5

Classification: Alternative Techniques

The previous chapter described a simple, yet quite effective, classification technique known as decision tree induction. Issues such as model overfitting and classifier evaluation were also discussed in great detail. This chapter presents alternative techniques for building classification models—from simple techniques such as rule-based and nearest-neighbor classifiers to more advanced techniques such as support vector machines and ensemble methods. Other key issues such as the class imbalance and multiclass problems are also discussed at the end of the chapter.

5.1 Rule-Based Classifier

A rule-based classifier is a technique for classifying records using a collection of "if ...then..." rules. Table 5.1 shows an example of a model generated by a rule-based classifier for the vertebrate classification problem. The rules for the model are represented in a disjunctive normal form, $R = (r_1 \vee r_2 \vee \ldots r_k)$, where R is known as the **rule set** and r_i's are the classification rules or disjuncts.

Table 5.1. Example of a rule set for the vertebrate classification problem.

r_1:	(Gives Birth = no) $\wedge$ (Aerial Creature = yes) $\longrightarrow$ Birds
r_2:	(Gives Birth = no) $\wedge$ (Aquatic Creature = yes) $\longrightarrow$ Fishes
r_3:	(Gives Birth = yes) $\wedge$ (Body Temperature = warm-blooded) $\longrightarrow$ Mammals
r_4:	(Gives Birth = no) $\wedge$ (Aerial Creature = no) $\longrightarrow$ Reptiles
r_5:	(Aquatic Creature = semi) $\longrightarrow$ Amphibians

Each classification rule can be expressed in the following way:

$$r_i: \quad (Condition_i) \longrightarrow y_i. \tag{5.1}$$

The left-hand side of the rule is called the **rule antecedent** or **precondition**. It contains a conjunction of attribute tests:

$$Condition_i = (A_1 \; op \; v_1) \wedge (A_2 \; op \; v_2) \wedge \ldots (A_k \; op \; v_k), \tag{5.2}$$

where (A_j, v_j) is an attribute-value pair and *op* is a logical operator chosen from the set $\{=, \neq, <, >, \leq, \geq\}$. Each attribute test $(A_j \; op \; v_j)$ is known as a conjunct. The right-hand side of the rule is called the **rule consequent**, which contains the predicted class y_i.

A rule r covers a record x if the precondition of r matches the attributes of x. r is also said to be fired or triggered whenever it covers a given record. For an illustration, consider the rule r_1 given in Table 5.1 and the following attributes for two vertebrates: hawk and grizzly bear.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates
hawk	warm-blooded	feather	no	no	yes	yes	no
grizzly bear	warm-blooded	fur	yes	no	no	yes	yes

r_1 covers the first vertebrate because its precondition is satisfied by the hawk's attributes. The rule does not cover the second vertebrate because grizzly bears give birth to their young and cannot fly, thus violating the precondition of r_1.

The quality of a classification rule can be evaluated using measures such as coverage and accuracy. Given a data set D and a classification rule $r: A \longrightarrow y$, the coverage of the rule is defined as the fraction of records in D that trigger the rule r. On the other hand, its accuracy or confidence factor is defined as the fraction of records triggered by r whose class labels are equal to y. The formal definitions of these measures are

$$\begin{aligned} \text{Coverage}(r) &= \frac{|A|}{|D|} \\ \text{Accuracy}(r) &= \frac{|A \cap y|}{|A|}, \end{aligned} \tag{5.3}$$

where $|A|$ is the number of records that satisfy the rule antecedent, $|A \cap y|$ is the number of records that satisfy both the antecedent and consequent, and $|D|$ is the total number of records.

Table 5.2. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	Mammals
python	cold-blooded	scales	no	no	no	no	yes	Reptiles
salmon	cold-blooded	scales	no	yes	no	no	no	Fishes
whale	warm-blooded	hair	yes	yes	no	no	no	Mammals
frog	cold-blooded	none	no	semi	no	yes	yes	Amphibians
komodo dragon	cold-blooded	scales	no	no	no	yes	no	Reptiles
bat	warm-blooded	hair	yes	no	yes	yes	yes	Mammals
pigeon	warm-blooded	feathers	no	no	yes	yes	no	Birds
cat	warm-blooded	fur	yes	no	no	yes	no	Mammals
guppy	cold-blooded	scales	yes	yes	no	no	no	Fishes
alligator	cold-blooded	scales	no	semi	no	yes	no	Reptiles
penguin	warm-blooded	feathers	no	semi	no	yes	no	Birds
porcupine	warm-blooded	quills	yes	no	no	yes	yes	Mammals
eel	cold-blooded	scales	no	yes	no	no	no	Fishes
salamander	cold-blooded	none	no	semi	no	yes	yes	Amphibians

Example 5.1. Consider the data set shown in Table 5.2. The rule

$$(\texttt{Gives Birth} = \texttt{yes}) \wedge (\texttt{Body Temperature} = \texttt{warm-blooded}) \longrightarrow \texttt{Mammals}$$

has a coverage of 33% since five of the fifteen records support the rule antecedent. The rule accuracy is 100% because all five vertebrates covered by the rule are mammals. ■

5.1.1 How a Rule-Based Classifier Works

A rule-based classifier classifies a test record based on the rule triggered by the record. To illustrate how a rule-based classifier works, consider the rule set shown in Table 5.1 and the following vertebrates:

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates
lemur	warm-blooded	fur	yes	no	no	yes	yes
turtle	cold-blooded	scales	no	semi	no	yes	no
dogfish shark	cold-blooded	scales	yes	yes	no	no	no

- The first vertebrate, which is a lemur, is warm-blooded and gives birth to its young. It triggers the rule r_3, and thus, is classified as a mammal.

- The second vertebrate, which is a turtle, triggers the rules r_4 and r_5. Since the classes predicted by the rules are contradictory (reptiles versus amphibians), their conflicting classes must be resolved.

- None of the rules are applicable to a dogfish shark. In this case, we need to ensure that the classifier can still make a reliable prediction even though a test record is not covered by any rule.

The previous example illustrates two important properties of the rule set generated by a rule-based classifier.

Mutually Exclusive Rules The rules in a rule set R are mutually exclusive if no two rules in R are triggered by the same record. This property ensures that every record is covered by at most one rule in R. An example of a mutually exclusive rule set is shown in Table 5.3.

Exhaustive Rules A rule set R has exhaustive coverage if there is a rule for each combination of attribute values. This property ensures that every record is covered by at least one rule in R. Assuming that `Body Temperature` and `Gives Birth` are binary variables, the rule set shown in Table 5.3 has exhaustive coverage.

Table 5.3. Example of a mutually exclusive and exhaustive rule set.

r_1: (Body Temperature = cold-blooded) $\longrightarrow$ Non-mammals
r_2: (Body Temperature = warm-blooded) $\wedge$ (Gives Birth = yes) $\longrightarrow$ Mammals
r_3: (Body Temperature = warm-blooded) $\wedge$ (Gives Birth = no) $\longrightarrow$ Non-mammals

Together, these properties ensure that every record is covered by exactly one rule. Unfortunately, many rule-based classifiers, including the one shown in Table 5.1, do not have such properties. If the rule set is not exhaustive, then a default rule, r_d : $() \longrightarrow y_d$, must be added to cover the remaining cases. A default rule has an empty antecedent and is triggered when all other rules have failed. y_d is known as the default class and is typically assigned to the majority class of training records not covered by the existing rules.

If the rule set is not mutually exclusive, then a record can be covered by several rules, some of which may predict conflicting classes. There are two ways to overcome this problem.

Ordered Rules In this approach, the rules in a rule set are ordered in decreasing order of their priority, which can be defined in many ways (e.g., based on accuracy, coverage, total description length, or the order in which the rules are generated). An ordered rule set is also known as a **decision list**. When a test record is presented, it is classified by the highest-ranked rule that covers the record. This avoids the problem of having conflicting classes predicted by multiple classification rules.

Unordered Rules This approach allows a test record to trigger multiple classification rules and considers the consequent of each rule as a vote for a particular class. The votes are then tallied to determine the class label of the test record. The record is usually assigned to the class that receives the highest number of votes. In some cases, the vote may be weighted by the rule's accuracy. Using unordered rules to build a rule-based classifier has both advantages and disadvantages. Unordered rules are less susceptible to errors caused by the wrong rule being selected to classify a test record (unlike classifiers based on ordered rules, which are sensitive to the choice of rule-ordering criteria). Model building is also less expensive because the rules do not have to be kept in sorted order. Nevertheless, classifying a test record can be quite an expensive task because the attributes of the test record must be compared against the precondition of every rule in the rule set.

In the remainder of this section, we will focus on rule-based classifiers that use ordered rules.

5.1.2 Rule-Ordering Schemes

Rule ordering can be implemented on a rule-by-rule basis or on a class-by-class basis. The difference between these schemes is illustrated in Figure 5.1.

Rule-Based Ordering Scheme This approach orders the individual rules by some rule quality measure. This ordering scheme ensures that every test record is classified by the "best" rule covering it. A potential drawback of this scheme is that lower-ranked rules are much harder to interpret because they assume the negation of the rules preceding them. For example, the fourth rule shown in Figure 5.1 for rule-based ordering,

$$\text{Aquatic Creature} = \text{semi} \longrightarrow \text{Amphibians},$$

has the following interpretation: If the vertebrate does not have any feathers or cannot fly, and is cold-blooded and semi-aquatic, then it is an amphibian.

Rule-Based Ordering

(Skin Cover=feathers, Aerial Creature=yes) ==> Birds

(Body temperature=warm-blooded, Gives Birth=yes) ==> Mammals

(Body temperature=warm-blooded, Gives Birth=no) ==> Birds

(Aquatic Creature=semi)) ==> Amphibians

(Skin Cover=scales, Aquatic Creature=no) ==> Reptiles

(Skin Cover=scales, Aquatic Creature=yes) ==> Fishes

(Skin Cover=none) ==> Amphibians

Class-Based Ordering

(Skin Cover=feathers, Aerial Creature=yes) ==> Birds

(Body temperature=warm-blooded, Gives Birth=no) ==> Birds

(Body temperature=warm-blooded, Gives Birth=yes) ==> Mammals

(Aquatic Creature=semi)) ==> Amphibians

(Skin Cover=none) ==> Amphibians

(Skin Cover=scales, Aquatic Creature=no) ==> Reptiles

(Skin Cover=scales, Aquatic Creature=yes) ==> Fishes

Figure 5.1. Comparison between rule-based and class-based ordering schemes.

The additional conditions (that the vertebrate does not have any feathers or cannot fly, and is cold-blooded) are due to the fact that the vertebrate does not satisfy the first three rules. If the number of rules is large, interpreting the meaning of the rules residing near the bottom of the list can be a cumbersome task.

Class-Based Ordering Scheme In this approach, rules that belong to the same class appear together in the rule set R. The rules are then collectively sorted on the basis of their class information. The relative ordering among the rules from the same class is not important; as long as one of the rules fires, the class will be assigned to the test record. This makes rule interpretation slightly easier. However, it is possible for a high-quality rule to be overlooked in favor of an inferior rule that happens to predict the higher-ranked class.

Since most of the well-known rule-based classifiers (such as C4.5rules and RIPPER) employ the class-based ordering scheme, the discussion in the remainder of this section focuses mainly on this type of ordering scheme.

5.1.3 How to Build a Rule-Based Classifier

To build a rule-based classifier, we need to extract a set of rules that identifies key relationships between the attributes of a data set and the class label.

There are two broad classes of methods for extracting classification rules: (1) direct methods, which extract classification rules directly from data, and (2) indirect methods, which extract classification rules from other classification models, such as decision trees and neural networks.

Direct methods partition the attribute space into smaller subspaces so that all the records that belong to a subspace can be classified using a single classification rule. Indirect methods use the classification rules to provide a succinct description of more complex classification models. Detailed discussions of these methods are presented in Sections 5.1.4 and 5.1.5, respectively.

5.1.4 Direct Methods for Rule Extraction

The **sequential covering** algorithm is often used to extract rules directly from data. Rules are grown in a greedy fashion based on a certain evaluation measure. The algorithm extracts the rules one class at a time for data sets that contain more than two classes. For the vertebrate classification problem, the sequential covering algorithm may generate rules for classifying birds first, followed by rules for classifying mammals, amphibians, reptiles, and finally, fishes (see Figure 5.1). The criterion for deciding which class should be generated first depends on a number of factors, such as the class prevalence (i.e., fraction of training records that belong to a particular class) or the cost of misclassifying records from a given class.

A summary of the sequential covering algorithm is given in Algorithm 5.1. The algorithm starts with an empty decision list, R. The Learn-One-Rule function is then used to extract the best rule for class y that covers the current set of training records. During rule extraction, all training records for class y are considered to be positive examples, while those that belong to

Algorithm 5.1 Sequential covering algorithm.

```
1: Let $E$ be the training records and $A$ be the set of attribute-value pairs, $\{(A_j, v_j)\}$.
2: Let $Y_o$ be an ordered set of classes $\{y_1, y_2, \ldots, y_k\}$.
3: Let $R = \{\ \}$ be the initial rule list.
4: for each class $y \in Y_o - \{y_k\}$ do
5:    while stopping condition is not met do
6:       $r \leftarrow$ Learn-One-Rule ($E$, $A$, $y$).
7:       Remove training records from $E$ that are covered by $r$.
8:       Add $r$ to the bottom of the rule list: $R \longrightarrow R \vee r$.
9:    end while
10: end for
11: Insert the default rule, $\{\} \longrightarrow y_k$, to the bottom of the rule list $R$.
```

other classes are considered to be negative examples. A rule is desirable if it covers most of the positive examples and none (or very few) of the negative examples. Once such a rule is found, the training records covered by the rule are eliminated. The new rule is added to the bottom of the decision list *R*. This procedure is repeated until the stopping criterion is met. The algorithm then proceeds to generate rules for the next class.

Figure 5.2 demonstrates how the sequential covering algorithm works for a data set that contains a collection of positive and negative examples. The rule *R*1, whose coverage is shown in Figure 5.2(b), is extracted first because it covers the largest fraction of positive examples. All the training records covered by *R*1 are subsequently removed and the algorithm proceeds to look for the next best rule, which is *R*2.

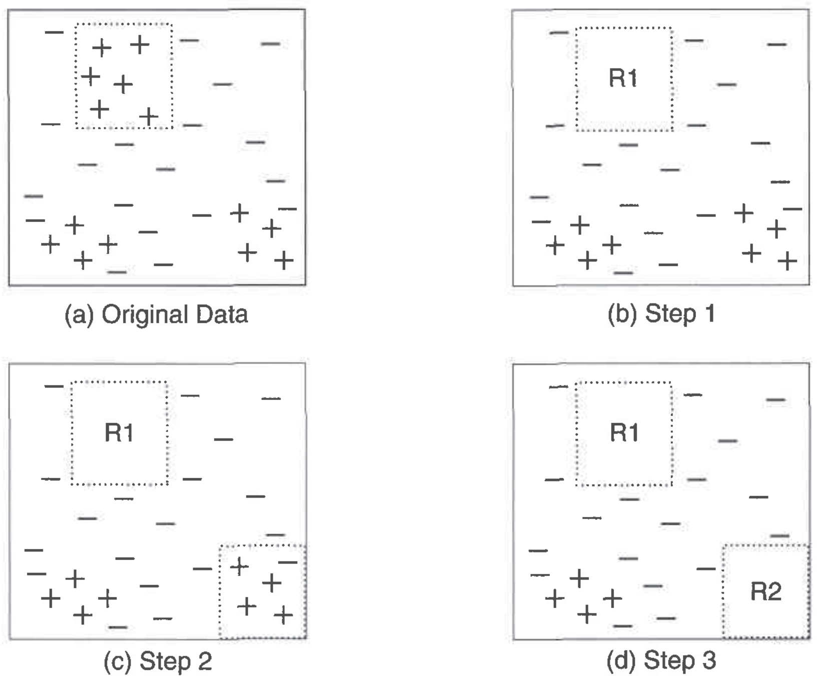


Figure 5.2. An example of the sequential covering algorithm.

Learn-One-Rule Function

The objective of the Learn-One-Rule function is to extract a classification rule that covers many of the positive examples and none (or very few) of the negative examples in the training set. However, finding an optimal rule is computationally expensive given the exponential size of the search space. The Learn-One-Rule function addresses the exponential search problem by growing the rules in a greedy fashion. It generates an initial rule r and keeps refining the rule until a certain stopping criterion is met. The rule is then pruned to improve its generalization error.

Rule-Growing Strategy There are two common strategies for growing a classification rule: general-to-specific or specific-to-general. Under the general-to-specific strategy, an initial rule $r : \{\} \longrightarrow y$ is created, where the left-hand side is an empty set and the right-hand side contains the target class. The rule has poor quality because it covers all the examples in the training set. New

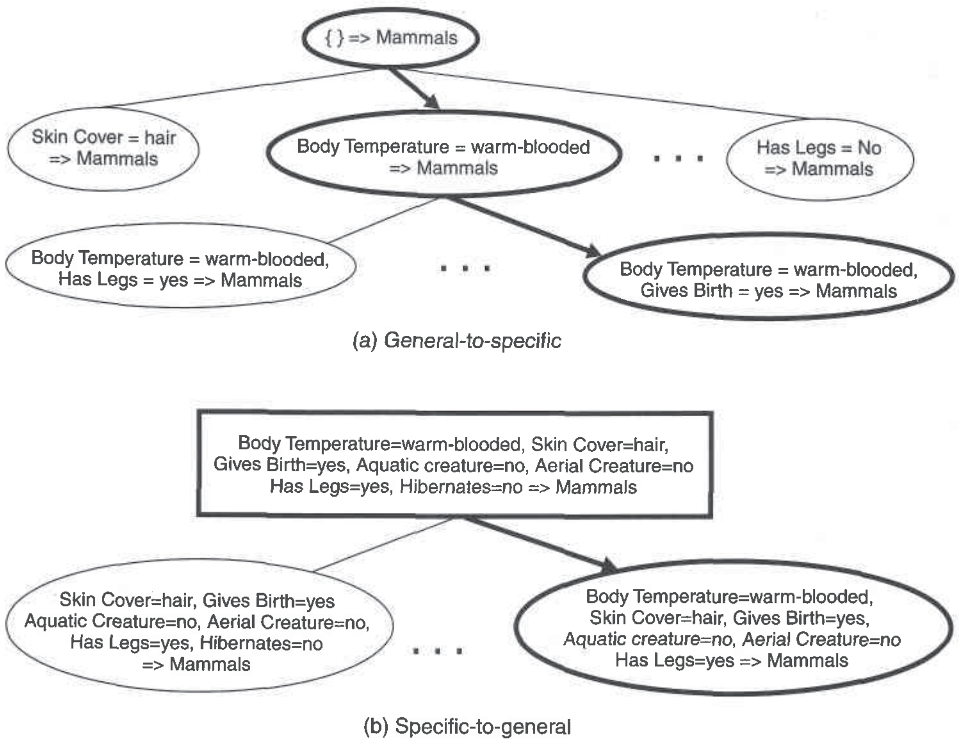

{ } => Mammals
Skin Cover = hair => Mammals
Body Temperature = warm-blooded => Mammals
. . .
Has Legs = No => Mammals
Body Temperature = warm-blooded, Has Legs = yes => Mammals
. . .
Body Temperature = warm-blooded, Gives Birth = yes => Mammals
(a) General-to-specific
Body Temperature=warm-blooded, Skin Cover=hair, Gives Birth=yes, Aquatic creature=no, Aerial Creature=no Has Legs=yes, Hibernates=no => Mammals
Skin Cover=hair, Gives Birth=yes Aquatic Creature=no, Aerial Creature=no, Has Legs=yes, Hibernates=no => Mammals
. . .
Body Temperature=warm-blooded, Skin Cover=hair, Gives Birth=yes, Aquatic creature=no, Aerial Creature=no Has Legs=yes => Mammals
(b) Specific-to-general

Figure 5.3. General-to-specific and specific-to-general rule-growing strategies.

conjuncts are subsequently added to improve the rule's quality. Figure 5.3(a) shows the general-to-specific rule-growing strategy for the vertebrate classification problem. The conjunct `Body Temperature=warm-blooded` is initially chosen to form the rule antecedent. The algorithm then explores all the possible candidates and greedily chooses the next conjunct, `Gives Birth=yes`, to be added into the rule antecedent. This process continues until the stopping criterion is met (e.g., when the added conjunct does not improve the quality of the rule).

For the specific-to-general strategy, one of the positive examples is randomly chosen as the initial seed for the rule-growing process. During the refinement step, the rule is generalized by removing one of its conjuncts so that it can cover more positive examples. Figure 5.3(b) shows the specific-to-general approach for the vertebrate classification problem. Suppose a positive example for mammals is chosen as the initial seed. The initial rule contains the same conjuncts as the attribute values of the seed. To improve its coverage, the rule is generalized by removing the conjunct `Hibernate=no`. The refinement step is repeated until the stopping criterion is met, e.g., when the rule starts covering negative examples.

The previous approaches may produce suboptimal rules because the rules are grown in a greedy fashion. To avoid this problem, a beam search may be used, where k of the best candidate rules are maintained by the algorithm. Each candidate rule is then grown separately by adding (or removing) a conjunct from its antecedent. The quality of the candidates are evaluated and the k best candidates are chosen for the next iteration.

Rule Evaluation An evaluation metric is needed to determine which conjunct should be added (or removed) during the rule-growing process. Accuracy is an obvious choice because it explicitly measures the fraction of training examples classified correctly by the rule. However, a potential limitation of accuracy is that it does not take into account the rule's coverage. For example, consider a training set that contains 60 positive examples and 100 negative examples. Suppose we are given the following two candidate rules:

Rule r_1: covers 50 positive examples and 5 negative examples,
Rule r_2: covers 2 positive examples and no negative examples.

The accuracies for r_1 and r_2 are 90.9% and 100%, respectively. However, r_1 is the better rule despite its lower accuracy. The high accuracy for r_2 is potentially spurious because the coverage of the rule is too low.

The following approaches can be used to handle this problem.

1. A statistical test can be used to prune rules that have poor coverage. For example, we may compute the following likelihood ratio statistic:

   $$R = 2\sum_{i=1}^{k} f_i \log(f_i/e_i),$$

   where k is the number of classes, f_i is the observed frequency of class i examples that are covered by the rule, and e_i is the expected frequency of a rule that makes random predictions. Note that R has a chi-square distribution with $k-1$ degrees of freedom. A large R value suggests that the number of correct predictions made by the rule is significantly larger than that expected by random guessing. For example, since r_1 covers 55 examples, the expected frequency for the positive class is $e_+ = 55 \times 60/160 = 20.625$, while the expected frequency for the negative class is $e_- = 55 \times 100/160 = 34.375$. Thus, the likelihood ratio for r_1 is

   $$R(r_1) = 2 \times [50 \times \log_2(50/20.625) + 5 \times \log_2(5/34.375)] = 99.9.$$

   Similarly, the expected frequencies for r_2 are $e_+ = 2 \times 60/160 = 0.75$ and $e_- = 2 \times 100/160 = 1.25$. The likelihood ratio statistic for r_2 is

   $$R(r_2) = 2 \times [2 \times \log_2(2/0.75) + 0 \times \log_2(0/1.25)] = 5.66.$$

   This statistic therefore suggests that r_1 is a better rule than r_2.

2. An evaluation metric that takes into account the rule coverage can be used. Consider the following evaluation metrics:

   $$\text{Laplace} = \frac{f_+ + 1}{n+k}, \tag{5.4}$$

   $$\text{m-estimate} = \frac{f_+ + kp_+}{n+k}, \tag{5.5}$$

   where n is the number of examples covered by the rule, f_+ is the number of positive examples covered by the rule, k is the total number of classes, and p_+ is the prior probability for the positive class. Note that the m-estimate is equivalent to the Laplace measure by choosing $p_+ = 1/k$. Depending on the rule coverage, these measures capture the trade-off between rule accuracy and the prior probability of the positive class. If the rule does not cover any training example, then the Laplace measure reduces to $1/k$, which is the prior probability of the positive class assuming a uniform class distribution. The m-estimate also reduces to the prior probability (p_+) when $n = 0$. However, if the rule coverage is large, then both measures asymptotically approach the rule accuracy, f_+/n. Going back to the previous example, the Laplace measure for r_1 is $51/57 = 89.47\%$, which is quite close to its accuracy. Conversely, the Laplace measure for r_2 (75%) is significantly lower than its accuracy because r_2 has a much lower coverage.

3. An evaluation metric that takes into account the support count of the rule can be used. One such metric is the **FOIL's information gain**. The support count of a rule corresponds to the number of positive examples covered by the rule. Suppose the rule $r: A \longrightarrow +$ covers p_0 positive examples and n_0 negative examples. After adding a new conjunct B, the extended rule $r': A \wedge B \longrightarrow +$ covers p_1 positive examples and n_1 negative examples. Given this information, the FOIL's information gain of the extended rule is defined as follows:

$$\text{FOIL's information gain} = p_1 \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right). \quad (5.6)$$

   Since the measure is proportional to p_1 and $p_1/(p_1 + n_1)$, it prefers rules that have high support count and accuracy. The FOIL's information gains for rules r_1 and r_2 given in the preceding example are 43.12 and 2, respectively. Therefore, r_1 is a better rule than r_2.

Rule Pruning The rules generated by the Learn-One-Rule function can be pruned to improve their generalization errors. To determine whether pruning is necessary, we may apply the methods described in Section 4.4 on page 172 to estimate the generalization error of a rule. For example, if the error on validation set decreases after pruning, we should keep the simplified rule. Another approach is to compare the pessimistic error of the rule before and after pruning (see Section 4.4.4 on page 179). The simplified rule is retained in place of the original rule if the pessimistic error improves after pruning.

Rationale for Sequential Covering

After a rule is extracted, the sequential covering algorithm must eliminate all the positive and negative examples covered by the rule. The rationale for doing this is given in the next example.

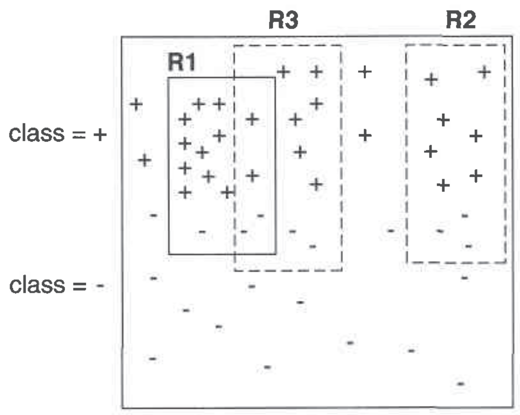


Figure 5.4. Elimination of training records by the sequential covering algorithm. $R1$, $R2$, and $R3$ represent regions covered by three different rules.

Figure 5.4 shows three possible rules, $R1$, $R2$, and $R3$, extracted from a data set that contains 29 positive examples and 21 negative examples. The accuracies of $R1$, $R2$, and $R3$ are 12/15 (80%), 7/10 (70%), and 8/12 (66.7%), respectively. $R1$ is generated first because it has the highest accuracy. After generating $R1$, it is clear that the positive examples covered by the rule must be removed so that the next rule generated by the algorithm is different than $R1$. Next, suppose the algorithm is given the choice of generating either $R2$ or $R3$. Even though $R2$ has higher accuracy than $R3$, $R1$ and $R3$ together cover 18 positive examples and 5 negative examples (resulting in an overall accuracy of 78.3%), whereas $R1$ and $R2$ together cover 19 positive examples and 6 negative examples (resulting in an overall accuracy of 76%). The incremental impact of $R2$ or $R3$ on accuracy is more evident when the positive and negative examples covered by $R1$ are removed before computing their accuracies. In particular, if positive examples covered by $R1$ are not removed, then we may overestimate the effective accuracy of $R3$, and if negative examples are not removed, then we may underestimate the accuracy of $R3$. In the latter case, we might end up preferring $R2$ over $R3$ even though half of the false positive errors committed by $R3$ have already been accounted for by the preceding rule, $R1$.

RIPPER Algorithm

To illustrate the direct method, we consider a widely used rule induction algorithm called RIPPER. This algorithm scales almost linearly with the number of training examples and is particularly suited for building models from data sets with imbalanced class distributions. RIPPER also works well with noisy data sets because it uses a validation set to prevent model overfitting.

For two-class problems, RIPPER chooses the majority class as its default class and learns the rules for detecting the minority class. For multiclass problems, the classes are ordered according to their frequencies. Let $(y_1, y_2, \ldots, y_c)$ be the ordered classes, where y_1 is the least frequent class and y_c is the most frequent class. During the first iteration, instances that belong to y_1 are labeled as positive examples, while those that belong to other classes are labeled as negative examples. The sequential covering method is used to generate rules that discriminate between the positive and negative examples. Next, RIPPER extracts rules that distinguish y_2 from other remaining classes. This process is repeated until we are left with y_c, which is designated as the default class.

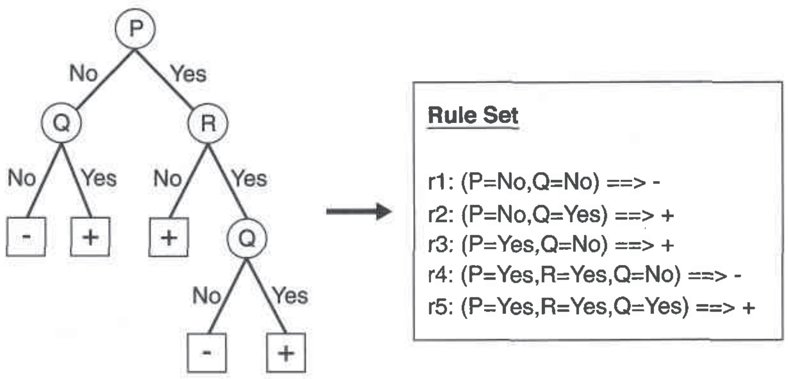
Rule Growing RIPPER employs a general-to-specific strategy to grow a rule and the FOIL's information gain measure to choose the best conjunct to be added into the rule antecedent. It stops adding conjuncts when the rule starts covering negative examples. The new rule is then pruned based on its performance on the validation set. The following metric is computed to determine whether pruning is needed: $(p-n)/(p+n)$, where p (n) is the number of positive (negative) examples in the validation set covered by the rule. This metric is monotonically related to the rule's accuracy on the validation set. If the metric improves after pruning, then the conjunct is removed. Pruning is done starting from the last conjunct added to the rule. For example, given a rule $ABCD \longrightarrow y$, RIPPER checks whether D should be pruned first, followed by CD, BCD, etc. While the original rule covers only positive examples, the pruned rule may cover some of the negative examples in the training set.

Building the Rule Set After generating a rule, all the positive and negative examples covered by the rule are eliminated. The rule is then added into the rule set as long as it does not violate the stopping condition, which is based on the minimum description length principle. If the new rule increases the total description length of the rule set by at least d bits, then RIPPER stops adding rules into its rule set (by default, d is chosen to be 64 bits). Another stopping condition used by RIPPER is that the error rate of the rule on the validation set must not exceed 50%.

RIPPER also performs additional optimization steps to determine whether some of the existing rules in the rule set can be replaced by better alternative rules. Readers who are interested in the details of the optimization method may refer to the reference cited at the end of this chapter.

5.1.5 Indirect Methods for Rule Extraction

This section presents a method for generating a rule set from a decision tree. In principle, every path from the root node to the leaf node of a decision tree can be expressed as a classification rule. The test conditions encountered along the path form the conjuncts of the rule antecedent, while the class label at the leaf node is assigned to the rule consequent. Figure 5.5 shows an example of a rule set generated from a decision tree. Notice that the rule set is exhaustive and contains mutually exclusive rules. However, some of the rules can be simplified as shown in the next example.



Rule Set

r1: (P=No,Q=No) ==> -
r2: (P=No,Q=Yes) ==> +
r3: (P=Yes,Q=No) ==> +
r4: (P=Yes,R=Yes,Q=No) ==> -
r5: (P=Yes,R=Yes,Q=Yes) ==> +

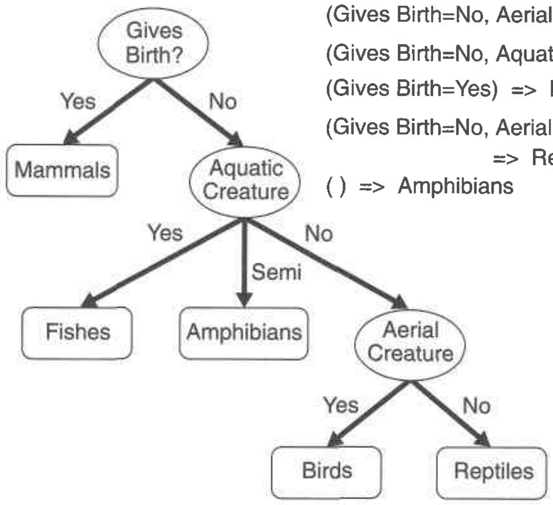
Figure 5.5. Converting a decision tree into classification rules.

Example 5.2. Consider the following three rules from Figure 5.5:

$$r2 : (\text{P} = \text{No}) \wedge (\text{Q} = \text{Yes}) \longrightarrow +$$
$$r3 : (\text{P} = \text{Yes}) \wedge (\text{R} = \text{No}) \longrightarrow +$$
$$r5 : (\text{P} = \text{Yes}) \wedge (\text{R} = \text{Yes}) \wedge (\text{Q} = \text{Yes}) \longrightarrow +$$

Observe that the rule set always predicts a positive class when the value of Q is Yes. Therefore, we may simplify the rules as follows:

$$r2': \ (\text{Q} = \text{Yes}) \longrightarrow +$$
$$r3: \ (\text{P} = \text{Yes}) \wedge (\text{R} = \text{No}) \longrightarrow +.$$



Rule-Based Classifier:

(Gives Birth=No, Aerial Creature=Yes) => Birds
(Gives Birth=No, Aquatic Creature=Yes) => Fishes
(Gives Birth=Yes) => Mammals
(Gives Birth=No, Aerial Creature=No, Aquatic Creature=No) => Reptiles
( ) => Amphibians

Figure 5.6. Classification rules extracted from a decision tree for the vertebrate classification problem.

r_3 is retained to cover the remaining instances of the positive class. Although the rules obtained after simplification are no longer mutually exclusive, they are less complex and are easier to interpret. ■

In the following, we describe an approach used by the C4.5rules algorithm to generate a rule set from a decision tree. Figure 5.6 shows the decision tree and resulting classification rules obtained for the data set given in Table 5.2.

Rule Generation Classification rules are extracted for every path from the root to one of the leaf nodes in the decision tree. Given a classification rule $r : A \longrightarrow y$, we consider a simplified rule, $r' : A' \longrightarrow y$, where A' is obtained by removing one of the conjuncts in A. The simplified rule with the lowest pessimistic error rate is retained provided its error rate is less than that of the original rule. The rule-pruning step is repeated until the pessimistic error of the rule cannot be improved further. Because some of the rules may become identical after pruning, the duplicate rules must be discarded.

Rule Ordering After generating the rule set, C4.5rules uses the class-based ordering scheme to order the extracted rules. Rules that predict the same class are grouped together into the same subset. The total description length for each subset is computed, and the classes are arranged in increasing order of their total description length. The class that has the smallest description length is given the highest priority because it is expected to contain the best set of rules. The total description length for a class is given by $L_{\text{exception}} + g \times L_{\text{model}}$, where $L_{\text{exception}}$ is the number of bits needed to encode the misclassified examples, L_{model} is the number of bits needed to encode the model, and g is a tuning parameter whose default value is 0.5. The tuning parameter depends on the number of redundant attributes present in the model. The value of the tuning parameter is small if the model contains many redundant attributes.

5.1.6 Characteristics of Rule-Based Classifiers

A rule-based classifier has the following characteristics:

- The expressiveness of a rule set is almost equivalent to that of a decision tree because a decision tree can be represented by a set of mutually exclusive and exhaustive rules. Both rule-based and decision tree classifiers create rectilinear partitions of the attribute space and assign a class to each partition. Nevertheless, if the rule-based classifier allows multiple rules to be triggered for a given record, then a more complex decision boundary can be constructed.

- Rule-based classifiers are generally used to produce descriptive models that are easier to interpret, but gives comparable performance to the decision tree classifier.

- The class-based ordering approach adopted by many rule-based classifiers (such as RIPPER) is well suited for handling data sets with imbalanced class distributions.

5.2 Nearest-Neighbor classifiers

The classification framework shown in Figure 4.3 involves a two-step process: (1) an inductive step for constructing a classification model from data, and (2) a deductive step for applying the model to test examples. Decision tree and rule-based classifiers are examples of **eager learners** because they are designed to learn a model that maps the input attributes to the class label as soon as the training data becomes available. An opposite strategy would be to delay the process of modeling the training data until it is needed to classify the test examples. Techniques that employ this strategy are known as **lazy learners**. An example of a lazy learner is the **Rote classifier**, which memorizes the entire training data and performs classification only if the attributes of a test instance match one of the training examples exactly. An obvious drawback of