# 11
# Inducing Modular Rules for Classification

Generating classification rules via the intermediate form of a decision tree is a widely used technique, which formed the main topic of the first part of this book. However, as pointed out in Chapter 9, like many other methods it suffers from the problem of overfitting to the training data. We begin this chapter by describing the 'rule post-pruning' method, which is an alternative to the post-pruning method discussed in Chapter 9. This leads on to the important topic of *conflict resolution*.

We go on to suggest that the decision tree representation is itself a major cause of overfitting and then look at an algorithm which generates rules directly without using the intermediate representation of a decision tree.

## 11.1 Rule Post-pruning

The Rule Post-pruning method begins by converting a decision tree to an equivalent set of rules and then examines the rules with the aim of simplifying them without any loss of (and preferably with a gain in) predictive accuracy.

Figure 11.1 shows the decision tree for the *degrees* dataset given in Chapter 4. It consists of five branches, each ending with a leaf node labelled with one of the valid classifications, i.e. FIRST or SECOND.

Each branch of the tree corresponds to a classification rule and so the rules equivalent to the decision tree can be extracted from it branch by branch. The order in which the branches are taken is arbitrary as for any unseen instance



M. Bramer, *Principles of Data Mining*, Undergraduate Topics
in Computer Science, DOI 10.1007/978-1-4471-7307-6_11

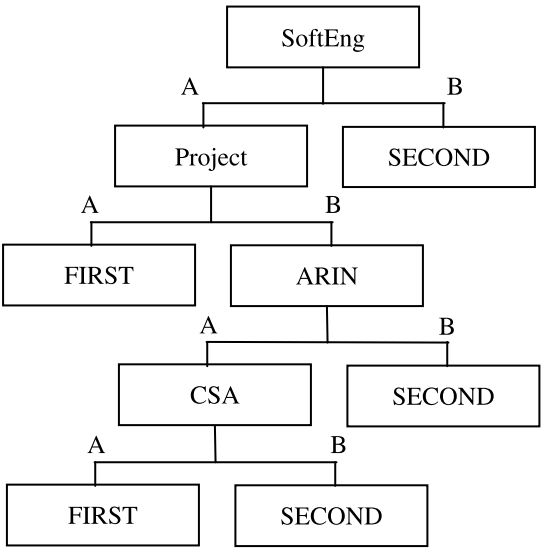


**Figure 11.1** Decision Tree for the *degrees* Dataset

only one rule (at most) can ever apply. The five rules corresponding to Figure 11.1 are as follows (in arbitrary order):

IF SoftEng = A AND Project = B AND
ARIN = A AND CSA = A THEN Class = FIRST
IF SoftEng = A AND Project = A THEN Class = FIRST
IF SoftEng = A AND Project = B AND ARIN = A AND
CSA = B THEN Class = SECOND
IF SoftEng = A AND Project = B AND ARIN = B THEN
Class = SECOND
IF SoftEng = B THEN Class = SECOND

We now examine each of the rules in turn to consider whether removing each of its terms increases or reduces its predictive accuracy. Thus for the first rule given above we consider the four terms 'SoftEng = A', 'Project = B', 'ARIN = A' and 'CSA = A'. We need some way of estimating whether removing each of these terms singly would increase or decrease the accuracy of the resulting rule set. Assuming we have such a method, we remove the term that gives the largest increase in predictive accuracy, say 'Project = B'. We then consider the removal of each of the other three terms. The processing of a rule ends when removing any of the terms would reduce (or leave unchanged) the predictive accuracy. We then go on to the next rule.

This description relies on there being some means of estimating the effect on the predictive accuracy of a ruleset of removing a single term from one of the rules. We may be able to use a probability-based formula to do this or we can simply use the original and revised rulesets to classify the instances in an unseen *pruning set* and compare the results. (Note that it would be methodologically unsound to improve the ruleset using a test set and then examine its performance on the same instances. For this method there needs to be three sets: training, pruning and test.)

## 11.2 Conflict Resolution

A second important issue raised by the use of rule post-pruning is of much wider applicability. Once even one term has been removed from a rule the property that for any unseen instance only one rule (at most) can ever apply is no longer valid.

The method of post-pruning described in Chapter 9, i.e. working bottom-up, repeatedly replacing a subtree by a single node has the very desirable property that the resulting branches will still fit together in a tree structure. For example the method might (probably unwisely) lead to the replacement of the test on the value of ARIN in Figure 11.1 and the subtree that hangs from it by a single node labelled SECOND. The result will still be a tree, as shown in Figure 11.2.

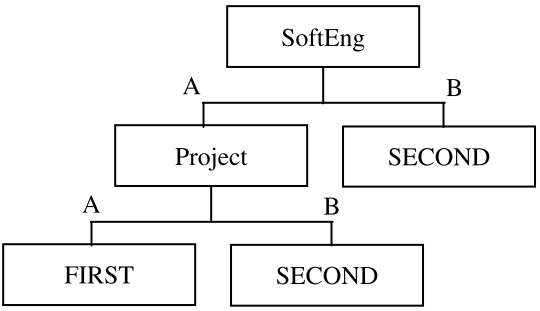


**Figure 11.2** Decision Tree for the *degrees* Dataset (revised)

Instead of this, suppose that, as part of a process such as rule post-pruning, we wish to remove the link corresponding to 'SoftEng = A' near the top of the tree, giving Figure 11.3.

If we do so, we will no longer have a tree — just two disconnected trees. It is unclear whether and how these can be used. The five rules listed in Section

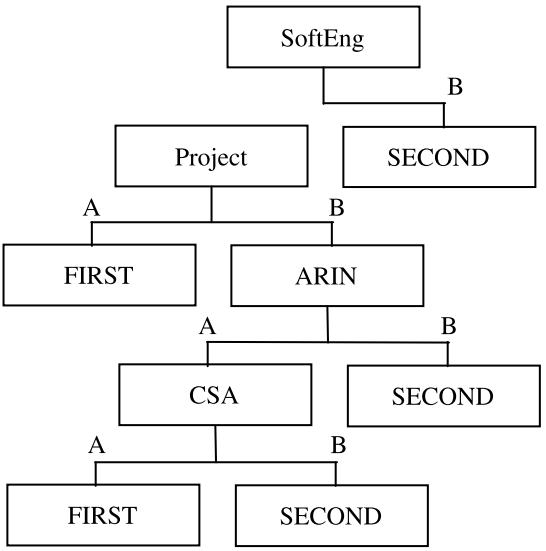


**Figure 11.3** Decision Tree for the *degrees* Dataset (revised – version 2)

11.1 have now become the following (the first four rules have changed).

IF Project = B AND ARIN = A AND CSA = A THEN Class = FIRST
IF Project = A THEN Class = FIRST
IF Project = B AND ARIN = A AND CSA = B
THEN Class = SECOND
IF Project = B AND ARIN = B THEN Class = SECOND
IF SoftEng = B THEN Class = SECOND

We will say that a rule *fires* if its condition part is satisfied for a given instance. If a set of rules fits into a tree structure there is only one rule that can fire for any instance. In the general case of a set of rules that do not fit into a tree structure, it is entirely possible for several rules to fire for a given test instance, and for those rules to give contradictory classifications.

Suppose that for the *degrees* application we have an unseen instance for which the values of SoftEng, Project, ARIN and CSA are 'B', 'B', 'A' and 'A', respectively. Both the first and the last rules will fire. The first rule concludes 'Class = FIRST'; the last rule concludes 'Class = SECOND'. Which one should we take?

The problem can be illustrated outside the context of the *degrees* dataset by considering just two rules from some imaginary ruleset:

IF $x = 4$ THEN Class = $a$

IF $y = 2$ THEN Class $= b$

What should the classification be for an instance with $x = 4$ and $y = 2$? One rule gives class $a$, the other class $b$.

We can easily extend the example with other rules such as

IF $w = 9$ and $k = 5$ THEN Class $= b$
IF $x = 4$ THEN Class $= a$
IF $y = 2$ THEN Class $= b$
IF $z = 6$ and $m = 47$ THEN Class $= b$

What should the classification be for an instance with $w = 9$, $k = 5$, $x = 4$, $y = 2$, $z = 6$ and $m = 47$? One rule gives class $a$, the other three rules give class $b$.

We need a method of choosing just one classification to give to the unseen instance. This method is known as a *conflict resolution strategy*. There are various strategies we can use, including:

- 'majority voting' (e.g. there are three rules predicting class $b$ and only one predicting class $a$, so choose class $b$)
- giving priority to certain types of rule or classification (e.g. rules with a small number of terms or predicting a rare classification might have a higher weighting than other rules in the voting)
- using a measure of the 'interestingness' of each rule (of the kind that will be discussed in Chapter 16), give priority to the most interesting rule.

It is possible to construct quite elaborate conflict resolution strategies but most of them have the same drawback: they require the condition part of *all* the rules to be tested for each unseen instance, so that all the rules that fire are known before the strategy is applied. By contrast, we need only work through the rules generated from a decision tree until the first one fires (as we know no others can).

A very basic but widely used conflict resolution strategy is to work through the rules in order and to take the first one that fires. This can reduce the amount of processing required considerably, but makes the order in which the rules are generated very important.

Whilst it is possible using a conflict resolution strategy to post-prune a decision tree to give a set of rules that do not fit together in a tree structure, it seems an unnecessarily indirect way of generating a set of rules. In addition if we wish to use the 'take the first rule that fires' conflict resolution strategy, the order in which the rules are extracted from the tree is likely to be of crucial importance, whereas it ought to be arbitrary.

In Section 11.4 we will describe an algorithm that dispenses with tree generation altogether and produces rules that are 'free standing', i.e. do not fit together into a tree structure, directly. We will call these *modular rules*.

## 11.3 Problems with Decision Trees

Although very widely used, the decision tree representation has a serious potential drawback: the rules derived from the tree may be much more numerous than necessary and may contain many redundant terms.

In a PhD project at the Open University, supervised by the present author, Cendrowska [1], [2] criticised the principle of generating decision trees which can then be converted to decision rules, compared with the alternative of generating decision rules directly from the training set. She comments as follows [the original notation has been changed to be consistent with that used in this book]:

"[The] decision tree representation of rules has a number of disadvantages. . . . [Most] importantly, there are rules that cannot easily be represented by trees.

Consider, for example, the following rule set:

Rule 1: IF $a = 1$ AND $b = 1$ THEN Class $= 1$
Rule 2: IF $c = 1$ AND $d = 1$ THEN Class $= 1$

Suppose that Rules 1 and 2 cover all instances of Class 1 and all other instances are of Class 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 is shown [in Figure 11.4].

The paths relating to Class 1 can be listed as follows:

IF $a = 1$ AND $b = 1$ THEN Class $= 1$
IF $a = 1$ AND $b = 2$ AND $c = 1$ AND $d = 1$ THEN Class $= 1$
IF $a = 1$ AND $b = 3$ AND $c = 1$ AND $d = 1$ THEN Class $= 1$
IF $a = 2$ AND $c = 1$ AND $d = 1$ THEN Class $= 1$

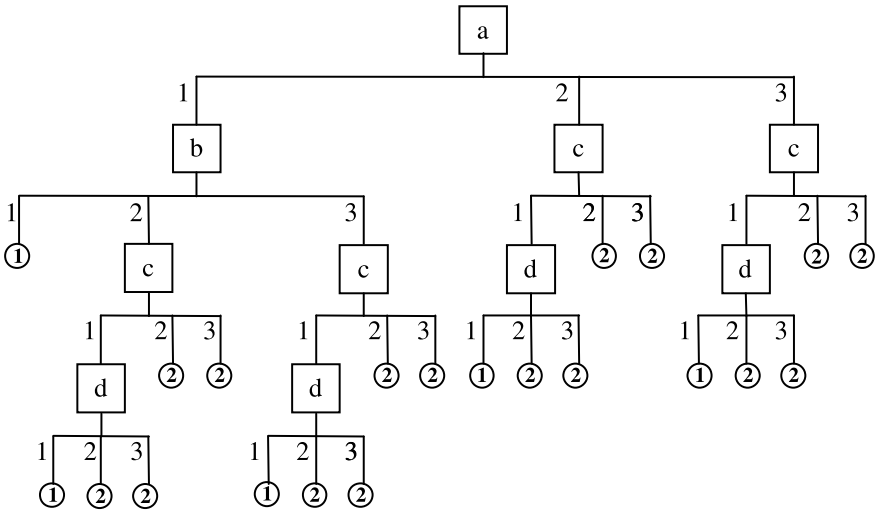


**Figure 11.4** Simplest Decision Tree Representation of Rules 1 and 2

IF $a = 3$ AND $c = 1$ AND $d = 1$ THEN Class = 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."

The phenomenon of unnecessarily large and confusing decision trees described by Cendrowska is far from being merely a rare hypothetical possibility. It will occur whenever there are two (underlying) rules with no attribute in common, a situation that is likely to occur frequently in practice.

All the rules corresponding to the branches of a decision tree must begin in the same way, i.e. with a test on the value of the attribute selected at the top level. Leaving aside issues of overfitting, this effect will inevitably lead to the introduction of terms in rules (branches) which are unnecessary except for the sole purpose of enabling a tree structure to be constructed.

Issues of the size and compactness of a rule set may not seem important when the training sets are small, but may become very important as they scale up to many thousands or millions of instances, especially if the number of attributes is also large.

Although in this book we have generally ignored issues of the practicality of and/or cost associated with finding the values of attributes, considerable practical problems can arise when the values of some attributes are unknown for an instance that needs to be classified or can only be obtained by means of

tests that carry an unusually high cost or risk to health. For many real-world applications a method of classifying unseen instances that avoided making unnecessary tests would be highly desirable.

## 11.4 The Prism Algorithm

The Prism algorithm was introduced by Cendrowska [1], [2]. The aim is to induce modular classification rules directly from the training set. The algorithm assumes that all the attributes are categorical. When there are continuous attributes they can first be converted to categorical ones (as described in Chapter 8). Alternatively the algorithm can be extended to deal with continuous attributes in much the same way as was described for TDIDT in Section 8.3.

Prism uses the 'take the first rule that fires' conflict resolution strategy when the resulting rules are applied to unseen data, so it is important that as far as possible the most important rules are generated first.

The algorithm generates the rules concluding each of the possible classes in turn. Each rule is generated term by term, with each term of the form 'attribute = value'. The attribute/value pair added at each step is chosen to maximise the probability of the target 'outcome class'.

In its basic form, the Prism algorithm is shown in Figure 11.5. Note that the training set is restored to its original state for each new class.

For each classification (class = $i$) in turn and starting with the complete training set each time:

1. Calculate the probability that class = $i$ for each attribute/value pair.
2. Select the pair with the largest probability and create a subset of the training set comprising all the instances with the selected attribute/value combination (for all classifications).
3. Repeat 1 and 2 for this subset until a subset is reached that contains only instances of class $i$. The induced rule is then the conjunction of all the attribute/value pairs selected.
4. Remove all instances covered by this rule from the training set.

*Repeat 1–4 until all instances of class $i$ have been removed*

**Figure 11.5** The Basic Prism Algorithm

We will illustrate the algorithm by generating rules for the *lens24* dataset (classification 1 only). The algorithm generates two classification rules for that class.

The initial training set for *lens24* comprises 24 instances, shown in Figure 11.6.

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 3 |
| 1 | 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 1 | 3 |
| 1 | 1 | 2 | 2 | 1 |
| 1 | 2 | 1 | 1 | 3 |
| 1 | 2 | 1 | 2 | 2 |
| 1 | 2 | 2 | 1 | 3 |
| 1 | 2 | 2 | 2 | 1 |
| 2 | 1 | 1 | 1 | 3 |
| 2 | 1 | 1 | 2 | 2 |
| 2 | 1 | 2 | 1 | 3 |
| 2 | 1 | 2 | 2 | 1 |
| 2 | 2 | 1 | 1 | 3 |
| 2 | 2 | 1 | 2 | 2 |
| 2 | 2 | 2 | 1 | 3 |
| 2 | 2 | 2 | 2 | 3 |
| 3 | 1 | 1 | 1 | 3 |
| 3 | 1 | 1 | 2 | 3 |
| 3 | 1 | 2 | 1 | 3 |
| 3 | 1 | 2 | 2 | 1 |
| 3 | 2 | 1 | 1 | 3 |
| 3 | 2 | 1 | 2 | 2 |
| 3 | 2 | 2 | 1 | 3 |
| 3 | 2 | 2 | 2 | 3 |

**Figure 11.6** The *lens24* Training Set

First Rule

Figure 11.7 shows the probability of class = 1 occurring for each attribute/value pair over the whole training set (24 instances).

The maximum probability is when *astig* = 2 or *tears* = 2.

Choose *astig* = 2 arbitrarily.

Incomplete rule induced so far:

IF astig = 2 THEN class = 1

| Attribute/value pair | Frequency for class = 1 | Total frequency (out of 24 instances) | Probability |
|---|---|---|---|
| age = 1 | 2 | 8 | 0.25 |
| age = 2 | 1 | 8 | 0.125 |
| age = 3 | 1 | 8 | 0.125 |
| specRx = 1 | 3 | 12 | 0.25 |
| specRx = 2 | 1 | 12 | 0.083 |
| astig = 1 | 0 | 12 | 0 |
| astig = 2 | 4 | 12 | 0.33 |
| tears = 1 | 0 | 12 | 0 |
| tears = 2 | 4 | 12 | 0.33 |

**Figure 11.7** First Rule: Probability of Attribute/value Pairs (Version 1)

The subset of the training set covered by this incomplete rule is given in Figure 11.8.

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 3 |
| 1 | 1 | 2 | 2 | 1 |
| 1 | 2 | 2 | 1 | 3 |
| 1 | 2 | 2 | 2 | 1 |
| 2 | 1 | 2 | 1 | 3 |
| 2 | 1 | 2 | 2 | 1 |
| 2 | 2 | 2 | 1 | 3 |
| 2 | 2 | 2 | 2 | 3 |
| 3 | 1 | 2 | 1 | 3 |
| 3 | 1 | 2 | 2 | 1 |
| 3 | 2 | 2 | 1 | 3 |
| 3 | 2 | 2 | 2 | 3 |

**Figure 11.8** First Rule: Subset of Training Set Covered by Incomplete Rule (Version 1)

Figure 11.9 shows the probability of each attribute/value pair (not involving attribute *astig*) occurring for this subset.

The maximum probability is when *tears* = 2.

Incomplete rule induced so far:

| Attribute/value pair | Frequency for class = 1 | Total frequency (out of 12 instances) | Probability |
|---|---|---|---|
| age = 1 | 2 | 4 | 0.5 |
| age = 2 | 1 | 4 | 0.25 |
| age = 3 | 1 | 4 | 0.25 |
| specRx = 1 | 3 | 6 | 0.5 |
| specRx = 2 | 1 | 6 | 0.17 |
| tears = 1 | 0 | 6 | 0 |
| tears = 2 | 4 | 6 | 0.67 |

**Figure 11.9** First Rule: Probability of Attribute/value Pairs (Version 2)

IF astig = 2 and tears = 2 THEN class = 1

The subset of the training set covered by this rule is shown in Figure 11.10.

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 1 |
| 1 | 2 | 2 | 2 | 1 |
| 2 | 1 | 2 | 2 | 1 |
| 2 | 2 | 2 | 2 | 3 |
| 3 | 1 | 2 | 2 | 1 |
| 3 | 2 | 2 | 2 | 3 |

**Figure 11.10** First Rule: Subset of Training Set Covered by Incomplete Rule (Version 2)

Figure 11.11 shows the probability of each attribute/value pair (not involving attributes *astig* or *tears*) occurring for this subset.

The maximum probability is when $age = 1$ or $specRx = 1$.
Choose (arbitrarily) $age = 1$.

Incomplete rule induced so far:

IF astig = 2 and tears = 2 and age = 1 THEN class = 1

The subset of the training set covered by this rule is given in Figure 11.12.

This subset contains only instances of class 1.

The final induced rule is therefore

IF astig = 2 and tears = 2 and age = 1 THEN class = 1

| Attribute/value pair | Frequency for class = 1 | Total frequency (out of 6 instances) | Probability |
|---|---|---|---|
| age = 1 | 2 | 2 | 1.0 |
| age = 2 | 1 | 2 | 0.5 |
| age = 3 | 1 | 2 | 0.5 |
| specRx = 1 | 3 | 3 | 1.0 |
| specRx = 2 | 1 | 3 | 0.33 |

**Figure 11.11** First Rule: Probability of Attribute/value Pairs (Version 3)

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 1 |
| 1 | 2 | 2 | 2 | 1 |

**Figure 11.12** First Rule: Subset of Training Set Covered by Incomplete Rule (Version 3)

Second Rule

Removing the two instances covered by the first rule from the training set gives a new training set with 22 instances. This is shown in Figure 11.13.

The table of frequencies is now as given in Figure 11.14 for attribute/value pairs corresponding to *class* = 1.

The maximum probability is achieved by *astig* = 2 and *tears* = 2. Choose *astig* = 2 arbitrarily.

Incomplete rule induced so far:

IF astig=2 THEN class = 1

The subset of the training set covered by this rule is shown in Figure 11.15.

This gives the frequency table shown in Figure 11.16.

The maximum probability is achieved by *tears* = 2.

Incomplete rule induced so far:

IF astig = 2 and tears = 2 then class = 1

The subset of the training set covered by this rule is shown in Figure 11.17.

This gives the frequency table shown in Figure 11.18.

The maximum probability is for *specRx* = 1.

Incomplete rule induced so far:

IF astig = 2 and tears = 2 and specRx = 1 THEN class = 1

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 3 |
| 1 | 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 1 | 3 |
| 1 | 2 | 1 | 1 | 3 |
| 1 | 2 | 1 | 2 | 2 |
| 1 | 2 | 2 | 1 | 3 |
| 2 | 1 | 1 | 1 | 3 |
| 2 | 1 | 1 | 2 | 2 |
| 2 | 1 | 2 | 1 | 3 |
| 2 | 1 | 2 | 2 | 1 |
| 2 | 2 | 1 | 1 | 3 |
| 2 | 2 | 1 | 2 | 2 |
| 2 | 2 | 2 | 1 | 3 |
| 2 | 2 | 2 | 2 | 3 |
| 3 | 1 | 1 | 1 | 3 |
| 3 | 1 | 1 | 2 | 3 |
| 3 | 1 | 2 | 1 | 3 |
| 3 | 1 | 2 | 2 | 1 |
| 3 | 2 | 1 | 1 | 3 |
| 3 | 2 | 1 | 2 | 2 |
| 3 | 2 | 2 | 1 | 3 |
| 3 | 2 | 2 | 2 | 3 |

**Figure 11.13** The *lens24* Training Set (Reduced)

| Attribute/value pair | Frequency for class = 1 | Total frequency (out of 22 instances) | Probability |
|---|---|---|---|
| age = 1 | 0 | 6 | 0 |
| age = 2 | 1 | 8 | 0.125 |
| age = 3 | 1 | 8 | 0.125 |
| specRx = 1 | 2 | 11 | 0.18 |
| specRx = 2 | 0 | 11 | 0 |
| astig = 1 | 0 | 12 | 0 |
| astig = 2 | 2 | 10 | 0.2 |
| tears = 1 | 0 | 12 | 0 |
| tears = 2 | 2 | 10 | 0.2 |

**Figure 11.14** Second Rule: Probability of Attribute/value Pairs (Version 1)

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 3 |
| 1 | 2 | 2 | 1 | 3 |
| 2 | 1 | 2 | 1 | 3 |
| 2 | 1 | 2 | 2 | 1 |
| 2 | 2 | 2 | 1 | 3 |
| 2 | 2 | 2 | 2 | 3 |
| 3 | 1 | 2 | 1 | 3 |
| 3 | 1 | 2 | 2 | 1 |
| 3 | 2 | 2 | 1 | 3 |
| 3 | 2 | 2 | 2 | 3 |

**Figure 11.15** Second Rule: Subset of Training Set Covered by Incomplete Rule (Version 1)

| Attribute/value pair | Frequency for class = 1 | Total frequency (out of 10 instances) | Probability |
|---|---|---|---|
| age = 1 | 0 | 2 | 0 |
| age = 2 | 1 | 4 | 0.25 |
| age = 3 | 1 | 4 | 0.25 |
| specRx = 1 | 0 | 5 | 0 |
| specRx = 2 | 2 | 5 | 0.4 |
| tears = 1 | 0 | 6 | 0 |
| tears = 2 | 2 | 4 | 0.5 |

**Figure 11.16** Second Rule: Probability of Attribute/value Pairs (Version 2)

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 2 | 1 | 2 | 2 | 1 |
| 2 | 2 | 2 | 2 | 3 |
| 3 | 1 | 2 | 2 | 1 |
| 3 | 2 | 2 | 2 | 3 |

**Figure 11.17** Second Rule: Subset of Training Set Covered by Incomplete Rule (Version 2)

| Attribute/value pair | Frequency for class = 1 | Total Frequency (out of 4 instances) | Probability |
|---|---|---|---|
| age = 1 | 0 | 0 | – |
| age = 2 | 1 | 2 | 0.5 |
| age = 3 | 1 | 2 | 0.5 |
| specRx = 1 | 2 | 2 | 1.0 |
| specRx = 2 | 0 | 2 | 0 |

**Figure 11.18** Second Rule: Probability of Attribute/value Pairs (Version 3)

| age | specRx | astig | tears | class |
|---|---|---|---|---|
| 2 | 1 | 2 | 2 | 1 |
| 3 | 1 | 2 | 2 | 1 |

**Figure 11.19** Second Rule: Subset of Training Set Covered by Incomplete Rule (Version 3)

The subset of the training set covered by this rule is shown in Figure 11.19. This subset contains only instances of class 1. So the final induced rule is:

IF astig = 2 and tears = 2 and specRx = 1 THEN class = 1

Removing the two instances covered by this rule from the current version of the training set (which has 22 instances) gives a training set of 20 instances from which all instances of class 1 have now been removed. So the Prism algorithm terminates (for classification 1).

The final pair of rules induced by Prism for class 1 are:

IF astig = 2 and tears = 2 and age = 1 THEN class = 1
IF astig = 2 and tears = 2 and specRx = 1 THEN class = 1

The algorithm will now go on to generate rules for the remaining classifications. It produces 3 rules for class 2 and 4 for class 3. Note that the training set is restored to its original state for each new class.

### 11.4.1 Changes to the Basic Prism Algorithm

1. *Tie-breaking*

   The basic algorithm can be improved slightly by choosing between attribute/value pairs which have equal probability not arbitrarily as above but by taking the one with the highest total frequency.

2. *Clashes in the Training Data*

   The original version of Prism does not include any method of dealing with clashes in the training set encountered during rule generation.

However, the basic algorithm can easily be extended to deal with clashes as follows.

Step 3 of the algorithm states:

> Repeat 1 and 2 for this subset until a subset is reached that contains only instances of class $i$.

To this needs to be added 'or a subset is reached which contains instances of more than one class, although values of all the attributes have already been used in creating the subset'.

The simple approach of assigning all instances in the subset to the majority class does not fit directly into the Prism framework. A number of approaches to doing so have been investigated, and the most effective would appear to be as follows.

> If a clash occurs while generating the rules for class $i$:
>
> 1. Determine the majority class for the subset of instances in the clash set.
> 2. If this majority class is class $i$, then complete the induced rule by assigning all the instances in the clash set to class $i$. If not, discard the rule.

### 11.4.2 Comparing Prism with TDIDT

Both the additional features described in Section 11.4.1 are included in a re-implementation of Prism by the present author [3].

The same paper describes a series of experiments to compare the performance of Prism with that of TDIDT on a number of datasets. The author concludes "The experiments presented here suggest that the Prism algorithm for generating modular rules gives classification rules which are at least as good as those obtained from the widely used TDIDT algorithm. There are generally fewer rules with fewer terms per rule, which is likely to aid their comprehensibility to domain experts and users. This result would seem to apply even more strongly when there is noise in the training set. As far as classification accuracy on unseen test data is concerned, there appears to be little to choose between the two algorithms for noise-free datasets, including ones with a significant proportion of clash instances in the training set. The main difference

is that Prism generally has a preference for leaving a test instance as 'unclassified' rather than giving it a wrong classification. In some domains this may be an important feature. When it is not, a simple strategy such as assigning unclassified instances to the majority class would seem to suffice. When noise is present, Prism would seem to give consistently better classification accuracy than TDIDT, even when there is a high level of noise in the training set.... The reasons why Prism should be more tolerant to noise than TDIDT are not entirely clear, but may be related to the presence of fewer terms per rule in most cases. The computational effort involved in generating rules using Prism ... is greater than for TDIDT. However, Prism would seem to have considerable potential for efficiency improvement by parallelisation."

These very positive conclusions are of course based on only a fairly limited number of experiments and need to be verified for a much wider range of datasets. In practice, despite the drawbacks of a decision tree representation and the obvious potential of Prism and other similar algorithms, TDIDT is far more frequently used to generate classification rules. The ready availability of C4.5 [4] and related systems is no doubt a significant factor in this.

In Chapter 16 we go on to look at the use of modular rules for predicting associations between attribute values rather than for classification.

## 11.5 Chapter Summary

This chapter begins by considering a method of post-pruning decision rules generated via a decision tree, which has the property that the pruned rules will not generally fit together to form a tree. Rules of this kind are known as *modular rules*. When using modular rules to classify unseen test data a *conflict resolution strategy* is needed and several possibilities for this are discussed. The use of a decision tree as an intermediate representation for rules is identified as a source of overfitting.

The Prism algorithm induces modular classification rules directly from a training set. Prism is described in detail, followed by a discussion of its performance as a classification algorithm relative to TDIDT.

## 11.6 Self-assessment Exercise for Chapter 11

What would be the first rule generated by Prism for the *degrees* dataset given in Chapter 4, Figure 4.3, for class 'FIRST'?

# References

[1] Cendrowska, J. (1987). PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, *27*, 349–370.

[2] Cendrowska, J. (1990). *Knowledge acquisition for expert systems: inducing modular rules from examples*. PhD Thesis, The Open University.

[3] Bramer, M. A. (2000). Automatic induction of classification rules from examples using N-prism. In *Research and development in intelligent systems XVI* (pp. 99–121). Berlin: Springer.

[4] Quinlan, J. R. (1993). *C4.5: programs for machine learning*. San Mateo: Morgan Kaufmann.