowledge bases for rule based expert systems, the potential usefulness of such programs is high.

Obviously, such programs cannot be used in all cases — the representation language must be suited to the domain, examples must be available or readily made available, etc., and of course, it is unlikely that all the knowledge necessary for an application can be so induced. Rule induction programs seek to reduce demand on the expert by transferring the burden of responsibility of rule formation and refinement to a computer program, but the expert's role cannot be completely automated. For a program to be able to acquire new knowledge, it must start with some basic information about the domain in which it is expected to operate. It is the expert who must provide this information, e.g. information about the domain structure, important concepts, attributes, their possible values, which itself can be a major source of difficulty.

For example, the identification of an adequate set of attributes may prove to be a time-consuming and laborious task. Attributes must be defined in such a way as to make it possible to discriminate between the classifications and to enable significant compression of the data, and the expert must see them as meaningful concepts. This problem manifests itself in different ways for different types of domain. The chess endgame used by Quinlan in [44] typifies the situation where any single instance can be described in a number of distinct ways, and attributes are not pre-defined. Consider, for example, the following board position:



There are many possible ways of describing this position (other than specifying the coordinates of the squares that the pieces occupy):

- The white king is on a corner square; the black knight is one square away from an edge.
- The distance between the white king and black knight is two king moves.
- The distance between the white king and black knight is one knight move.
- The black knight checks white.
- The black knight is in a rank or file adjoining the white king.
- The white king can move next to the black knight.

This list grows rapidly as the level of abstractness of the descriptions is increased. The expert is faced with the dilemma of selecting those descriptions which are 'best' for a particular classification problem, when the definition of 'best' depends on the classification itself and on the actual instance being described. Fortunately, descriptive attributes often present themselves naturally. AQ11, in its application to soybean disease classification, uses a set of attributes which are simply symptoms of a diseased plant or external factors affecting the growth of the plant. The terminology used is the standard terminology of plant pathologists, and relatively easy to define.

However, AQ11 also requires extra domain-specific information. Its highly complex representation language employs many different types of operator, not all of which apply to all attributes. Attributes can have values which are discrete, linear or structured and have to be classified accordingly. In the case of structured attributes, the typical structure and any constraints on combinations of values have to be specified, either in the form of rules or otherwise, and the expert may need to be familiar with formal set manipulation procedures. This again requires him to go through the iterative process of refine and test, the problem being that it is not obvious when enough information has been supplied.

An examination of some of the rules derived by AQ11 reveals the nature of the difficulty. A number of rules contain the disjunction (leaves = normal) $\vee$ (leaf malformation = absent) [35] (Rules D3 and D5. Other rules contain similar disjunctions.) However, the first term (leaves = normal) is redundant as it is a generalization of the second term. AQ11 contains information showing that these two attributes are related, but the type of relationship is not described explicitly enough for the program to recognize the generalization.

Other types of background information are also frequently required. For example, several of the derived rules include a term specifying that leaves, stem or some other factor is normal. However, unless this is a specific requirement of the disease, its inclusion is unnecessary and could even prohibit diagnosis in cases where a plant has more than one disease. This indicates that it is necessary for the expert (using AQ11) to state which attributes are relevant and which are irrelevant to each classification. It is also possible that where a classification can be described in a number of ways, an attribute may be relevant to only one or some, but not to other descriptions.

Demand on the expert is increased still further if he is expected to provide control information. For example, AQ11 employs an algorithm which relies on user-specified criteria for limiting search of implausible hypotheses. Many hypotheses are generated during the process of induction, only a few of which can be retained. The expert must decide how these hypotheses are to be

selected, by choosing from a list which includes such criteria as *minimize the number of terms* and *maximize the number of positive examples covered.* Unless the expert is familiar with the details of the algorithm, this sort of decision is quite difficult to make and prone to errors.

These demands on the expert are not much reduced once a set of rules has been induced. It is likely that one or more of the rules will be faulty and need correction. The expert then has to decide where the source of each error may lie. It may be that critical instances have been omitted from the training set; or it may be that not enough domain-specific information has been supplied, or that it is not specific enough; or the fault may lie with the selection of plausible hypotheses. The task is not trivial.

If automatic rule induction systems are to be used for knowledge acquisition for expert systems, they must reduce significantly the time and effort required of the domain expert. A system which releases the expert from the task of defining rules, but requires him to learn a complex representation language or to define search techniques for a control structure with which he may be unfamiliar is not likely to be popular, irrespective of the elegance of its induction algorithm.

The representation language used by ID3 is much simpler. Although this places constraints on the types of domain to which the algorithm can be applied, in those domains which are suitable, the expert's role is greatly simplified. Having decided that parts of the domain may be amenable to rule induction, the expert's role would be simply to identify a set of attributes and supply sufficient examples. However, as explained in Chapter 6, new problems arise, problems associated with a decision tree representation. Decision trees are difficult to manipulate. They contain a lot of information, much of which may be irrelevant to the classification, but is included to maintain structure, which in turn is often unnecessary or even damaging. If a set of modular rules is required, the expert is left with the difficult task of dismantling the tree, which may involve identifying and removing redundant nodes, identifying common branches or parts of branches and selecting appropriate generalizations. This can be a daunting task, particularly for

large and complex trees, and especially so if they contain errors. The expert is again required to iteratively test and debug the knowledge base until he feels it is satisfactory, with the added difficulty of having to simplify a decision tree at perhaps every iteration.

PRISM was designed as an alternative to ID3 for use when the knowledge base would be better expressed in modular rule form rather than as a decision tree. As with all rule induction systems, if the training set is incomplete, PRISM's output cannot be guaranteed to be totally error-free. But because PRISM searches only for necessary and sufficient attributes, and tends to capture the essence of causality, any errors should be easily identifiable. The advantage is that PRISM's output can be examined one rule at a time. If a faulty rule is identified, it can either be corrected by hand or a counter-example can be added to the training set and the program re-run. In this way rapid progress can be made, as the inclusion of a new instance which supplies new information usually has an immediate beneficial effect on the quality of the rules. Consequently, PRISM provides a starting point much further along in the elicitation process than starting with nothing or starting with an inappropriate representation. The laborious task of testing and refining a knowledge base can be simplified considerably without introducing an expensive overhead to deal with the difficulties of a complex representation language or control structure.

The development of PRISM is still in its early stages. Nevertheless, its performance has been shown to better that of ID3 in some domains (see section 9.6). The predictive power of induced rules is an important issue. It will most often be the case that rules induced from incomplete training sets will be expected to predict the class of unseen instances. ID3 performs very well in this respect — in the chess endgame referred to in section 9.6 a decision tree induced from 10% of the complete data set classifies more than 92% of instances correctly. PRISM, however, performs better still. A set of rules induced by PRISM classifies the same number of instances correctly and significantly fewer incorrectly. PRISM's rules are also less specific, indicating that the goal of avoiding redundancy has been achieved

without sacrificing predictive power.

Of course, PRISM can only induce modular rules. It cannot induce decision trees; nor can it induce structural descriptions at the present time. Its representation language is not as rich as, for example, that of AQ11. But PRISM is not intended to be a universal induction program, applicable in all situations. Indeed, it was designed for use only in those situations where simple modular classification rules are required and a sufficient number of examples are available. However, examples of such applications are numerous. The ID3 algorithm has been used in, amongst others, medical, engineering and business domains, some of which might be better represented in modular rule form rather than as a decision tree. Because PRISM induces modular rules from training sets which are identical to those used by ID3, it is potentially more suitable than ID3 to these applications.

However, PRISM cannot yet deal with attributes with linear values, nor has its performance in the presence of noise been evaluated. Thus some further work on PRISM has to be undertaken before it can be used for complex real-world applications. This futher research is discussed below.

11.2 Directions for further research

The project reported in this thesis was concerned with designing an induction algorithm with a sound theoretical basis for inducing modular classification rules from sets of examples. The work was completed successfully and PRISM performs exceptionally well when applied to a training set of high quality examples. However, real-world problems do not always present examples which are of high quality. Very often, data is noisy, uncertain or ambiguous, and the most obvious next step in the development of PRISM is to assess its performance in these cases. Quinlan [45] has shown that the presence of noise or uncertainty in data can have a significant impact on the quality of induced rules. No algorithm can be expected to perform well with very poor data, but it is essential to ensure that any degradation in performance is graceful; that the algorithm is robust enough to cope with

levels of noise which can be reasonably expected in the real world. PRISM, in its current form, has not been designed to deal with noise, and will almost certainly require some modification — at least some sort of stopping criterion may need to be included if its output is to remain dependable. Further research is necessary to determine precisely how errors are caused by different types of noise and thus what sorts of preventative measures are likely to be effective in minimizing these errors.

Uncertainty or ambiguity in the data can arise at all levels, e.g. it may be that the value of an attribute is uncertain, or it may not be possible to classify one or more instances with certainty, or two experts may disagree on either of these or other points. Because PRISM has an information theoretic foundation, it should be amenable to modification to allow induction under uncertainty.

PRISM's ability to resolve ambiguity caused by contradictory rules seems highly successful in the domain to which it was applied. However, clashes may need to be resolved in different ways for different domains, or perhaps a domain expert may prefer clashes to remain unresolved but to include some measure of uncertainty. An assessment needs to be made of the relative merits of different approaches and their applicability to different domains.

Further work is also necessary to enable PRISM to deal with attributes which have linear values. Chapter 10 described three ways in which this might be done. Of these, the third method — that of range selection — appeared to be the most promising. However, this method employed an ad hoc procedure for determining the precise limits of a relevant range, and although the algorithm seemed quite good at selecting appropriate ranges, this slight imprecision tended to cause errors which were propagated throughout the whole induction process. Further research is necessary to determine a better way of fixing limits and a way of preventing the propagation of errors. Alternatively, the second method, that of iterative binary split, might be found to be appropriate if the algorithm could be prevented from performing the split in such a way as to isolate a single value or very small range of values, thus causing over-specialization.

Other related work could include, for example, the designing of a PRISM-like algorithm to induce structural descriptions, or for incremental induction. A useful development might be a system which uses PRISM to perform structured induction in a similar way to that described by Shapiro in [52].

On a broader front, there is still much research to be done in the general field of knowledge elicitation for expert systems. It has been recognized that modern expert systems should not depend on a single representational form, that different parts of the domain may each need to be represented differently, for which a knowledge engineer may need to call upon different elicitation techniques. At the present time, knowledge engineering is still very much an art. There is no set theory as to how the task should be performed. Indeed, there are no guidelines for choosing the best representational form for a domain or part of a domain, let alone for eliciting knowledge in that form. The definition of such a set of guidelines, although a major undertaking, could make a useful contribution to the field. Armed with a basic methodology, knowledge engineers would no longer be pioneers in a new field of expertise, but true expert practitioners in the domain of knowledge engineering. As such, they should have to hand some basic tools of the trade. They should not be expected to make each tool afresh as it is needed, but to be able to use tools which are readily available and either perfectly suited to the job in hand or needing no more than minor adjustment. PRISM could be one such tool.

Bibliography

[1] T. R. Addis. *Designing Knowledge-Based Systems.* Kogan Page, London, 1985.

[2] J. L. Alty and M. J. Coombs. *Expert Systems, Concepts and Examples.* NCC Publications, Manchester, England, 1984.

[3] A. Barr and E. A. Feigenbaum. *The Handbook of Artificial Intelligence.* Volume 1, William Kaufmann, Los Altos, 1981.

[4] A. Barr and E. A. Feigenbaum. *The Handbook of Artificial Intelligence.* Volume 2, William Kaufmann, Los Altos, 1982.

[5] A. H. Bond, editor. *Machine Learning: Infotech State of the Art Report. Series 9 No. 3.* Pergamon Infotech Ltd., 1981.

[6] M. A. Bramer. Automatic induction of rules from examples: a critical analysis of the ID3 family of rule induction systems. In *Proceedings of the First European Workshop on Knowledge Acquisition*, Reading, England, September 1987.

[7] M. A. Bramer, editor. *Research and Development in Expert Systems: Proceedings of the Fourth Technical Conference of the British Computer Society Specialist Group on Expert Systems, University of Warwick*, Cambridge University Press, Cambridge, England, 1984.

[8] M. A. Bramer. A survey and critical review of expert systems research. In D. Michie, editor, *Introductory Readings in Expert Systyems*, pages 3–29, Gordon and Breach, London, 1982.

[9] B. G. Buchanan. *Research on Expert Systems.* Technical Report STAN-CS-81-837, Stanford University, Dept. of Computer Science, Stanford University, Stanford, CA, 1981.

[10] B. G. Buchanan. *Some Approaches to Knowledge Acquisition.* Technical Report STAN-CS-85-1076, Stanford University, Dept. of Computer Science, Stanford University, Stanford, CA, 1985.

[11] B. G. Buchanan and E. A. Feigenbaum. DENDRAL and meta-DENDRAL: their applications dimension. *Artificial Intelligence*, 11:5–24, 1978.

[12] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An overview of machine learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Tioga, Palo Alto, 1983.

[13] J. Cendrowska. PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27:349–370, 1987.

[14] J. Cendrowska and M. A. Bramer. A rational reconstruction of the MYCIN consultation system. *International Journal of Man-Machine Studies*, 20:229–317, 1984.

[15] P. R. Cohen and E. A. Feigenbaum, editors. *The Handbook of Artificial Intelligence.* Volume 3, William Kaufmann, Los Altos, 1982.

[16] R. A. Corlett. Explaining induced decision trees. In *Expert Systems 83: Proceedings of the Third Technical Conference of the British Computer Society Specialist Group on Expert Systems*, 1983.

[17] R. Davis. *Applications of Meta-Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases.* Technical Report STAN-CS-76-552, Stanford University, Computer Science Dept., Stanford University, Stanford, 1976.

[18] R. Davis and J. King. An overview of production systems. In E. W. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 300–332, Edinburgh University Press, Edinburgh, 1976.

[19] R. O. Duda, J. Gaschnig, and P. E. Hart. Model design in the prospector consultant program for mineral exploration. In D. Michie, editor, *Expert Systems in the Microelectronic Age*, Edinburgh University Press, Edinburgh, 1979.

[20] E. Edwards. *Information Transmission.* Chapman and Hall, London, 1964.

[21] K. Ericsson and H. A. Simon. Verbal reports on data. *Psychological Review*, 87:215–251, 1980.

[22] E. A. Feigenbaum. Expert systems in the 1980s. In A. H. Bond, editor, *Machine Learning: Infotech State of the Art Report. Series 9 No. 3*, Pergamon Infotech Ltd., 1981.

[23] E. A. Feigenbaum. Knowledge engineering: the applied side. In J. E. Hayes and D. Michie, editors, *Intelligent Systems: The Unprecedented Opportunity*, Ellis Horwood Ltd., Chichester, England, 1984.

[24] E. A. Feigenbaum. Themes and case studies of knowledge engineering. In D. Michie, editor, *Expert Systems in the Microelectronic Age*, Edinburgh University Press, Edinburgh, 1979.

[25] P. E. Friedland. *Knowledge-Based Experimental Design in Molecular Genetics.* PhD thesis, Heuristic Programming Project, Computer Science Dept., Stanford University, Stanford, CA, 1979.

[26] B. R. Gaines. An overview of knowledge-acquisition and transfer. In B. R. Gaines and J. H. Boose, editors, *Knowledge Acquisition for Knowledge-Based Systems*, Academic Press Ltd., London, 1988.

[27] S. Goldman. *Information Theory.* Dover Publications, New York, 1968.

[28] A. E. Hart. Experience in the use of an inductive system in knowledge engineering. In M. A. Bramer, editor, *Research and Development in Expert Systems: Proceedings of the Fourth Technical Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge University Press, Cambridge, 1985.

[29] E. B. Hunt, J. Marin, and P. J. Stone. *Experiments in Induction.* Academic Press, New York, 1966.

[30] I. Kononenko, I. Bratko, and E. Roskar. *Experiments in Automatic Learning of Medical Diagnostic Rules.* Technical Report, Jozef Stefan Institute, Ljubljana, Yugoslavia, 1984.

[31] J. Kunz et al. *A Physiological Rule-Based System for Interpreting Pulmonary Function Test Results.* Technical Report HPP-78-19, Heuristic Programming Project, Computer Science Dept., Stanford University, Stanford CA., 1978.

[32] J. McDermott. R1: a rule-based configurer of computer systems. *Artificial Intelligence*, 19:39–88, 1982.

[33] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2), February 1983.

[34] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach.* Tioga, Palo Alto, 1983.

[35] R. S. Michalski and R. L. Chilausky. Knowledge acquisition by encoding expert rules versus computer induction from examples : a case study involving soybean pathology. *International Journal of Man-Machine Studies*, 12, 1980.

[36] R. S. Michalski and J. B. Larson. *Selection of Most Representative Training Examples and Incremental Generation of VL1 Hypotheses: The Underlying Methodology and the Description of Programs ESEL and AQ11.* Technical Report 867, Computer Science Dept. University of Illinois, Urbana, 1978.

[37] D. Michie. Current developments in expert systems. In J. R. Quinlan, editor, *Applications of Expert Systems*, Addison-Wesley, Maidenhead, 1987.

[38] D. Michie. Inductive rule generation in the context of the fifth generation. In R. S. Michalski, editor, *Proceedings of the International Machine Learning Workshop*, University of Illinois, June 1983.

[39] T. M. Mitchell. Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 305–310, MIT, Cambridge, Mass., 1977.

[40] N. J. Nilsson. *Principles of Artificial Intelligence.* Springer-Verlag, 1980.

[41] P. O'Rorke. *A Comparative Study of Inductive Learning Systems AQ11P and ID-3 Using a Chess Endgame Test Problem.* Technical Report UIUCDCS-F-82-899, Dept. of Computer Science, University of Illinois, September 1982.

[42] A. Paterson and T. Niblett. *ACLS User Manual.* Technical Report, Intelligent Terminals Limited, Glasgow, 1982.

[43] J. R. Quinlan. Decision trees as probabilistic classifiers. In P. Langley, editor, *Proceedings of the Fourth International Workshop on Machine Learning*, Morgan Kaufmann, Los Altos, CA, June 1987.

[44] J. R. Quinlan. Discovering rules by induction from large collections of examples: a case study. In D. Michie, editor, *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press, 1979.

[45] J. R. Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, Los Altos, California, 1986.

[46] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[47] J. R. Quinlan. *Induction Over Large Data Bases.* Technical Report HPP - 79 - 14, Heuristic Programming Project, Stanford University, 1979.

[48] J. R. Quinlan. 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*, Tioga, Palo Alto, 1983.

[49] J. R. Quinlan. Learning from noisy data. In R. S. Michalski, editor, *Proceedings of the International Machine Learning Workshop*, University of Illinois, June 1983.

[50] J. R. Quinlan, P. J. Compton, K. A. Horn, and L. Lazarus. Inductive knowledge acquisition: a case study. In J. R. Quinlan, editor, *Applications of Expert Systems*, Addison-Wesley, Maidenhead, 1987.

[51] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, 1949. Published in 1964.

[52] A. D. Shapiro. *The Role of Structured Induction in Expert Systems.* PhD thesis, The University of Edinburgh, 1983.

[53] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York, 1976.

[54] H. A. Simon and G. Lea. Problem solving and rule induction: a unified view. In L. Gregg, editor, *Knowledge and Cognition*, pages 105–127, Lawrence Erlbaum Associates, Potomac, Maryland, 1974.

[55] L. Steels. Second generation expert systems. In M. A. Bramer, editor, *Research and Development in Expert Systems III: Proceedings of*

the Sixth Technical Conference of the British Computer Society Specialist Group on Expert Systems, Cambridge University Press, Cambridge, 1986.

[56] M. J. Stefik. *Planning with Constraints.* PhD thesis, Heuristic Programming Project, Computer Science Dept., Stanford University, Stanford, CA, 1980.

[57] M. Welbank. *A Review of Knowledge Acquisition Techniques for Expert Systems.* Technical Report, British Telecom, Martlesham Heath, Ipswich, England, 1983.

[58] P. H. Winston. *Artificial Intelligence.* Addison-Wesley, second edition, 1984.

[59] P. H. Winston. *Learning Structural Descriptions from Examples.* Technical Report AI-TR-231, MIT, Cambridge, Mass., Sept. 1970.

[60] R. M. Young. Role of intermediate representations in knowledge elicitations. In D. S. Moralee, editor, *Research and Development in Expert Systems: Proceedings of Expert Systems 87, the Seventh Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge University Press, Cambridge, 1987.

Appendix A

PRISM

```
{------------------------------------------------------------------------}
{                              PRISM                                     }
{                                                                        }
{          Jadzia Cendrowska       June 1989                             }
{                                                                        }
{    A program for inducing modular rules from examples.                 }
{                                                                        }
{    Inputs : static data base                                           }
{             training set of examples                                   }
{    Output : set of modular rules                                       }
{                                                                        }
{    Language : ProPascal (version iid 2.1) for MS-DOS                   }
{                                                                        }
{------------------------------------------------------------------------}
```

```
program prism;

{globals}

const maxnatt    = 7;        {maximum no. of attributes}
      maxninst   = 720;      {maximum no. of instances}
      maxnpv     = 20;       {maximum no. of possible attribute values}
      maxrul     = 50;       {maximum possible no. of rules}
      namelength = 6;        {maximum length of attribute name}

type  zott       = 0..3;
      byte       = 0..255;
      twobyte    = 0..65535;

      vallist    = array [0..maxnatt] of byte;
                           {used for storing either an instance or a}
                           {rule. Each element specifies the position}
                           {of a value in a list of possible values.}
                           {The zeroth element refers to the class;}
                           {the remaining elements refer to each}
                           {attribute in turn. If used for storing a}
                           {rule, 0 indicates that the attribute is}
                           {irrelevant.}

rule       = record
               abbr       : vallist;          {abbreviated rule}
               lprem      : byte; {no. of clauses in the premise}
               noic, npic : twobyte;
                           {number of actual instances and number of}
                           {possible instances covered by this rule,}
                           {respectively}
               toogeneral : boolean;
                           {indicates whether the rule needs to be}
                           {specialized}
             clashwith  : array[0..maxrul] of byte;
                           {a list of rules which contradict this rule}
```

```
                end; {record}

        att         = record
                        name        : packed array [1..namelength] of char;
                        npv         : byte;     {number of possible values}
                        pdv         : array [1..maxnpv] of twobyte;
                                                {list of possible values}
                      end; {record}

        tspt        = array [0..maxninst] of twobyte;
                                {list of instances. Each element is the}
                                {position of an instance in the training}
                                {set.}

        ft          = array [0..maxnpv,1..maxnpv] of byte;
                                {frequency table used for calculating}
                                {relative information gain}

var     f           : text;

        natt,
        nclass,
        nrul        : byte;    {number of attributes, classes and rules}

        ninst       : twobyte; {number of instances}

        ts          : array [1..maxninst] of vallist;
                                {contains the training set}

        attset      : array [0..maxnatt] of att;
                                {list of attributes. The zeroth attribute}
                                {refers to the class.}

        pt1,
        pt2,
        pt3         : tspt;  {contain various subsets of the training set}
```

```
used        : array [1..maxnatt] of boolean;
                        {indicates which attributes are available}
                        {for selection}

bestrule,
temprule    : rule;     {used for storing rules currently being}
                        {induced.}

rulset      : array[1..maxrul] of rule;      {set of induced rules}

uniq        : array[0..maxninst] of twobyte;
                        {list of instances covered uniquely by rule}
                        {to be specialized}

uvals       : array[0..maxnpv] of byte;
                        {list of values of a specific attribute}
                        {appearing in array uniq}

{-------------------------------------------------------------------}

procedure exitprog(returncode:integer); external;

{===================================================================}
```

```
procedure initdata;
{initialises all data and prepares the training set}

var fil          : string[11];
    ch           : char;
    a, i, j, pos : integer;

begin
    {read in the attribute details from the static database and}
    {complete the attribute records}
    write('Name of static data file? ');
    readln(fil); assign(f,fil); reset(f);
    read(f,natt);

    if natt > maxnatt then
    begin
        writeln('Number of attributes exceeds the maximum allowed');
        exitprog(1);
    end;

for a := 0 to natt do
begin
    readln(f);
    read(f,ch);
    while ch = ' ' do read(f,ch);

    {read in the name of the attribute}
    i := 1;
    attset[a].name[i] := ch;
    read(f,ch);
    while ((ch <> ' ') and (i < namelength)) do
    begin
        i := i+1;
        attset[a].name[i] := ch;
        read(f,ch);
end;
```

```
i := i+1;
while i <= namelength do
begin
    attset[a].name[i] := ' ';
    i := i+1;
end;

    {read the number and list of possible values}
    read(f,attset[a].npv);
    if attset[a].npv > maxnpv then
    begin
        write('Number of possible values of attribute ',a:1);
        writeln(' exceeds the maximum allowed');
        exitprog(1);
    end;
    for i :=1 to attset[a].npv do
        read(f,attset[a].pdv[i]);
end;

nclass := attset[0].npv;
close(f);

{read and prepare the training set}
write('Name of file containing training set? ');
readln(fil); assign(f,fil); reset(f);
read(f,ninst);

if ninst > maxninst then
begin
    writeln('Number of instances exceeds the maximum allowed');
    exitprog(1);
end;

for i := 1 to ninst do
begin
    for j := 1 to natt do
```

```
            read(f,ts[i,j]);
        read(f,ts[i,0]);
end;
close(f);

{modify ts so that each element represents the position of a value}
{in the list of possible values of an attribute, rather than the}
{value itself.}
for i := 1 to ninst do
    for j := 0 to natt do
    begin
        pos := 0;
        repeat
            pos := pos+1;
        until (pos > attset[j].npv) or
              (ts[i,j] = attset[j].pdv[pos]);

        if pos <= attset[j].npv then
            ts[i,j] := pos
        else
        begin
            if j = 0 then
                write('The class value')
            else write('The value of attribute no. ',j:1);
            writeln(' in instance no. ',i:1,' cannot be identified');
            exitprog(1);
        end;
    end;

{prepare a file for results and initialize all structures for}
{holding rule information}
write('Name of file to write to? ');
readln(fil); assign(f,fil); rewrite(f);
for i := 1 to maxrul do
begin
    rulset[i].toogeneral := false;
```

```
        for j := 1 to maxrul do
            rulset[i].clashwith[j] := 0;
        rulset[i].lprem := 0;
    end;
end;

{=============================================================================}

procedure checlass (cc : byte; var w : tspt; var b : zott);
{Checks the classes of instances in the training set w and returns          }
{0, 1, 2 or 3:                                                              }
{0 = the training set is empty                                              }
{1 = all instances are of class cc                                          }
{2 = there are no instances of class cc in the training set w               }
{3 = some but not all instances are of class cc                             }

var i : twobyte;

begin
    b := 0;
    i := 1;
    while (i <= w[0]) and (b <> 3) do
    begin
        if ts[w[i],0] = cc then
            if b = 2 then b := 3
        else b := 1;
        if ts[w[i],0] <> cc then
            if b = 1 then b := 3
        else b := 2;
        i := i+1;
    end;
end;

{-----------------------------------------------------------------------------}

procedure countval (a : byte; var freqtab : ft);
{fills in the 2-dimensional array freqtab, where col is the column and}
```

```
{cls is the row in the array. The elements are as follows, e.g. for}
{attribute A, where A has two possible values, and there are two classes:}
{                ------------------------------------------                }
{                | total no. of instances|total no. of instances |        }
{                |  which have the first | which have the second |        }
{                |       value for A     |      value for A      |        }
{                |-----------------------|-----------------------|        }
{                |    no. of instances   |   no. of instances    |        }
{                |     which have the    |    which have the     |        }
{                |   first value for A   |  second value for A   |        }
{                |   and are of class 1  |  and are of class 1   |        }
{                |-----------------------|-----------------------|        }
{                |    no. of instances   |   no. of instances    |        }
{                |     which have the    |    which have the     |        }
{                |   first value for A   |  second value for A   |        }
{                |   and are of class 2  |  and are of class 2   |        }
{                 ------------------------------------------               }

var i, j, col, cls : byte;
    r              : twobyte;

begin
    for i := 0 to nclass do
        for j := 1 to attset[a].npv do
            freqtab[i,j] := 0;

    r := 1;
    while r <= pt3[0] do
    begin
        col := ts[pt3[r],a];
        cls := ts[pt3[r],0];
        freqtab[cls,col] := freqtab[cls,col]+1;
        r := r+1;
    end;

    for j := 1 to attset[a].npv do
```

```
        for i := 1 to nclass do
            freqtab[0,j] := freqtab[0,j]+freqtab[i,j];
end;

{-------------------------------------------------------------------}

function checkprem (ins : twobyte; var prem : vallist) : boolean;
{returns true if the instance ins is covered by the premise of the}
{rule prem, false otherwise}

var p      : byte;
    checkp : boolean;

begin
    checkp := true;
    p := 1;
    while (p <= natt) and checkp do
        if (prem[p] <>0) and (ts[ins,p] <> prem[p]) then
            checkp := false
        else p := p+1;
    checkprem := checkp;
end;

{-------------------------------------------------------------------}

procedure findcovers;
{counts the no. of instances which are covered by the rule temprule}

var i : twobyte;

begin
    with temprule do
        for i := 1 to ninst do
            if (ts[i,0] = abbr[0]) and checkprem(i,abbr) then
                noic := noic + 1;
end;
```

```
{-------------------------------------------------------------------}

procedure getav (cc : byte; var besta, bestv : byte);
{uses the frequency table freqtab to calculate which attribute-value}
{pair provides the most information about class cc}

var noi, a, j : twobyte;
    info, f   : real;
    ftab      : ft;

begin
    besta := 0;                 {the best attribute so far}
    bestv := 0;                 {its best value so far}
    info := 0;                  {indicates relative information gain}
    noi := 0;                   {the number of instances covered}

    for a := 1 to natt do
        if not(used[a]) then {attribute a is available for selection}
        begin
            countval(a,ftab);   {sets up the frequency table}
            for j := 1 to attset[a].npv do
                if ftab[0,j] <> 0 then
                begin
                    {calculate relative information gain for a}
                    f := ftab[cc,j]/ftab[0,j];
                    if ((abs(f-info) < 0.001) and
                       (ftab[cc,j] > noi)) or
                       (((f-info) > 0.001) and
                       (ftab[cc,j] > 0)) then
                        {attribute a, value j, provides more}
                    {information than besta, bestv}
                    begin
                        info := f;
                        noi := ftab[cc,j];
                        besta := a;
```

```
                                bestv := j;
                            end;
                        end;
                end;
        end;

{-------------------------------------------------------------------------}

procedure selectin (a,v : byte);
{removes from pt3 any instances which do not have value v for}
{attribute a}

var m, n : twobyte;

begin
    m := 1;
    n := 1;
    while m <= pt3[0] do
    begin
        if ts[pt3[m],a] = v then
        begin
            pt3[n] := pt3[m];
            n := n+1;
        end;
        m := m+1;
    end;
    pt3[0] := n-1;
end;

{-------------------------------------------------------------------------}

procedure ftrule (cc : byte);
{induces a rule temprule from the training set represented by pt2}

var besta, bestv : byte;
    p, i         : twobyte;
```

```
    t               : zott;

begin
    {initialize temprule}
    for i := 1 to natt do
    begin
        used[i] := false;
        temprule.abbr[i] := 0;
    end;

    temprule.noic := 0;
    for i := 0 to ninst do
        pt3[i] := pt2[i];   {pt3 will contain only those instances}
                            {covered by temprule. These will later be}
                            {removed from pt2.}
    t := 3;
    p := 0;                 {index for the premise of temprule}
    getav(cc,besta,bestv); {finds the best attribute-value pair}

    while (t = 3) and (besta <> 0) do
    begin
        {add the new term to the premise of temprule}
        p := p + 1;
        used[besta] := true;
        temprule.abbr[besta] := bestv;

        {modify pt3 so that it contains only instances which have}
        {value bestv for attribute besta}
        selectin(besta,bestv);

        {check if all instances in pt3 are of class cc}
        checlass(cc,pt3,t);
        {if not, select the next attribute-value pair}
        if t = 3 then getav(cc,besta,bestv);
    end;
```

```
        temprule.lprem := p;
        temprule.abbr[0] := cc;

        {count the number of instances covered by temprule}
        findcovers;
    end;

{-------------------------------------------------------------------------}

procedure modmts (var w : tspt; var rl : vallist);
{removes from the training set w all instances covered by rule rl}

var i, n : twobyte;

begin
    n := 0;                     {index for new training set}
    for i := 1 to w[0] do       {w[0] contains the no. of instances in w}
        if (ts[w[i],0] <> rl[0]) or not(checkprem(w[i],rl)) then
                                {instance w[i] is not covered by rl}
        begin
            n := n + 1;
            w[n] := w[i];
        end;
    w[0] := n;
end;

{-------------------------------------------------------------------------}

procedure ftrset (cc : byte);
{induces all rules for the class cc}

var t : zott;

begin
    ftrule(cc);                 {induces the first rule}
```

```
bestrule := temprule;

{remove from pt2 all instances covered by temprule}
modmts(pt2,temprule.abbr);

{check if all instances in pt2 are of class cc}
checlass(cc,pt2,t);

{if not, induce the next rule and modify pt2 accordingly}
while t = 3 do
begin
    ftrule(cc);
    modmts(pt2,temprule.abbr);

    {keep the more general rule}
    if (temprule.noic > bestrule.noic) or
       ((temprule.noic = bestrule.noic) and
       (temprule.lprem < bestrule.lprem)) then
        bestrule := temprule;
        checlass(cc,pt2,t);
    end;
end;

{-----------------------------------------------------------------

procedure bestcover;
{adds a rule to rulset and calculates the possible no. of instances}
{covered}

var i : byte;

begin
begin
    nrul := nrul + 1;

    if nrul > maxrul then
    begin
```

```
        writeln('Number of rules exceeds the maximum allowed');
        exitprog(1);
    end;

    rulset[nrul] := bestrule;
    with rulset[nrul] do
    begin
        {calculate the possible number of instances covered}
        npic := 1;
        for i := 1 to natt do
            if abbr[i] = 0 then
                npic := npic * attset[i].npv;
    end;
end;

{-------------------------------------------------------------------------}

procedure consistency (cc : twobyte);
{checks if the latest rule contradicts any rule previously induced,}
{and if so, selects one for specialization}

var i, j  : byte;
    clash : boolean;

begin
    for i := 1 to (nrul-1) do
        {check that rulset[nrul] and rulset[i] conclude about}
        {different classes}
        if (cc <> rulset[i].abbr[0]) then
        begin
            {two rules do not contradict each other only if they}
            {reference different values of the same attribute}
            {otherwise a clash occurs}
            clash := true;
            j := 1;
            while clash and (j <= natt) do
```

```
begin
    if (rulset[i].abbr[j] <> rulset[nrul].abbr[j]) and
       (rulset[i].abbr[j] <> 0) and
       (rulset[nrul].abbr[j] <> 0) then
          clash := false;
    j := j + 1;
end;
{Select the less general rule for specialization. If the}
{two rules are equally general, select nrul unless i has}
{already been selected and nrul has not. Thus if i is}
{specialized because it contradicts a rule other than nrul}
{the clash with nrul may also be removed, and nrul may no}
{longer need to be specialized}
if clash then
    if (rulset[i].npic < rulset[nrul].npic) or
       ((rulset[i].npic = rulset[nrul].npic) and
       rulset[i].toogeneral and
       not(rulset[nrul].toogeneral)) then
        with rulset[i] do
        begin
            toogeneral := true;
            {add nrul to clashwith list for i}
            clashwith[0] := clashwith[0] + 1;
            clashwith[clashwith[0]] := nrul;
        end
else
    with rulset[nrul] do
    begin
        toogeneral := true;
        {add i to clashwith list for nrul}
        clashwith[0] := clashwith[0] + 1;
        clashwith[clashwith[0]] := i;
    end;
    end;
end;
```

```
{-------------------------------------------------------------------------

function clashes (x, y : byte) : boolean;
{returns true if the two rules x and y contradict each other,}
{false otherwise}

var a  : byte;
    cl : boolean;

begin
    a := 1;
    if rulset[x].abbr[0] = rulset[y].abbr[0] then
        cl := false
    else cl := true;

    {two rules contradict each other if the classes are different and}
    {each attribute is either not specified in one or both rules, or}
    {has the same value in both rules}
    while cl and (a <= natt) do
        if (rulset[x].abbr[a] <> rulset[y].abbr[a]) and
           (rulset[x].abbr[a] <> 0) and (rulset[y].abbr[a] <> 0) then
            cl := false
        else a := a + 1;

        clashes := cl;
end;

{-------------------------------------------------------------------------

procedure finduniq (r : byte);
{finds all instances which are covered uniquely by rule r and stores}
{them in array uniq}

var i, n    : twobyte;
    j       : byte;
    covered : boolean;
```

```
begin
    uniq[0] := 0;        {uniq[0] = no. of instances covered uniquely}

    {first find all instances covered by rule r}
    for i := 1 to ninst do
        if (ts[i,0] = rulset[r].abbr[0]) and
           checkprem(i,rulset[r].abbr) then
begin
    uniq[0] := uniq[0] + 1;
    uniq[uniq[0]] := i;
end;

{next remove from uniq all instances which are covered by}
{some other rule}
n := 0;
for i := 1 to uniq[0] do
begin
    j := 1;
    covered := false;
    while ((j <= nrul) and not(covered)) do
        if ((j <> r) and (ts[uniq[i],0] = rulset[j].abbr[0]) and
           checkprem(uniq[i],rulset[j].abbr)) then
            covered := true
        else j := j + 1;
        if not(covered) then
        begin
            n := n + 1;
            uniq[n] := uniq[i];
        end;
end;
    uniq[0] := n;
end;

{------------------------------------------------------------------------}
```

```
procedure findbestatt (r : byte; var a : byte);
{selects from the available attributes the attribute which occurs most}
{frequently in the rules which contradict rule r}

var i, x, inu, anu : byte;

begin
    a := 0;                    {a is the most frequently used attribute}
    anu := 0;                  {anu is the number of times a is used}

    for i := 1 to natt do
    if rulset[r].abbr[i] = 0 then
                               {attribute i is available for selection}
    begin
        inu := 0;              {inu is the number of times i is used}
        for x := 1 to rulset[r].clashwith[0] do
            {rulset[r].clashwith[x] is a rule which clashes with r}
            if rulset[rulset[r].clashwith[x]].abbr[i] <> 0 then
                {attribute i is used in this rule}
                inu := inu + 1;
            if inu > anu then
            {attribute i is used more often than attribute a}
            begin
                a := i;
                anu := inu;
            end;
    end;
end;

{-------------------------------------------------------------------}

procedure findvals (a : byte);
{stores in array uvals all values of attribute a which are used in the}
{set of instances comprising uniq}

var i : twobyte;
```

```
    j : byte;

begin
    uvals[0] := 0;              {number of elements in uvals}
    for i := 1 to uniq[0] do
    begin
        {search uvals for the value of a in instance uniq[i]}
        j := 1;
        while (j <= uvals[0]) and (ts[uniq[i],a] <> uvals[j]) do
            j := j + 1;

        {if the value is not already there, add it to uvals}
        if j > uvals[0] then
        begin
            uvals[j] := ts[uniq[i],a];
            uvals[0] := uvals[0] + 1;
        end;
    end;
end;

{-------------------------------------------------------------------}


procedure specialize;
{specializes rules to remove all contradictions}

var r, n, i, a : byte;

begin
    r := 1;
    while r <= nrul do
    begin
        if rulset[r].toogeneral then
        {remove from clashwith list all rules which no longer}
        {contradict rule r}
        begin
            n := 0;
```

```
for i := 1 to rulset[r].clashwith[0] do
    if (rulset[r].clashwith[i] > r) or
       clashes(rulset[r].clashwith[i],r) then
    begin
        n := n + 1;
        rulset[r].clashwith[n] := rulset[r].clashwith[i];
    end;
rulset[r].clashwith[0] := n;

if rulset[r].clashwith[0] = 0 then
    rulset[r].toogeneral := false
else    {specialize rule r}
    if rulset[r].npic > 1 then
    {r is not maximally specific. A maximally specific}
    {rule only cotradicts another rule if there are}
    {contradictory instances in the training set}
    repeat
        {find all instances covered uniquely by r}
        finduniq(r);

{if there are no instances covered uniquely}
{this rule can be removed and so does not}
{need to be specialized}
if uniq[0] > 0 then
begin
    {select an attribute}
    findbestatt(r,a);

    {search uniq for values of a}
    findvals(a);

    {specialize r by adding attribute a, taking}
    {the first value for a which appears in uvals}
    rulset[r].abbr[a] := uvals[1];
    rulset[r].lprem := rulset[r].lprem + 1;
    rulset[r].npic := round(rulset[r].npic/
```

```
                                                attset[a].npv);

{create a new rule for every other value}
{in uvals}
for i := 2 to uvals[0] do
begin
    nrul := nrul + 1;
    if nrul > maxrul then
    begin
writeln('Number of rules exceeds the maximum allowed');
        exitprog(1);
    end;
    rulset[nrul] := rulset[r];
    rulset[nrul].abbr[a] := uvals[i];
end;

{remove from the clashwith list all rules}
{which no longer contradict rule r}
n := 0;
for i := 1 to rulset[r].clashwith[0] do
if clashes(rulset[r].clashwith[i],r) then
begin
    n := n + 1;
    rulset[r].clashwith[n] :=
                                rulset[r].clashwith[i];
end;
rulset[r].clashwith[0] := n;

{if there are no more contradictory rules,}
{rule r does not need to be specialized}
{further}
        if rulset[r].clashwith[0] = 0 then
            rulset[r].toogeneral := false;
    end;
until not(rulset[r].toogeneral) or
      (uniq[0] = 0) or (rulset[r].npic = 1);
```

```
{count the number of instances covered by rule r}
with rulset[r] do
begin
    noic := 0;
    for i := 1 to ninst do
        if (ts[i,0] = abbr[0]) and
                                checkprem(i,abbr) then
                                 noic := noic + 1;
                    end;
          end;
          r := r + 1;
     end;
end;

{-------------------------------------------------------------------------}

procedure findrule;
{induces all rules and stores them in rulset}

var i  : twobyte;
    cc : byte;
    t  : zott;

begin
    nrul := 0;
    for cc := 1 to nclass do
    begin
        {initialize pt1 to represent the full training set}
        for i := 0 to ninst do
            pt1[i] := i;
    pt1[0] := ninst;

    {check if all instances in pt1 are of the same class}
    checlass(cc,pt1,t);
```

```
case t of
    0 : begin
            writeln(f,'The training set is empty');
            exitprog(1);
        end;

    1 : begin
            writeln(f,'All instances are of class ',cc:1);
            exitprog(1);
        end;

2 : writeln(f,'There are no instances of class ',cc:1);

3 : while t = 3 do
    begin
        {induce a set of rules for class cc and add the}
        {most general one to rulset}
        for i := 0 to ninst do pt2[i] := pt1[i];
        ftrset(cc);
        bestcover;

        {check consistency with other rulse in rulset}
        consistency(cc);

        {remove from pt1 all instances covered by this rule}
        {and check the classes of the remaining instances.}
        modmts(pt1,bestrule.abbr);
        checlass(cc,pt1,t);
    end;
  end;
end;

    {specialize any contradictory rules}
    specialize;
end;
```

```
{=============================================================================}

procedure wrabbr;
{writes rules to file}

var r, p, a, val : byte;

begin
  for r := 1 to nrul do
    if not(rulset[r].toogeneral) then
    {toogeneral is true if r does not cover any instances uniquely,}
    {or if there are clashing instances in the training set, i.e.}
    {the attributes are inadequate}
    begin
      write(f,'if    ');

      p := 0;
      for a := 1 to natt do
        if rulset[r].abbr[a] <> 0 then
        begin
        {write clause}
        write(f,attset[a].name:6,' = ');
        val := attset[a].pdv[rulset[r].abbr[a]];
        write(f,val:1);
        p := p + 1;
        if p < rulset[r].lprem then
        begin
          writeln(f,'and':6);
          write(f,'      ');
        end
        else writeln(f);
  end;

  {write class and number of instances covered}
  write(f,'then  ',attset[0].name:6,' = ');
  val := attset[0].pdv[rulset[r].abbr[0]];
  writeln(f,val:1,'(':10,rulset[r].noic:1,')');
```

```
                writeln(f);
        end;
end;

{=============================================================================}

begin {prism}
    initdata;
    findrule;
    wrabbr;
    close(f);
end.

{=============================================================================}
```

Appendix B

Inputs

The following examples of a static data base and training set provide information about the contact lens fitting problem detailed in Chapter 8.

The static data base:

```
5
Class 3 1 2 3
age 3 1 2 3
specRx 3 1 2 3
astig 2 1 2
tears 2 1 2
tbu 3 1 2 3
```

The training set:

```
21
1 1 2 1 3 3
1 1 2 2 3 1
1 2 2 2 2 2
1 3 1 2 3 3
2 1 1 2 2 3
2 1 2 1 1 3
2 1 2 2 2 3
2 2 1 1 2 3
2 2 1 1 3 3
2 2 1 2 1 3
2 2 1 2 3 2
2 2 2 2 2 3
2 2 2 2 3 3
2 3 1 1 3 3
3 1 1 2 3 3
3 1 2 2 1 3
3 2 1 1 1 3
3 2 1 2 2 3
3 2 1 2 3 2
3 3 2 1 1 3
3 3 2 1 3 3
```

Appendix C

The results

The following rules are those induced by PRISM from the training set given in Appendix B. The attributes and their values are described in Chapter 9.

```
if    age    = 1   and
      specRx = 1   and
      tears  = 2   and
      tbu    = 3
then  Class  = 1          (1)

if    specRx = 2   and
      astig  = 1   and
      tears  = 2   and
      tbu    = 3
then  Class  = 2          (2)

if    age    = 1   and
      specRx = 2   and
      astig  = 2   and
      tears  = 2   and
      tbu    = 2
then  Class  = 2          (1)
```

```
if    tears  = 1
then  Class  = 3          (8)

if    tbu    = 1
then  Class  = 3          (5)

if    specRx = 3
then  Class  = 3          (4)

if    age    = 2   and
      astig  = 2
then  Class  = 3          (4)

if    astig  = 1   and
      tbu    = 2
then  Class  = 3          (3)

if    age    = 3   and
      specRx = 1
then  Class  = 3          (2)
```