6

DECISION TREES AND DECISION RULES

Chapter Objectives

- Analyze the characteristics of a logic-based approach to classification problems.
- Describe the differences between decision-tree and decision-rule representations in a final classification model.
- Explain in-depth the C4.5 algorithm for generating decision trees and decision rules.
- Identify the required changes in the C4.5 algorithm when missing values exist in training or testing data set.
- Introduce the basic characteristics of Classification and Regression Trees (CART) algorithm and Gini index.
- Know when and how to use pruning techniques to reduce the complexity of decision trees and decision rules.
- Summarize the limitations of representing a classification model by decision trees and decision rules.

Data Mining: Concepts, Models, Methods, and Algorithms, Second Edition. Mehmed Kantardzic.

Decision trees and decision rules are data-mining methodologies applied in many real-world applications as a powerful solution to classification problems. Therefore, to be begin with, let us briefly summarize the basic principles of classification. In general, classification is a process of learning a function that maps a data item into one of several predefined classes. Every classification based on inductive-learning algorithms is given as an input a set of samples that consist of vectors of attribute values (also called feature vectors) and a corresponding class. The goal of learning is to create a classification model, known as a *classifier*, which will predict, with the values of its available input attributes, the class for some entity (a given sample). In other words, classification is the process of assigning a discrete label value (class) to an unlabeled record, and a classifier is a model (a result of classification) that predicts one attribute—class of a sample—when the other attributes are given. In doing so, samples are divided into predefined groups. For example, a simple classification might group customer billing records into two specific classes: those who pay their bills within 30 days and those who takes longer than 30 days to pay. Different classification methodologies are applied today in almost every discipline where the task of classification, because of the large amount of data, requires automation of the process. Examples of classification methods used as a part of data-mining applications include classifying trends in financial market and identifying objects in large image databases.

A more formalized approach to classification problems is given through its graphical interpretation. A data set with n features may be thought of as a collection of discrete points (one per example) in an n-dimensional space. A classification rule is a hypercube that contains one or more of these points. When there is more than one cube for a given class, all the cubes are OR-ed to provide a complete classification for the class, such as the example of two-dimensional (2-D) classes in Figure 6.1. Within a cube the conditions for each part are AND-ed. The size of a cube indicates its generality, that is, the larger the cube is, the more vertices it contains and potentially covers more sample points.

In a classification model, the connection between classes and other properties of the samples can be defined by something as simple as a flowchart or as complex and unstructured as a procedure manual. Data-mining methodologies restrict discussion to formalized, "executable" models of classification, and there are two very different ways in which they can be constructed. On the one hand, the model might be obtained by

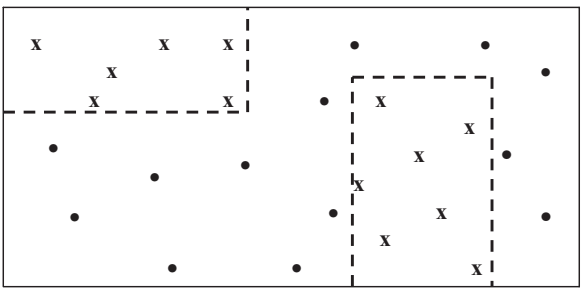

Figure 6.1. Classification of samples in a 2-D space.

interviewing the relevant expert or experts, and most knowledge-based systems have been built this way despite the well-known difficulties in taking this approach. Alternatively, numerous recorded classifications might be examined and a model constructed inductively by generalizing from specific examples that are of primary interest for data-mining applications.

The statistical approach to classification explained in Chapter 5 gives one type of model for classification problems: summarizing the statistical characteristics of the set of samples. The other approach is based on logic. Instead of using math operations like addition and multiplication, the logical model is based on expressions that are evaluated as true or false by applying Boolean and comparative operators to the feature values. These methods of modeling give accurate classification results compared with other nonlogical methods, and they have superior explanatory characteristics. Decision trees and decision rules are typical data-mining techniques that belong to a class of methodologies that give the output in the form of logical models.

6.1 DECISION TREES

A particularly efficient method of producing classifiers from data is to generate a decision tree. The decision-tree representation is the most widely used logic method. There is a large number of decision-tree induction algorithms described primarily in the machine-learning and applied-statistics literature. They are supervised learning methods that construct decision trees from a set of input–output samples. It is an efficient nonparametric method for classification and regression. A decision tree is a hierarchical model for supervised learning where the *local region* is identified in a sequence of recursive *splits* through decision nodes with test function. A decision tree is also a nonparametric model in the sense that we *do not assume any parametric form* for the class density.

A typical decision-tree learning system adopts a top-down strategy that searches for a solution in a part of the search space. It guarantees that a simple, but not necessarily the simplest, tree will be found. A decision tree consists of *nodes* where attributes are tested. In a univariate tree, for each internal node, the test uses only one of the attributes for testing. The outgoing *branches* of a node correspond to all the possible outcomes of the test at the node. A simple decision tree for classification of samples with two input attributes X and Y is given in Figure 6.2. All samples with feature values $X > 1$ and $Y = B$ belong to Class2, while the samples with values $X < 1$ belong to Class1, whatever the value for feature Y is. The samples, at a non-leaf node in the tree structure, are thus partitioned along the branches and each child node gets its corresponding subset of samples. Decision trees that use univariate splits have a simple representational form, making it relatively easy for the user to understand the inferred model; at the same time, they represent a restriction on the expressiveness of the model. In general, any restriction on a particular tree representation can significantly restrict the functional form and thus the approximation power of the model. A well-known tree-growing algorithm for generating decision trees based on univariate splits is Quinlan's *ID3* with an extended version called *C4.5*. Greedy search methods, which

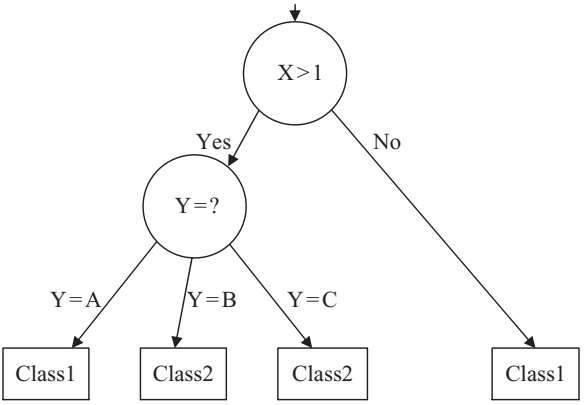

Figure 6.2. A simple decision tree with the tests on attributes X and Y.

involve growing and pruning decision-tree structures, are typically employed in these algorithms to explore the exponential space of possible models.

The ID3 algorithm starts with all the training samples at the root node of the tree. An attribute is selected to partition these samples. For each value of the attribute a branch is created, and the corresponding subset of samples that have the attribute value specified by the branch is moved to the newly created child node. The algorithm is applied recursively to each child node until all samples at a node are of one class. Every path to the leaf in the decision tree represents a classification rule. Note that the critical decision in such a top-down decision tree-generation algorithm is the choice of an attribute at a node. Attribute selection in ID3 and C4.5 algorithms are based on minimizing an information entropy measure applied to the examples at a node. The approach based on information entropy insists on minimizing the number of tests that will allow a sample to classify in a database. The attribute-selection part of ID3 is based on the assumption that the complexity of the decision tree is strongly related to the amount of information conveyed by the value of the given attribute. An information-based heuristic selects the attribute providing the highest information gain, that is, the attribute that minimizes the information needed in the resulting subtree to classify the sample. An extension of ID3 is the C4.5 algorithm, which extends the domain of classification from categorical attributes to numeric ones. The measure favors attributes that result in partitioning the data into subsets that have a low-class entropy, that is, when the majority of examples in it belong to a single class. The algorithm basically chooses the attribute that provides the maximum degree of discrimination between classes locally. More details about the basic principles and implementation of these algorithms will be given in the following sections.

To apply some of the methods, which are based on the inductive-learning approach, several key requirements have to be satisfied:

1. *Attribute-Value Description.* The data to be analyzed must be in a flat-file form—all information about one object or example must be expressible in terms

of a fixed collection of properties or attributes. Each attribute may have either discrete or numeric values, but the attributes used to describe samples must not vary from one case to another. This restriction rules out domains in which samples have an inherently variable structure.

2. *Predefined Classes.* The categories to which samples are to be assigned must have been established beforehand. In the terminology of machine learning this is supervised learning.
3. *Discrete Classes.* The classes must be sharply delineated: A case either does or does not belong to a particular class. It is expected that there will be far more samples than classes.
4. *Sufficient Data.* Inductive generalization given in the form of a decision tree proceeds by identifying patterns in data. The approach is valid if enough number of robust patterns can be distinguished from chance coincidences. As this differentiation usually depends on statistical tests, there must be sufficient number of samples to allow these tests to be effective. The amount of data required is affected by factors such as the number of properties and classes and the complexity of the classification model. As these factors increase, more data will be needed to construct a reliable model.
5. *"Logical" Classification Models.* These methods construct only such classifiers that can be expressed as decision trees or decision rules. These forms essentially restrict the description of a class to a logical expression whose primitives are statements about the values of particular attributes. Some applications require weighted attributes or their arithmetic combinations for a reliable description of classes. In these situations logical models become very complex and, in general, they are not effective.

6.2 C4.5 ALGORITHM: GENERATING A DECISION TREE

The most important part of the C4.5 algorithm is the process of generating an initial decision tree from the set of training samples. As a result, the algorithm generates a classifier in the form of a decision tree: a structure with two types of nodes—*a leaf* indicating a class, or *a decision node* specifying some tests to be carried out on a single-attribute value, with one branch and a subtree for each possible outcome of the test.

A decision tree can be used to classify a new sample by starting at the root of the tree and moving through it until a leaf is encountered. At each non-leaf decision node, the features' outcome for the test at the node is determined and attention shifts to the root of the selected subtree. For example, if the classification model of the problem is given with the decision tree in Figure 6.3a, and the sample for classification in Figure 6.3b, then the algorithm will create the path through the nodes **A**, **C**, and **F** (leaf node) until it makes the final classification decision: *CLASS2*.

The skeleton of the C4.5 algorithm is based on Hunt's Concept Learning System (CLS) method for constructing a decision tree from a set T of training samples. Let the classes be denoted as $\{C_1, C_2, \ldots, C_k\}$. There are three possibilities for the content of the set T:

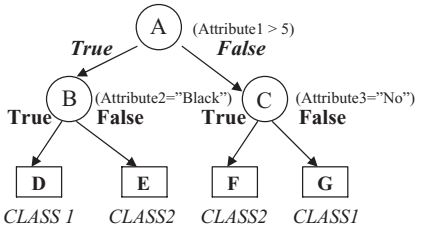

Figure 6.3. Classification of a new sample based on the decision-tree model. (a) Decision tree; (b) An example for classification.

1. T contains one or more samples, all belonging to a single class C_j. The decision tree for T is a leaf-identifying class C_j.
2. T contains no samples. The decision tree is again a leaf but the class to be associated with the leaf must be determined from information other than T, such as the overall majority class in T. The C4.5 algorithm uses as a criterion the most frequent class at the parent of the given node.
3. T contains samples that belong to a mixture of classes. In this situation, the idea is to refine T into subsets of samples that are heading toward a single-class collection of samples. Based on single attribute, an appropriate test that has one or more mutually exclusive outcomes $\{O_1, O_2, \ldots, O_n\}$ is chosen. T is partitioned into subsets $T_1, T_2, \ldots, T_n$, where T_i contains all the samples in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test and one branch for each possible outcome (examples of this type of nodes are nodes A, B, and C in the decision tree in Fig. 6.3a).

The same tree-building procedure is applied recursively to each subset of training samples, so that the *i*th branch leads to the decision tree constructed from the subset T_i of the training samples. The successive division of the set of training samples proceeds until all the subsets consist of samples belonging to a single class.

The tree-building process is not uniquely defined. For different tests, even for a different order of their application, different trees will be generated. Ideally, we would like to choose a test at each stage of sample-set splitting so that the final tree is small. Since we are looking for a compact decision tree that is consistent with the training set, why not explore all possible trees and select the simplest? Unfortunately, the problem of finding the smallest decision tree consistent with a training data set is NP-complete. Enumeration and analysis of all possible trees will cause a combinatorial explosion for any real-world problem. For example, for a small database with five attributes and only 20 training examples, the possible number of decision trees is greater than 10^6, depending on the number of different values for every attribute. Therefore, most decision tree-construction methods are non-backtracking, greedy algorithms.

Once a test has been selected using some heuristics to maximize the measure of progress and the current set of training cases has been partitioned, the consequences of alternative choices are not explored. The measure of progress is a local measure, and the gain criterion for a test selection is based on the information available for a given step of data splitting.

Suppose we have the task of selecting a possible test with n outcomes (n values for a given feature) that partitions the set T of training samples into subsets T_1, T_2, . . . , T_n. The only information available for guidance is the distribution of classes in T and its subsets T_i. If S is any set of samples, let *freq*(C_i, S) stand for the number of samples in S that belong to class C_i (out of k possible classes), and let |S| denote the number of samples in the set S.

The original ID3 algorithm used a criterion called *gain* to select the attribute to be tested that is based on the information theory concept: *entropy*. The following relation gives the computation of the entropy of the set T (bits are units):

$$\text{Info}(T) = -\sum_{i=1}^{k}((\text{freq}(C_i, T)/|T|)\cdot \log_2(\text{freq}(C_i, T)/|T|))$$

Now consider a similar measurement after T has been partitioned in accordance with n outcomes of one attribute test X. The expected information requirement can be found as the weighted sum of entropies over the subsets:

$$\text{Info}_x(T) = \sum_{i=1}^{n}((|T_i|/|T|)\cdot \text{Info}(T_i))$$

The quantity

$$\text{Gain}(X) = \text{Info}(T) - \text{Info}_x(T)$$

measures the information that is gained by partitioning T in accordance with the test X. The gain criterion selects a test X to maximize Gain(X), that is, this criterion will select an attribute with the highest information gain.

Let us analyze the application of these measures and the creation of a decision tree for one simple example. Suppose that the database T is given in a flat form in which each of 14 examples (cases) is described by three input attributes and belongs to one of two given classes: CLASS1 or CLASS2. The database is given in tabular form in Table 6.1.

Nine samples belong to CLASS1 and five samples to CLASS2, so the entropy before splitting is

$$\text{Info}(T) = -9/14\log_2(9/14) - 5/14\log_2(5/14) = 0.940 \text{ bits}$$

After using Attribute1 to divide the initial set of samples T into three subsets (test x_1 represents the selection one of three values A, B, or C), the resulting information is given by:

TABLE 6.1. A Simple Flat Database of Examples for Training

Database T:

Attribute1	Attribute2	Attribute3	Class
A	70	True	CLASS1
A	90	True	CLASS2
A	85	False	CLASS2
A	95	False	CLASS2
A	70	False	CLASS1
B	90	True	CLASS1
B	78	False	CLASS1
B	65	True	CLASS1
B	75	False	CLASS1
C	80	True	CLASS2
C	70	True	CLASS2
C	80	False	CLASS1
C	80	False	CLASS1
C	96	False	CLASS1

$$\begin{aligned}\text{Info}_{x1}(T) &= 5/14\ (-2/5\ \log_2\ [(/5) - 3/5\ \log_2\ (3/5)) \\ &\quad + 4/14\ (-4/4\ \log_2\ (4/4) - 0/4\ \log_2\ (0/4)) \\ &\quad + 5/14\ (-3/5\ \log_2\ (3/5) - 2/5\ \log_2\ (2/5)) \\ &= 0.694 \text{ bits}\end{aligned}$$

The information gained by this test x_1 is

$$\text{Gain}\ (x_1) = 0.940 - 0.694 = 0.246 \text{ bits}$$

If the test and splitting is based on Attribute3 (test x_2 represents the selection one of two values True or False), a similar computation will give new results:

$$\begin{aligned}\text{Info}_{x2}(T) &= 6/14\ (-3/6\ \log_2\ (3/6) - 3/6\ \log_2\ (3/6)) \\ &\quad + 8/14\ (-6/8\ \log_2\ (6/8) - 2/8\ \log_2\ (2/8)) \\ &= 0.892 \text{ bits}\end{aligned}$$

and corresponding gain is

$$\text{Gain}\ (x_2) = 0.940 - 0.892 = 0.048 \text{ bits}$$

Based on the gain criterion, the decision-tree algorithm will select test x_1 as an initial test for splitting the database T because this gain is higher. To find the optimal test it will be necessary to analyze a test on Attribute2, which is a numeric feature with continuous values. In general, C4.5 contains mechanisms for proposing three types of tests:

1. The "standard" test on a discrete attribute, with one outcome and one branch for each possible value of that attribute (in our example these are both tests x_1 for Attribute1 and x_2 for Attribute3).
2. If attribute Y has continuous numeric values, a binary test with outcomes $Y \le Z$ and $Y > Z$ could be defined by comparing its value against a threshold value Z.
3. A more complex test is also based on a discrete attribute, in which the possible values are allocated to a variable number of groups with one outcome and branch for each group.

While we have already explained standard test for categorical attributes, additional explanations are necessary about a procedure for establishing tests on attributes with numeric values. It might seem that tests on continuous attributes would be difficult to formulate, since they contain an arbitrary threshold for splitting all values into two intervals. But there is an algorithm for the computation of optimal threshold value Z. The training samples are first sorted on the values of the attribute Y being considered. There are only a finite number of these values, so let us denote them in sorted order as $\{v_1, v_2, \ldots, v_m\}$. Any threshold value lying between v_i and v_{i+1} will have the same effect as dividing the cases into those whose value of the attribute Y lies in $\{v_1, v_2, \ldots, v_i\}$ and those whose value is in $\{v_{i+1}, v_{i+2}, \ldots, v_m\}$. There are thus only m-1 possible splits on Y, all of which should be examined systematically to obtain an optimal split. It is usual to choose the midpoint of each interval, $(v_i + v_{i+1})/2$, as the representative threshold. The algorithm C4.5 differs in choosing as the threshold a smaller value v_i for every interval $\{v_i, v_{i+1}\}$, rather than the midpoint itself. This ensures that the threshold values appearing in either the final decision tree or rules or both actually occur in the database.

To illustrate this threshold-finding process, we could analyze, for our example of database T, the possibilities of Attribute2 splitting. After a sorting process, the set of values for Attribute2 is {65, 70, 75, 78, 80, 85, 90, 95, 96} and the set of potential threshold values Z is {65, 70, 75, 78, 80, 85, 90, 95}. Out of these eight values the optimal Z (with the highest information gain) should be selected. For our example, the optimal Z value is Z = 80 and the corresponding process of information-gain computation for the test x_3 (Attribute2 ≤ 80 or Attribute2 > 80) is the following:

$$\begin{aligned}\text{Info}_{x3}(T) &= 9/14\ (-7/9 \log_2 (7/9) - 2/9 \log_2 (2/9)) \\ &\quad + 5/14\ (-2/5 \log_2 (2/5) - 3/5 \log_2 (3/5)) \\ &= 0.837 \text{ bits}\end{aligned}$$

$$\text{Gain}(x_3) = 0.940 - 0.837 = 0.103 \text{ bits}$$

Now, if we compare the information gain for the three attributes in our example, we can see that Attribute1 still gives the highest gain of 0.246 bits and therefore this attribute will be selected for the first splitting in the construction of a decision tree. The root node will have the test for the values of Attribute1, and three branches will be created, one for each of the attribute values. This initial tree with the corresponding subsets of samples in the children nodes is represented in Figure 6.4.

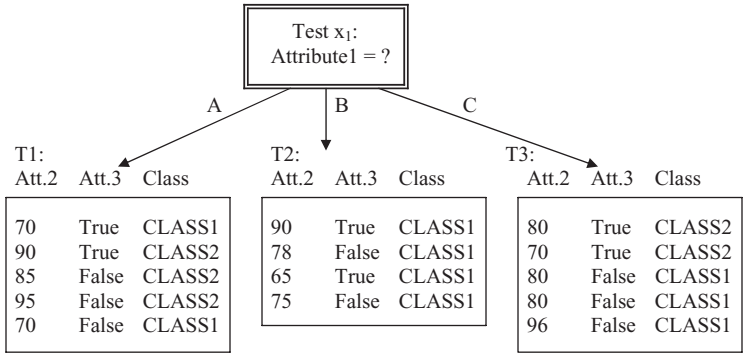

Figure 6.4. Initial decision tree and subset cases for a database in Table 6.1.

After initial splitting, every child node has several samples from the database, and the entire process of test selection and optimization will be repeated for every child node. Because the child node for test x_1, Attribute1 = B, has four cases and all of them are in CLASS1, this node will be the leaf node, and no additional tests are necessary for this branch of the tree.

For the remaining child node where we have five cases in subset T_1, tests on the remaining attributes can be performed; an optimal test (with maximum information gain) will be test x_4 with two alternatives: Attribute2 ≤ 70 or Attribute2 > 70.

$$\text{Info}(T_1) = -2/5 \log_2 (2/5) - 3/5 \log_2 (3/5) = 0.97 \text{ bits}$$

Using Attribute2 to divide T_1 into two subsets (test x_4 represents the selection of one of two intervals), the resulting information is given by:

$$\begin{aligned}\text{Info}_{x4}(T_1) &= 2/5\ (-2/2 \log_2 (2/2) - 0/2 \log_2 (0/2)) \\ &\quad + 3/5\ (-0/3 \log_2 (0/3) - 3/3 \log_2 (3/3)) \\ &= 0 \text{ bits}\end{aligned}$$

The information gained by this test is maximal:

$$\text{Gain } (x_4) = 0.97 - 0 = 0.97 \text{ bits}$$

and two branches will create the final leaf nodes because the subsets of cases in each of the branches belong to the same class.

A similar computation will be carried out for the third child of the root node. For the subset T_3 of the database T, the selected optimal test x_5 is the test on Attribute3 values. Branches of the tree, Attribute3 = True and Attribute3 = False, will create uniform subsets of cases that belong to the same class. The final decision tree for database T is represented in Figure 6.5.

Alternatively, a decision tree can be presented in the form of an executable code (or pseudo-code) with if-then constructions for branching into a tree structure. The transformation of a decision tree from one representation to the other is very simple

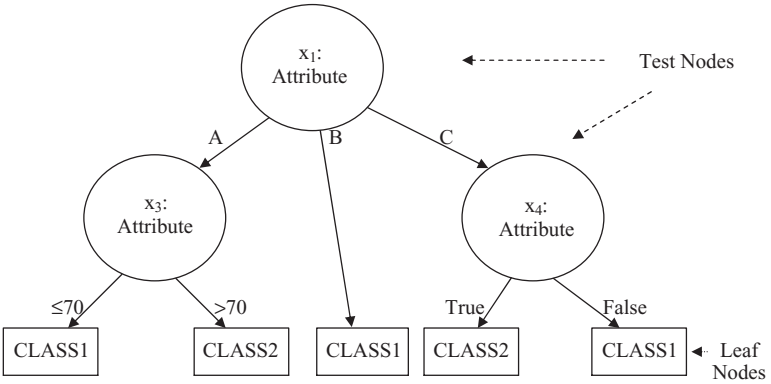

Figure 6.5. A final decision tree for database T given in Table 6.1.

```
If      Attribute1 = A
        Then
                If      Attribute2 <= 70
                        Then
                                Classification = CLASS1;
                        Else
                                Classification = CLASS2;
Elseif  Attribute1 = B
        Then
                                Classification = CLASS1;
Elseif  Attribute1 = C
        Then
                If      Attribute3 = True
                        Then
                                Classification = CLASS2;
                        Else
                                Classification = CLASS1.
```

Figure 6.6. A decision tree in the form of pseudocode for the database T given in Table 6.1.

and straightforward. The final decision tree for our example is given in pseudocode in Figure 6.6.

While the gain criterion has had some good results in the construction of compact decision trees, it also has one serious deficiency: a strong bias in favor of tests with many outcomes. A solution was found in some kinds of normalization. By analogy with the definition of Info(S), an additional parameter was specified:

$$\text{Split-info}(X) = -\sum_{i=1}^{n} ((|T_i|/|T|)\log_2(|T_i|/|T|)$$

This represented the potential information generated by dividing set T into n subsets T_i. Now, a new gain measure could be defined:

$$\text{Gain-ratio}(X) = \text{gain}(X) / \text{Split-info}(X)$$

This new gain measure expresses the proportion of information generated by the split that is useful, that is, that appears helpful in classification. The gain-ratio criterion also selects a test that maximizes the ratio given earlier. This criterion is robust and typically gives a consistently better choice of a test than the previous gain criterion. A computation of the gain-ratio test can be illustrated for our example. To find the gain-ratio measure for the test x_1, an additional parameter Split-info(x_1) is calculated:

$$\text{Split-info}(x_1) = -5/14 \log_2 (5/14) - 4/14 \log_2 (4/14) - 5/14 \log_2 (5/14) = 1.577 \text{ bits}$$

$$\text{Gain-ratio}(x_1) = 0.246/1.557 = 0.156$$

A similar procedure should be performed for other tests in the decision tree. Instead of gain measure, the maximal gain ratio will be the criterion for attribute selection, along with a test to split samples into subsets. The final decision tree created using this new criterion for splitting a set of samples will be the most compact.

6.3 UNKNOWN ATTRIBUTE VALUES

The previous version of the C4.5 algorithm is based on the assumption that all values for all attributes are determined. But in a data set, often some attribute values for some samples can be missing—such incompleteness is typical in real-world applications. This might occur because the value is not relevant to a particular sample, or it was not recorded when the data were collected, or an error was made by the person entering data into a database. To solve the problem of missing values, there are two choices:

1. Discard all samples in a database with missing data.
2. Define a new algorithm or modify an existing algorithm that will work with missing data.

The first solution is simple but unacceptable when large amounts of missing values exist in a set of samples. To address the second alternative, several questions must be answered:

1. How does one compare two samples with different numbers of unknown values?
2. Training samples with unknown values cannot be associated with a particular value of the test, and so they cannot be assigned to any subsets of cases. How should these samples be treated in the partitioning?
3. In a testing phase of classification, how does one treat a missing value if the test is on the attribute with the missing value?

All these and many other questions arise with any attempt to find a solution for missing data. Several classification algorithms that work with missing data are usually

based on filling in a missing value with the most probable value, or on looking at the probability distribution of all values for the given attribute. None of these approaches is uniformly superior.

In C4.5, it is an accepted principle that samples with unknown values are distributed probabilistically according to the relative frequency of known values. Let Info(T) and $\text{Info}_x(T)$ be calculated as before, except that only samples with known values of attributes are taken into account. Then the gain parameter can reasonably be corrected with a factor F, which represents the probability that a given attribute is known (F = number of samples in the database with a known value for a given attribute/total number of samples in a data set). The new gain criterion will have the form

$$\text{Gain}(x) = F\ (\text{Info}(T) - \text{Info}_x(T))$$

Similarly, Split-info(x) can be altered by regarding the samples with unknown values as an additional group in splitting. If the test x has n outcomes, its Split-info(x) is computed as if the test divided the data set into n + 1 subsets. This modification has a direct influence on the final value of the modified criterion Gain-ratio(x).

Let us explain the modifications of the C4.5 decision-tree methodology applied on one example. The database is similar to the previous one (Table 6.1), only there is now one value missing for Attribute1 denoted by "?" as presented in Table 6.2.

The computation of the gain parameter for Attribute1 is similar to as before, only the missing value corrects some of the previous steps. Eight out of the 13 cases with values for Attribute1 belong to CLASS1 and five cases to CLASS2, so the entropy before splitting is

TABLE 6.2. A Simple Flat Database of Examples with One Missing Value

Database T:

Attribute1	Attribute2	Attribute3	Class
A	70	True	CLASS1
A	90	True	CLASS2
A	85	False	CLASS2
A	95	False	CLASS2
A	70	False	CLASS1
?	90	True	CLASS1
B	78	False	CLASS1
B	65	True	CLASS1
B	75	False	CLASS1
C	80	True	CLASS2
C	70	True	CLASS2
C	80	False	CLASS1
C	80	False	CLASS1
C	96	False	CLASS1

$$\text{Info}(T) = -8/13 \log_2 (8/13) - 5/13 \log_2 (5/13) = 0.961 \text{ bits}$$

After using Attribute1 to divide T into three subsets (test x_1 represents the selection one of three values A, B, or C), the resulting information is given by

$$\begin{aligned}\text{Info}_{x1}(T) = {} & 5/13\ (-2/5 \log_2 [2/5] - 3/5 \log_2 [3/5]) \\ & + 3/13\ (-3/3 \log_2 [3/3] - 0/3 \log_2 [0/3]) \\ & + 5/13\ (-3/5 \log_2 [3/5] - 2/5 \log_2 [2/5]) \\ = {} & 0.747 \text{ bits}\end{aligned}$$

The information gained by this test is now corrected with the factor F (F = 13/14 for our example):

$$\text{Gain } (x1) = 13/14\ (0.961 - 0.747) = 0.199 \text{ bits}$$

The gain for this test is slightly lower than the previous value of 0.216 bits. The split information, however, is still determined from the entire training set and is larger, since there is an extra category for unknown values.

$$\begin{aligned}\text{Split-info}(x1) = {} & -(5/14 \log(5/14) + 3/14 \log(3/14) + 5/14 \log(5/14) + \\ & + 1/14 \log(1/14)) = \\ = {} & 1.8\end{aligned}$$

Additionally, the concept of partitioning must be generalized. With every sample a new parameter, probability, is associated. When a case with known value is assigned from T to subset T_i, the probability of it belonging to T_i is 1, and in all other subsets is 0. When a value is not known, only a weaker probabilistic statement can be made. C4.5 therefore associates with each sample (having a missing value) in each subset T_i a weight w, representing the probability that the case belongs to each subset. To make the solution more general, it is necessary to take into account that the probabilities of samples before splitting are not always equal to one (in subsequent iterations of the decision-tree construction). Therefore, new parameter w_{new} for missing values after splitting is equal to the old parameter w_{old} before splitting multiplied by the probability that the sample belongs to each subset $P(T_i)$, or more formally:

$$w_{new} = w_{old} \cdot P(T_i)$$

For example, the record with the missing value, given in the database in Figure 6.7, will be represented in all three subsets after the splitting set T into subsets T_i using test x_1 on Attribute1. New weights w_i will be equal to probabilities 5/13, 3/13, and 5/13, because the initial (old) value for w is equal to one. The new subsets are given in Figure 6.7. $|T_i|$ can now be reinterpreted in C4.5 not as a number of elements in a set T_i, but as a sum of all weights w for the given set T_i. From Figure 6.7, the new values are computed as: $|T_1| = 5 + 5/13$, $|T_2| = 3 + 3/13$, and $|T_3| = 5 + 5/13$.

T_1: (Attribute1 = A)

Att.2	Att.3	Class	w
70	True	CLASS1	1
90	True	CLASS2	1
85	False	CLASS2	1
95	False	CLASS2	1
70	False	CLASS1	1
90	***True***	***CLASS1***	***5/13***

T_2: (Attribute1 = B)

Att.2	Att.3	Class	w
90	***True***	***CLASS1***	***3/13***
78	False	CLASS1	1
65	True	CLASS1	1
75	False	CLASS1	1

T_3: (Attribute1 = C)

Att.2	Att.3	Class	w
80	True	CLASS2	1
70	True	CLASS2	1
80	False	CLASS1	1
80	False	CLASS1	1
96	False	CLASS1	1
90	***True***	***CLASS1***	***5/13***

Figure 6.7. Results of test x_1 are subsets T_i (initial set T is with missing value).

```
If    Attribute1 = A
      Then
            If    Attribute2 <= 70
                  Then
                        Classification = CLASS1     (2.0 / 0);
                  Else
                        Classification = CLASS2     (3.4 / 0.4);
      Elseif Attribute1 = B
      Then
                        Classification = CLASS1     (3.2 / 0);
      Elseif Attribute1 = C
      Then
            If    Attribute3 = True
                  Then
                        Classification = CLASS2     (2.4 / 0.4);
                  Else
                        Classification = CLASS1     (3.0 / 0).
```

Figure 6.8. Decision tree for the database T with missing values.

If these subsets are partitioned further by the tests on Attribute2 and Attribute3, the final decision tree for a data set with missing values has the form shown in Figure 6.8

The decision tree in Figure 6.8 has much the same structure as before (Fig. 6.6), but because of the ambiguity in final classification, every decision is attached with two parameters in a form ($|T_i|/E$). $|T_i|$ is the sum of the fractional samples that reach the leaf and E is the number of samples that belong to classes other than the nominated class.

For example, (3.4/0.4) means that 3.4 (or 3 + 5/13) fractional training samples reached the leaf, of which 0.4 (or 5/13) did not belong to the class assigned to the leaf. It is possible to express the $|T_i|$ and E parameters in percentages:

$$3/3.4 \cdot 100\% = 88\%$$ of cases at a given leaf would be classified as CLASS2.

$$0.4/3.4 \cdot 100\% = 12\%$$ of cases at a given leaf would be classified as CLASS1.

A similar approach is taken in C4.5 when the decision tree is used to classify a sample previously not present in a database, that is, the *testing phase*. If all attribute values are known then the process is straightforward. Starting with a root node in a decision tree, tests on attribute values will determine traversal through the tree, and at the end, the algorithm will finish in one of the leaf nodes that uniquely define the class of a testing example (or with probabilities, if the training set had missing values). If the value for a relevant testing attribute is unknown, the outcome of the test cannot be determined. Then the system explores all possible outcomes from the test and combines the resulting classification arithmetically. Since there can be multiple paths from the root of a tree or subtree to the leaves, a classification is a class distribution rather than a single class. When the total class distribution for the tested case has been established, the class with the highest probability is assigned as the predicted class.

6.4 PRUNING DECISION TREES

Discarding one or more subtrees and replacing them with leaves simplify a decision tree, and that is the main task in decision-tree pruning. In replacing the subtree with a leaf, the algorithm expects to lower the *predicted error rate and* increase the quality of a classification model. But computation of error rate is not simple. An error rate based only on a training data set does not provide a suitable estimate. One possibility to estimate the predicted error rate is to use a new, additional set of test samples if they are available, or to use the cross-validation techniques explained in Chapter 4. This technique divides initially available samples into equal-sized blocks and, for each block, the tree is constructed from all samples except this block and tested with a given block of samples. With the available training and testing samples, the basic idea of decision tree pruning is to remove parts of the tree (subtrees) that do not contribute to the classification accuracy of unseen testing samples, producing a less complex and thus more comprehensible tree. There are two ways in which the recursive-partitioning method can be modified:

1. *Deciding Not to Divide a Set of Samples Any Further under Some Conditions.* The stopping criterion is usually based on some statistical tests, such as the χ^2 test: If there are no significant differences in classification accuracy before and after division, then represent a current node as a leaf. The decision is made in advance, before splitting, and therefore this approach is called *prepruning*.
2. *Removing Retrospectively Some of the Tree Structure Using Selected Accuracy Criteria.* The decision in this process of *postpruning* is made after the tree has been built.

C4.5 follows the *postpruning* approach, but it uses a specific technique to estimate the predicted error rate. This method is called *pessimistic pruning*. For every node in a tree, the estimation of the upper confidence limit U_{cf} is computed using the statistical tables for binomial distribution (given in most textbooks on statistics). Parameter U_{cf} is a function of $|T_i|$ and E for a given node. C4.5 uses the default confidence level of

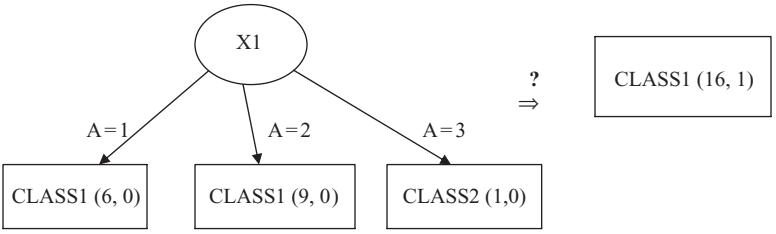

Figure 6.9. Pruning a subtree by replacing it with one leaf node.

25%, and compares $U_{25\%}$ ($|T_i|/E$) for a given node T_i with a weighted confidence of its leaves. Weights are the total number of cases for every leaf. If the predicted error for a root node in a subtree is less than weighted sum of $U_{25\%}$ for the leaves (predicted error for the subtree), then a subtree will be replaced with its root node, which becomes a new leaf in a pruned tree.

Let us illustrate this procedure with one simple example. A subtree of a decision tree is given in Figure 6.9, where the root node is the test x_1 on three possible values {1, 2, 3} of the attribute A. The children of the root node are leaves denoted with corresponding classes and ($|Ti|/E$) parameters. The question is to estimate the possibility of pruning the subtree and replacing it with its root node as a new, generalized leaf node.

To analyze the possibility of replacing the subtree with a leaf node it is necessary to compute a predicted error PE for the initial tree and for a replaced node. Using default confidence of 25%, the upper confidence limits for all nodes are collected from statistical tables: $U_{25\%}(6,0) = 0.206$, $U_{25\%}(9,0) = 0.143$, $U_{25\%}(1,0) = 0.750$, and $U_{25\%}(16,1) = 0.157$. Using these values, the predicted errors for the initial tree and the replaced node are

$$PE_{tree} = 6 \cdot 0.206 + 9 \cdot 0.143 + 1 \cdot 0.750 = 3.257$$

$$PE_{node} = 16 \cdot 0.157 = 2.512$$

Since the existing subtree has a higher value of predicted error than the replaced node, it is recommended that the decision tree be pruned and the subtree replaced with the new leaf node.

6.5 C4.5 ALGORITHM: GENERATING DECISION RULES

Even though the pruned trees are more compact than the originals, they can still be very complex. Large decision trees are difficult to understand because each node has a specific context established by the outcomes of tests at antecedent nodes. To make a decision-tree model more readable, a path to each leaf can be transformed into an IF-THEN production rule. The IF part consists of all tests on a path, and the THEN part is a final classification. Rules in this form are called *decision rules*, and a collection of

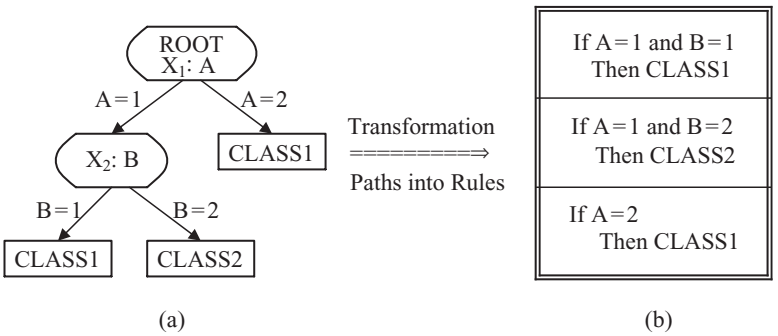

Figure 6.10. Transformation of a decision tree into decision rules. (a) Decision tree; (b) decision rules.

decision rules for all leaf nodes would classify samples exactly as the tree does. As a consequence of their tree origin, the IF parts of the rules would be mutually exclusive and exhaustive, so the order of the rules would not matter. An example of the transformation of a decision tree into a set of decision rules is given in Figure 6.10, where the two given attributes, A and B, may have two possible values, 1 and 2, and the final classification is into one of two classes.

For our trained decision tree in Figure 6.8, the corresponding decision rules will be

If	Attribute1 = A and Attribute2 < = 70
	Then Classification = CLASS1 (2.0/0);
If	Attribute1 = A and Attribute2 > 70
	Then Classification = CLASS2 (3.4/0.4);
If	Attribute1 = B
	Then Classification = CLASS1 (3.2/0);
If	Attribute1 = C and Attribute3 = True
	Then Classification = CLASS2 (2.4/0);
If	Attribute1 = C and Attribute3 = False
	Then Classification = CLASS1 (3.0/0).

Rewriting the tree to a collection of rules, one for each leaf in the tree, would not result in a simplified model. The number of decision rules in the classification model can be extremely large and pruning of rules can improve readability of the model. In some cases, the antecedents of individual rules may contain irrelevant conditions. The rules can be generalized by deleting these superfluous conditions without affecting rule-set accuracy. What are criteria for deletion of rule conditions? Let rule R be

If A then Class C

and a more general rule R' could be

If A' then Class C

where A' is obtained by deleting one condition X from A ($A = A' \cup X$). The evidence for the importance of condition X must be found in the training samples. Each sample in the database that satisfies the condition A' either satisfies or does not satisfy the extended conditions A. Also, each of these cases does or does not belong to the designated Class C. The results can be organized into a contingency 2×2 table:

	Class C	Other Classes
Satisfies condition X	Y_1	E_1
Does not satisfy condition X	Y_2	E_2

There are $Y_1 + E_1$ cases that are covered by the original rule R, where R misclassifies E_1 of them since they belong to classes other than C. Similarly, $Y_1 + Y_2 + E_1 + E_2$ is the total number of cases covered by rule R', and $E_1 + E_2$ are errors. The criterion for the elimination of condition X from the rule is based on a pessimistic estimate of the accuracy of rules R and R'. The estimate of the error rate of rule R can be set to $U_{cf}(Y_1 + E_1, E_1)$, and for that of rule R' to $U_{cf}(Y_1 + Y_2 + E_1 + E_2, E_1 + E_2)$. If the pessimistic error rate of rule R' is no greater than that of the original rule R, then it makes sense to delete condition X. Of course, more than one condition may have to be deleted when a rule is generalized. Rather than looking at all possible subsets of conditions that could be deleted, the C4.5 system performs greedy elimination: At each step, a condition with the lowest pessimistic error is eliminated. As with all greedy searches, there is no guarantee that minimization in every step will lead to a global minimum.

If, for example, the contingency table for a given rule R is given in Table 6.3, then the corresponding error rates are

1. for initially given rule R:

$$U_{cf}(Y_1 + E_1, E_1) = U_{cf}(9, 1) = 0.183, \text{ and}$$

2. for a general rule R' without condition X:

$$U_{cf}(Y_1 + Y_2 + E_1 + E_2, E_1 + E_2) = U_{cf}(16, 1) = 0.157$$

Because the estimated error rate of the rule R' is lower than the estimated error rate for the initial rule R, a rule set pruning could be done by simplifying the decision rule R and replacing it with R'.

TABLE 6.3. Contingency Table for the Rule R

	Class C	Other Classes
Satisfies condition X	***8***	***1***
Does not satisfy condition X	7	0

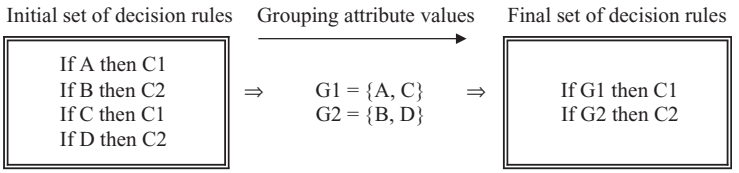

Figure 6.11. Grouping attribute values can reduce decision-rules set.

One complication caused by a rule's generalization is that the rules are no more mutually exclusive and exhaustive. There will be the cases that satisfy the conditions of more than one rule, or of no rules. The conflict resolution schema adopted in C4.5 (detailed explanations have not been given in this book) selects one rule when there is "multiple-rule satisfaction." When no other rule covers a sample, the solution is a *default rule* or a *default class.* One reasonable choice for the default class would be the class that appears most frequently in the training set. C.4.5 uses a modified strategy and simply chooses as the default class the one that contains the most training samples not covered by any rule.

The other possibility of reducing the complexity of decision rules and decision trees is a process of grouping attribute values for categorical data. A large number of values cause a large space of data. There is a concern that useful patterns may not be detectable because of the insufficiency of training data, or that patterns will be detected but the model will be extremely complex.. To reduce the number of attribute values it is necessary to define appropriate groups. The number of possible splitting is large: for n values, there exist $2^{n-1} - 1$ nontrivial binary partitions. Even if the values are ordered, there are $n - 1$ "cut values" for binary splitting. A simple example, which shows the advantages of grouping categorical values in decision-rules reduction, is given in Figure 6.11.

C4.5 increases the number of grouping combinations because it does not include only binary categorical data, but also n-ary partitions. The process is iterative, starting with an initial distribution where every value represents a separate group, and then, for each new iteration, analyzing the possibility of merging the two previous groups into one. Merging is accepted if the information gain ratio (explained earlier) is nondecreasing. A final result may be two or more groups that will simplify the classification model based on decision trees and decision rules.

C4.5 was superseded in 1997 by a commercial system C5.0. The changes include

1. a variant of boosting technique, which constructs an ensemble of classifiers that are then voted to give a final classification, and
2. new data types such as dates, work with "not applicable" values, concept of variable misclassification costs, and mechanisms to pre-filter attributes.

C5.0 greatly improves scalability of both decision trees and rule sets, and enables successful applications with large real-world data sets. In practice, these data can be translated into more complicated decision trees that can include dozens of levels and hundreds of variables.

6.6 CART ALGORITHM & GINI INDEX

CART, as indicated earlier, is an acronym for Classification and Regression Trees. The basic methodology of divide and conquer described in C4.5 is also used in CART. The main differences are in the tree structure, the splitting criteria, the pruning method, and the way missing values are handled.

CART constructs trees that have only binary splits. This restriction simplifies the splitting criterion because there need not be a penalty for multi-way splits. Furthermore, if the label is binary, the binary split restriction allows CART to optimally partition categorical attributes (minimizing any concave splitting criteria) to two subsets of values in the number of attribute values. The restriction has its disadvantages, however, because the tree may be less interpretable with multiple splits occurring on the same attribute at the adjacent levels.

CART uses the *Gini* diversity index as a splitting criterion instead of information-based criteria for C4.5. The CART authors favor the Gini criterion over information gain because the Gini can be extended to include symmetrized costs, and it is computed more rapidly than information gain. The Gini index is used to select the feature at each internal node of the decision tree. We define the Gini index for a data set *S* as follows:

$$\text{Gini}(S) = 1 - \sum_{i=0}^{c-1} p_i^2$$

where

- c is the number of predefined classes,
- C_i are classes for i = 1, . . . , c − 1,
- s_i is the number of samples belonging to class C_i, and
- $p_i = s_i/S$ is a relative frequency of class C_i in the set.

This metric indicates the partition purity of the data set *S*. For branch prediction where we have two classes the Gini index lies within [0, 0.5]. If all the data in *S* belong to the same class, *Gini S* equals the minimum value 0, which means that *S* is pure. If *Gini S* equals 0.5, all observations in *S* are equally distributed among two classes. This decreases as a split favoring one class: for instance a 70/30 distribution produces Gini index of 0.42. If we have more than two classes, the maximum possible value for index increases; for instance, the worst possible diversity for three classes is a 33% split, and it produces a *Gini* value of 0.67. The Gini coefficient, which ranges from 0 to 1 (for extremely large number of classes), is multiplied by 100 to range between 0 and 100 in some commercial tools.

The quality of a split on a feature into *k* subsets S_i is then computed as the weighted sum of the Gini indices of the resulting subsets:

$$Gini_{split} = \sum_{i=0}^{k-1} n_i / n \, \text{Gini}(S_i)$$

where

- n_i is the number of samples in subset S_i after splitting, and
- n is the total number of samples in the given node.

Thus, $Gini_{split}$ is calculated for all possible features, and the feature with minimum $Gini_{split}$ is selected as split point. The procedure is repetitive as in C4.5. We may compare the results of CART and C4.5 attribute selection by applying entropy index (C4.5) and Gini index (CART) for splitting Attribute1 in Table 6.1. While we already have results for C4.5 (gain ratio for Attribute 1), $Gini_{split}$ index for the same attribute may be calculated as:

$$Gini_{split} = \sum_{i=0}^{2} n_i / n \text{Gini}(S_i) = 5/14 \times (1 - (2/5)^2 - (3/5)^2 +$$
$$+ 4/14 \times (1 - (0/4)^2 - (4/4)^2) + 5/14 \times (1 - (2/5)^2 - (3/5)^2) =$$
$$= 0.34$$

Interpretation of the absolute value for $Gini_{split}$ index is not important. It is important that relatively this value is lower for AttributeA1 than for the other two attributes in Table 6.1. The reader may easily check this claim. Therefore, based on Gini index Attribute1 is selected for splitting. It is the same result obtained in C4.5 algorithm using an entropy criterion. Although the results are the same in this example, for many other data sets there could be (usually small) differences in results between these two approaches.

CART also supports the *twoing* splitting criterion, which can be used for multiclass problems. At each node, the classes are separated into two superclasses containing disjoint and mutually exhaustive classes. A splitting criterion for a two-class problem is used to find the attribute, and the two superclasses that optimize the two-class criterion. The approach gives "strategic" splits in the sense that several classes that are similar are grouped together. Although twoing splitting rule allows us to build more balanced trees, this algorithm works slower than Gini rule. For example, if the total number of classes is equal to K, then we will have $2K - 1$ possible grouping into two classes. It can be seen that there is a small difference between trees constructed using Gini and trees constructed via twoing rule. The difference can be seen mainly at the bottom of the tree where the variables are less significant in comparison with the top of the tree.

Pruning technique used in CART is called minimal cost complexity pruning, while C4.5 uses binomial confidence limits. The proposed approach assumes that the bias in the re-substitution error of a tree increases linearly with the number of leaf nodes. The cost assigned to a subtree is the sum of two terms: the re-substitution error and the number of leaves times a complexity parameter α. It can be shown that, for every α value, there exists a unique smallest tree minimizing cost of the tree. Note that, although α runs through a continuum of values, there are at most a finite number of possible subtrees for analysis and pruning.

Unlike C4.5, CART does not penalize the splitting criterion during the tree construction if examples have unknown values for the attribute used in the split. The criterion uses only those instances for which the value is known. CART finds several surrogate splits that can be used instead of the original split. During classification, the first surrogate split based on a known attribute value is used. The surrogates cannot be chosen based on the original splitting criterion because the subtree at each node is constructed based on the original split selected. The surrogate splits are therefore chosen to maximize a measure of predictive association with the original split. This procedure works well if there are attributes that are highly correlated with the chosen attribute.

As its name implies, CART also supports building regression trees. Regression trees are somewhat simpler than classification trees because the growing and pruning criteria used in CART are the same. The regression tree structure is similar to a classification tree, except that each leaf predicts a real number. The re-substitution estimate for pruning the tree is the mean-squared error.

Among the main advantages of CART method is its robustness to outliers and noisy data. Usually the splitting algorithm will isolate outliers in an individual node or nodes. Also, an important practical property of CART is that the structure of its classification or regression trees is invariant with respect to monotone transformations of independent variables. One can replace any variable with its logarithm or square root value, the structure of the tree will not change. One of the disadvantages of CART is that the system may have unstable decision trees. Insignificant modifications of learning samples, such as eliminating several observations, could lead to radical changes in a decision tree: with a significant increase or decrease in tree complexity are changes in splitting variables and values.

C4.5 and CART are two popular algorithms for decision tree induction; however, their corresponding splitting criteria, information gain, and Gini index are considered to be skew-sensitive. In other words, they are not applicable, or at least not successfully applied, in cases where classes are not equally distributed in training and testing data sets. It becomes important to design a decision tree-splitting criterion that captures the divergence in distributions without being dominated by the class priors. One of the proposed solutions is the *Hellinger distance* as a decision tree-splitting criterion. Recent experimental results show that this distance measure is skew-insensitive.

For application as a decision tree-splitting criterion, we assume a countable space, so all continuous features are discretized into p partitions or bins. Assuming a two-class problem (class+ and class−), let X+ be samples belonging to class+, and X− are samples with class−. Then, we are essentially interested in calculating the "distance" in the normalized frequencies distributions aggregated over all the partitions of the two-class distributions X+ and X−. The Hellinger distance between X+ and X− is:

$$d_H(X_+, X_-) = \sqrt{\sum_{j=1}^{P} \left(\sqrt{\frac{|X_{+j}|}{|X_+|}} - \sqrt{\frac{|X_{-j}|}{|X_-|}} \right)^2}$$

This formulation is strongly skew-insensitive. Experiments with real-world data show that the proposed measure may be successfully applied to cases where X+ << X−.

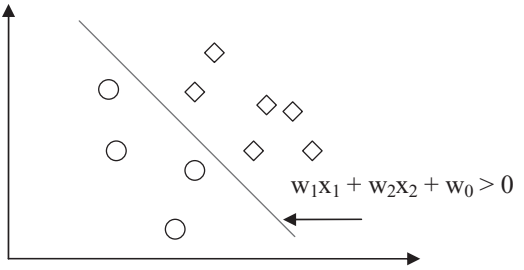

Figure 6.12. Multivariate decision node.

It essentially captures the divergence between the feature value distributions given the two different classes.

Recent developments in the field extend technology toward multivariate trees. In a multivariate tree, at a decision node, all input dimensions can be used for testing (e.g., $w_1x_1 + w_2x_2 + w_0 > 0$ as presented in Fig. 6.12). It is a hyperplane with an arbitrary orientation. This is 2^d (^N_d) possible hyperplanes and exhaustive search is not practical.

With linear multivariate nodes, we can use hyperplanes for better approximation using fewer nodes. A disadvantage of the technique is that multivariate nodes are mode difficult to interpret. Also, more complex nodes require more data. The earliest version of multivariate trees is implemented in CART algorithm, which fine-tunes the weights w_i one by one to decrease impurity. CART also has a preprocessing stage to decrease dimensionality through subset input selection (and therefore reduction of node complexity).

6.7 LIMITATIONS OF DECISION TREES AND DECISION RULES

Decision rule- and decision tree-based models are relatively simple and readable, and their generation is very fast. Unlike many statistical approaches, a logical approach does not depend on assumptions about distribution of attribute values or independence of attributes. Also, this method tends to be more robust across tasks than most other statistical methods. But there are also some disadvantages and limitations of a logical approach, and a data-mining analyst has to be aware of it because the selection of an appropriate methodology is a key step to the success of a data-mining process.

If data samples are represented graphically in an n-dimensional space, where n is the number of attributes, then a logical classifier (decision trees or decision rules) divides the space into regions. Each region is labeled with a corresponding class. An unseen testing sample is then classified by determining the region into which the given point falls. Decision trees are constructed by successive refinement, splitting existing regions into smaller ones that contain highly concentrated points of one class. The number of training cases needed to construct a good classifier is proportional to the number of regions. More complex classifications require more regions that are described

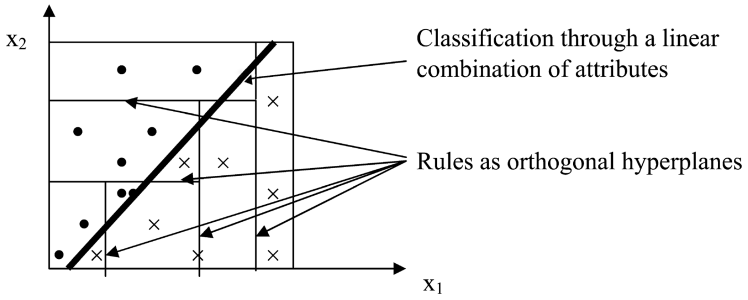

Figure 6.13. Approximation of non-orthogonal classification with hyperrectangles.

with more rules and a tree with higher complexity. All that will require an additional number of training samples to obtain a successful classification.

A graphical representation of decision rules is given by orthogonal hyperplanes in an n-dimensional space. The regions for classification are hyperrectangles in the same space. If the problem at hand is such that the classification hyperplanes are not orthogonal, but are defined through a linear (or nonlinear) combination of attributes, such as the example in Figure 6.13, then that increases the complexity of a rule-based model. A logical approach based on decision rules tries to approximate non-orthogonal, and sometimes, nonlinear classification with hyperrectangles; classification becomes extremely complex with large number of rules and a still larger error.

A possible solution to this problem is an additional iteration of the data-mining process: Returning to the beginning of preprocessing phases, it is necessary to transform input features into new dimensions that are linear (or nonlinear) combinations of initial inputs. This transformation is based on some domain heuristics, and it requires emphasis with additional effort in data preparation; the reward is a simpler classification model with a lower error rate.

The other types of classification problems, where decision rules are not the appropriate tool for modeling, have classification criteria in the form: A given class is supported if n out of m conditions are present. To represent this classifier with rules, it would be necessary to define $\binom{m}{n}$ regions only for one class. Medical diagnostic decisions are a typical example of this kind of classification. If four out of 11 symptoms support diagnosis of a given disease, then the corresponding classifier will generate 330 regions in an 11-dimensional space for positive diagnosis only. That corresponds to 330 decision rules. Therefore, a data-mining analyst has to be very careful in applying the orthogonal-classification methodology of decision rules for this type of nonlinear problems.

Finally, introducing new attributes rather than removing old ones can avoid the sometimes intensive fragmentation of the n-dimensional space by additional rules. Let us analyze a simple example. A classification problem is described by nine binary inputs $\{A_1, A_2, \ldots, A_9\}$, and the output class C is specified by the logical relation

$$(A_1 \vee A_2 \vee A_3) \wedge (A_4 \vee A_5 \vee A_6) \wedge (A_7 \vee A_8 \vee A_9) \rightarrow C$$

The above expression can be rewritten in a conjunctive form:

$$((A_1 \wedge A_4 \wedge A_7) \vee (A_1 \wedge A_5 \wedge A_7) \vee \ldots) \rightarrow C$$

and it will have 27 factors with only $\wedge$ operations. Every one of these factors is a region in a 9-D space and corresponds to one rule. Taking into account regions for negative examples, there exist about 50 leaves in the decision tree (and the same number of rules) describing class C. If new attributes are introduced:

$$B_1 = A_1 \vee A_2 \vee A_3,$$

$$B_2 = A_4 \vee A_5 \vee A_6, \text{ and}$$

$$B_3 = A_7 \vee A_8 \vee A_9$$

the description of class C will be simplified into the logical rule

$$B_1 \wedge B_2 \wedge B_3 \rightarrow C$$

It is possible to specify the correct classification using a decision tree with only four leaves. In a new three-dimensional space (B_1, B_2, B_3) there will be only four decision regions. This kind of simplification via constructive induction (development of new attributes in the preprocessing phase) can be applied also in a case n-of-m attributes' decision. If none of the previous transformations are found appropriate, the only way to deal with the increased fragmentation of an n-dimensional space is to bring more data to bear on the problem.

6.8 REVIEW QUESTIONS AND PROBLEMS

1. Explain the differences between the statistical and logical approaches in the construction of a classification model.
2. What are the new features of C4.5 algorithm comparing with original Quinlan's ID3 algorithm for decision tree generation?
3. Given a data set X with 3-D categorical samples:

X:	Attribute1	Attribute2	Class
	T	1	C2
	T	2	C1
	F	1	C2
	F	2	C2

Construct a decision tree using the computation steps given in the C4.5 algorithm.

4. Given a training data set Y:

Y	A	B	C	Class
	15	1	A	C_1
	20	3	B	C_2
	25	2	A	C_1
	30	4	A	C_1
	35	2	B	C_2
	25	4	A	C_1
	15	2	B	C_2
	20	3	B	C_2

Find the best threshold (for the maximal gain) for AttributeA.

(a) Find the best threshold (for the maximal gain) for AttributeB.
(b) Find a decision tree for data set Y.
(c) If the testing set is:

A	B	C	Class
10	2	A	C_2
20	1	B	C_1
30	3	A	C_2
40	2	B	C_2
15	1	B	C_1

What is the percentage of correct classifications using the decision tree developed in (c).
(d) Derive decision rules from the decision tree.

5. Use the C4.5 algorithm to build a decision tree for classifying the following objects:

Class	Size	Color	Shape
A	Small	Yellow	Round
A	Big	Yellow	Round
A	Big	Red	Round
A	Small	Red	Round
B	Small	Black	Round
B	Big	Black	Cube
B	Big	Yellow	Cube
B	Big	Black	Round
B	Small	Yellow	Cube

6. Given a training data set Y* with missing values:

Y*:	A	B	C	Class
	15	1	A	C1
	20	3	B	C2
	25	2	A	C1
	–	4	A	C1
	35	2	–	C2
	25	4	A	C1
	15	2	B	C2
	20	3	B	C2

(a) Apply a modified C4.5 algorithm to construct a decision tree with the (T_i/E) parameters explained in Section 7.3.
(b) Analyze the possibility of pruning the decision tree obtained in (a).
(c) Generate decision rules for the solution in (a). Is it necessary to generate a default rule for this rule-based model?

7. Why is postpruning in C4.5 defined as pessimistic pruning?

8. Suppose that two decision rules are generated with C4.5:

Rule1: $(X > 3) \wedge (Y \geq 2) \rightarrow$ Class1 (9.6/0.4)
Rule2: $(X > 3) \wedge (Y < 2) \rightarrow$ Class2 (2.4/2.0).

Analyze if it is possible to generalize these rules into one using confidence limit $U_{25\%}$ for the binomial distribution.

9. Discuss the complexity of the algorithm for optimal splitting of numeric attributes into more than two intervals.

10. In real-world data-mining applications, a final model consists of extremely large number of decision rules. Discuss the potential actions and analyses you should perform to reduce the complexity of the model.

11. Search the Web to find the basic characteristics of publicly available or commercial software tools for generating decision rules and decision trees. Document the results of your search.

12. Consider a binary classification problem (output attribute value = {Low, High}) with the following set of input attributes and attribute values:

- Air Conditioner = {Working, Broken}
- Engine = {Good, Bad}
- Mileage = {High, Medium, Low}
- Rust = {Yes, No}

Suppose a rule-based classifier produces the following rule set:

Mileage = High $\longrightarrow$ *Value = Low*

Mileage = Low $\longrightarrow$ *Value = High*

Air Conditioner = Working and Engine = Good $\longrightarrow$ *Value = High*

Air Conditioner = Working and Engine = Bad $\longrightarrow$ *Value = Low*

Air Conditioner = Broken $\longrightarrow$ *Value = Low*

(a) Are the rules mutually exclusive? Explain your answer.
(b) Is the rule set exhaustive (covering each possible case)? Explain your answer.
(c) Is ordering needed for this set of rules? Explain your answer.
(d) Do you need a default class for the rule set? Explain your answer.

13. Of the following algorithms:

- C4.5
- K-Nearest Neighbor
- Naïve Bayes
- Linear Regression

(a) Which are fast in training but slow in classification?
(b) Which one produces classification rules?
(c) Which one requires discretization of continuous attributes before application?
(d) Which model is the most complex?

14. (a) How much information is involved in choosing one of eight items, assuming that they have an equal frequency?
(b) One of 16 items?

15. The following data set will be used to learn a decision tree for predicting whether a mushroom is edible or not based on its shape, color, and odor.

Shape	Color	Odor	Edible
C	B	1	Yes
D	B	1	Yes
D	W	1	Yes
D	W	2	Yes
C	B	2	Yes
D	B	2	No
D	G	2	No
C	U	2	No
C	B	3	No
C	W	3	No
D	W	3	No

(a) What is entropy H(Edible|Odor = 1 or Odor = 3)?
(b) Which attribute would the C4.5 algorithm choose to use for the root of the tree?
(c) Draw the full decision tree that would be learned for this data (no pruning).
(d) Suppose we have a validation set as follows. What will be the training set error and validation set error of the tree? Express your answer as the number of examples that would be misclassified.

Shape	Color	Odor	Edible
C	B	2	No
D	B	2	No
C	W	2	Yes

6.9 REFERENCES FOR FURTHER STUDY

Dzeroski, S., N. Lavrac, eds., *Relational Data Mining*, Springer-Verlag, Berlin, Germany, 2001.

Relational data mining has its roots in inductive logic programming, an area in the intersection of machine learning and programming languages. The book provides a thorough overview of different techniques and strategies used in knowledge discovery from multi-relational data. The chapters describe a broad selection of practical, inductive-logic programming approaches to relational data mining, and give a good overview of several interesting applications.

Kralj Novak, P., N. Lavrac, G. L. Webb, Supervised Descriptive Rule Discovery: A Unifying Survey of Contrast Set, Emerging Pattern and Subgroup Mining, *Journal of Machine Learning Research*, Vol. 10, 2009, p. 377–403.

This paper gives a survey of contrast set mining (CSM), emerging pattern mining (EPM), and subgroup discovery (SD) in a unifying framework named *supervised descriptive rule discovery*. While all these research areas aim at discovering patterns in the form of rules induced from labeled data, they use different terminology and task definitions, claim to have different goals, claim to use different rule learning heuristics, and use different means for selecting subsets of induced patterns. This paper contributes a novel understanding of these subareas of data mining by presenting a unified terminology, by explaining the apparent differences between the learning tasks as variants of a unique supervised descriptive rule discovery task and by exploring the apparent differences between the approaches.

Mitchell, T., *Machine Learning*, McGraw Hill, New York, NY, 1997.

This is one of the most comprehensive books on machine learning. Systematic explanations of all methods and a large number of examples for all topics are the main strengths of the book. Inductive machine-learning techniques are only a part of the book, but for a deeper understanding of current data-mining technology and newly developed techniques, it is very useful to get a global overview of all approaches in machine learning.

Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1992.

The book outlines the C4.5 algorithm step by step, with detailed explanations and many illustrative examples. The second part of the book is taken up by the source listing of the C program that makes up the C4.5 system. The explanations in the book are intended to give a broad-brush view of C4.5 inductive learning with many small heuristics, leaving the detailed discussion to the code itself.

Russell, S., P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, NJ, 1995.

The book gives a unified presentation of the artificial intelligence field using an agent-based approach. Equal emphasis is given to theory and practice. An understanding of the basic concepts in artificial intelligence (AI), including an approach to inductive machine learning, is obtained through layered explanations and agent-based implementations of algorithms.