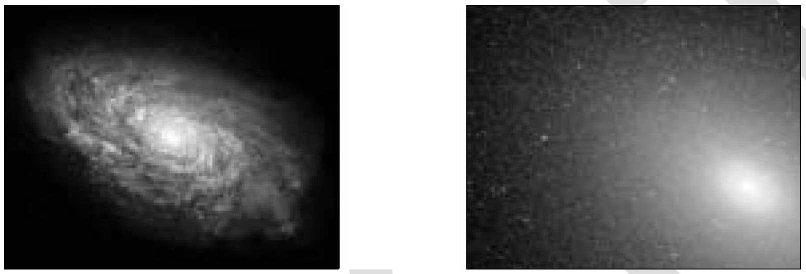
UNIT-4

Classification analysis

Classification, which is the task of assigning objects to one of several predefined categories, is a pervasive problem that encompasses many diverse applications. Examples include detecting spam email messages based upon the message header and content, categorizing cells as malignant or benign based upon the results of MRI scans, and classifying galaxies based upon their shapes .



(a) A spiral galaxy. (b) An elliptical galaxy.

Figure 4.1. Classification of galaxies. The images are from the NASA website.

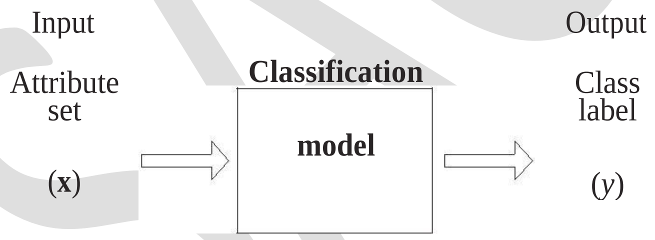


Figure 4.2. Classification as the task of mapping an input attribute set x into its class label y.

This chapter introduces the basic concepts of classification, describes some of the key issues such as model overfitting, and presents methods for evaluating and comparing the performance of a classification technique. While it focuses mainly on a technique known as decision tree induction, most of the discussion in this chapter is also applicable to other classification techniques, many of which are covered in Chapter 5.

4.1 Preliminaries

The input data for a classification task is a collection of records. Each record, also known as an instance or example, is characterized by a tuple (x, y), where x is the attribute set and y is a special attribute, designated as the class label (also known as category or target attribute). Table

4.1 shows a sample data set used for classifying vertebrates into one of the following categories: mammal, bird, fish, reptile, or amphibian. The attribute set includes properties of a vertebrate such as its body temperature, skin cover, method of reproduction, ability to fly, and ability to live in water. Although the attributes presented in Table 4.1 are mostly discrete, the attribute set can also contain continuous features. The class label, on the other hand, must be a discrete attribute. This is a key characteristic that distinguishes classification from regression, a predictive modeling task in which y is a continuous attribute. Regression techniques are covered in Appendix D.

Definition 4.1 (Classification). Classification is the task of learning a tar-get function f that maps each attribute set x to one of the predefined class labels y.

The target function is also known informally as a classification model. A classification model is useful for the following purposes.

Descriptive Modeling A classification model can serve as an explanatory tool to distinguish between objects of different classes. For example, it would be useful—for both biologists and others—to have a descriptive model that

Table 4.1. 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	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

summarizes the data shown in Table 4.1 and explains what features define a vertebrate as a mammal, reptile, bird, fish, or amphibian.

Predictive Modeling A classification model can also be used to predict the class label of unknown records. As shown in Figure 4.2, a classification model can be treated as a black box that automatically assigns a class label when presented with the attribute set of an unknown record. Suppose we are given the following characteristics of a creature known as a gila monster:

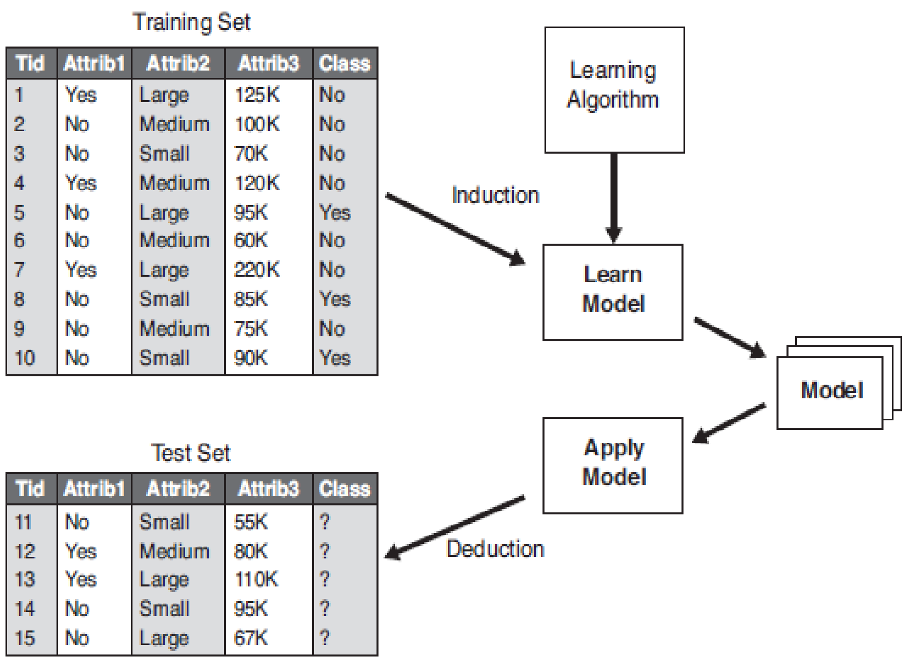
Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

We can use a classification model built from the data set shown in Table 4.1 to determine the class to which the creature belongs.

Classification techniques are most suited for predicting or describing data sets with binary or nominal categories. They are less effective for ordinal categories (e.g., to classify a person as a member of high-, medium-, or low-income group) because they do not consider the implicit order among the categories. Other forms of relationships, General Approach to Solving a Classification Problem.

A **classification technique** (or classifier) is a systematic approach to building classification models from an input data set. Examples include decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and na¨ıve Bayes classifiers. Each technique employs a learning algorithm to identify a model that best fits the relationship between the attribute set and class label of the input data.

Figure 4.3. General approach for building a classification model.



Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Test Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Table 4.2. Confusion matrix for a 2-class problem.

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

be provided. The training set is used to build a classification model, which is subsequently applied to the test set, which consists of records with unknown class labels.

Evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model. These counts are tabulated in a table known as a confusion matrix. Table 4.2 depicts the confusion matrix for a binary classification problem. Each entry f_{ij} in this table denotes the number of records from class i predicted to be of class j. For instance, f_{01} is the number of records from class 0 incorrectly predicted as class 1. Based on the entries in the confusion matrix, the total number of correct predictions made by the model is $(f_{11} + f_{00})$ and the total number of incorrect predictions is $(f_{10} + f_{01})$.

This can be done using a performance metric such as accuracy, which is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (4.1)$$

Equivalently, the performance of a model can be expressed in terms of its error rate, which is given by the following equation:

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (4.2)$$

Most classification algorithms seek models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set

4.3 Decision Tree Induction

This section introduces a decision tree classifier, which is a simple yet widely used classification technique.

4.3.1 How a Decision Tree Works

To illustrate how classification with a decision tree works, consider a simpler version of the vertebrate classification problem described in the previous sec-tion. Instead of classifying the vertebrates into five distinct groups of species, we assign them to two categories: mammals and non-mammals.

Suppose a new species is discovered by scientists. How can we tell whether it is a mammal or a non-mammal? One approach is to pose a series of questions about the characteristics of the species. The first question we may ask is whether the species is cold- or warm-blooded. If it is cold-blooded, then it is definitely not a mammal. Otherwise, it is either a bird or a mammal. In the latter case, we need to ask a follow-up question:

.

- Internal nodes, each of which has exactly one incoming edge and two or more outgoing edges.

- Leaf or terminal nodes, each of which has exactly one incoming edge and no outgoing edges.

In a decision tree, each leaf node is assigned a class label. The non-terminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteris-tics. For example, the root node shown in Figure 4.4 uses the attribute Body

4.3 Decision Tree Induction 151

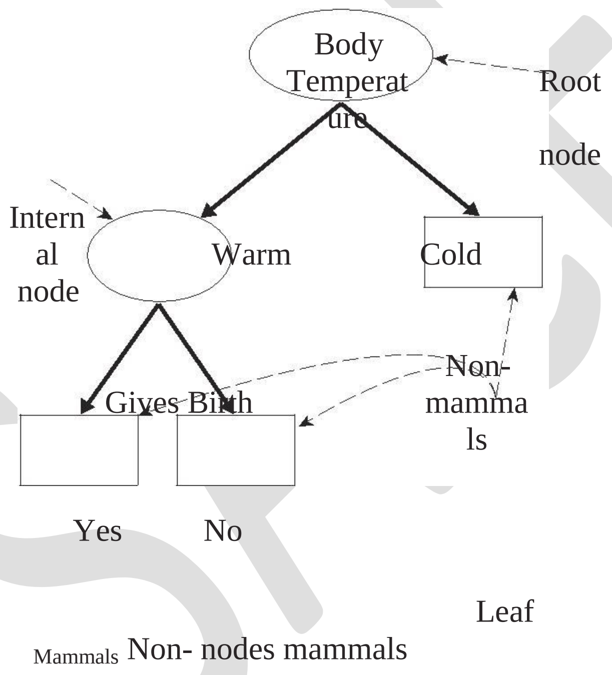


Figure 4.4. A decision tree for the mammal classification problem.

Temperature to separate warm-blooded from cold-blooded vertebrates. Since all cold-blooded vertebrates are non-mammals, a leaf node labeled Non-mammals is created as the right child of the root node. If the vertebrate is warm-blooded, a subsequent attribute, Gives Birth, is used to distinguish mammals from other warm-blooded creatures, which are mostly birds.

Classifying a test record is straightforward once a decision tree has been constructed. Starting from the root node, we apply the test condition to the record and follow the appropriate branch based on the outcome of the test. This will lead us either to another internal node, for which a new test condition is applied, or to a leaf node. The class label associated with the leaf node is then assigned to the record. As an illustration, Figure 4.5 traces the path in the decision tree that is used to predict the class label of a flamingo. The path terminates at a leaf node labeled Non-mammals.

4.3.2 How to Build a Decision Tree

In principle, there are exponentially many decision trees that can be con-structed from a given set of attributes. While some of the trees are more accu-rate than others, finding the optimal tree is computationally infeasible because of the exponential size of the search space. Nevertheless, efficient algorithms have been developed to induce a reasonably accurate, albeit suboptimal, de-cision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy that grows a decision tree by making a series of locally op-

152 Chapter 4 Classification

Unlabeled data	**Name**	**Body temperature**	**Gives Birth**	**...**	**Class**
	Flamingo	Warm	No	...	?

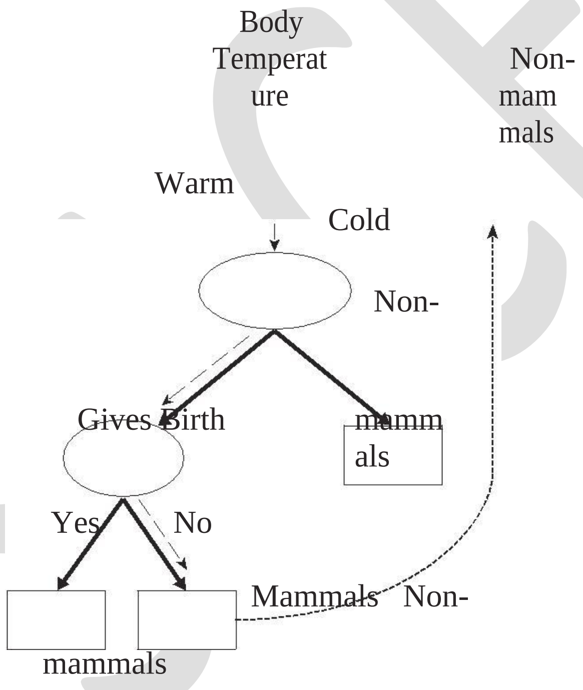


Figure 4.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the Non-mammal class.

Hunt's Algorithm

In Hunt's algorithm, a decision tree is grown in a recursive fashion by parti-tioning the training records into successively purer subsets. Let D_t be the set of training records that are associated with node t and y = $\{y_1, y_2, . . . , y_c\}$ be the class labels. The following is a recursive definition of

Hunt's algorithm.

Step 1: If all the records in D_t belong to the same class y_t, then t is a leaf node labeled as y_t.

Step 2: If D_t contains records that belong to more than one class, an at-tribute test condition is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condi-tion and the records in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

	binary	**categorical**	**continuous**	**class**
Tid	**Home Owner**	**Marital Status**	**Annual Income**	**Defaulted Borrower**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Figure 4.6. Training set for predicting borrowers who will default on loan payments.

The initial tree for the classification problem contains a single node with class label Defaulted = No (see Figure 4.7(a)), which means that most of the borrowers successfully repaid their loans. The tree, however, needs to be refined since the root node contains records from both classes. The records are subsequently divided into smaller subsets based on the outcomes of the Home Owner test condition,

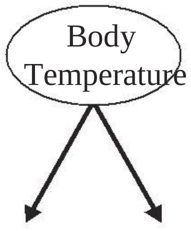
Design Issues of Decision Tree Induction

A learning algorithm for inducing decision trees must address the following two issues.

1. How should the training records be split? Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition.
2. How should the splitting procedure stop? A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records belong to the same class or all the records have identical attribute values. Although both conditions are sufficient to stop any decision tree induction algorithm, other criteria can be imposed to allow the tree-growing procedure to terminate earlier. The advantages of early termination will be discussed later in Section 4.4.5.

4.3.3 Methods for Expressing Attribute Test Conditions

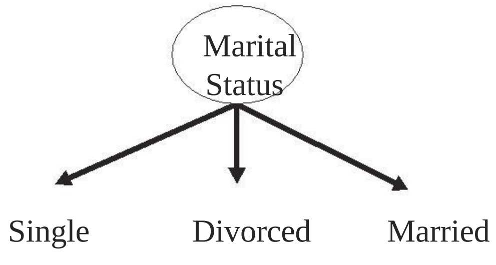
Decision tree induction algorithms must provide a method for expressing an attribute test condition and its corresponding outcomes for different attribute types.Binary Attributes The test condition for a binary attribute generates two potential outcomes,



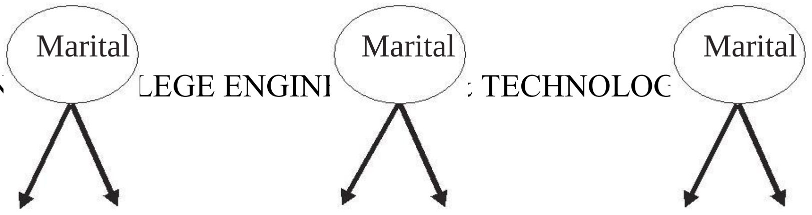
Warm- Cold-blooded blooded

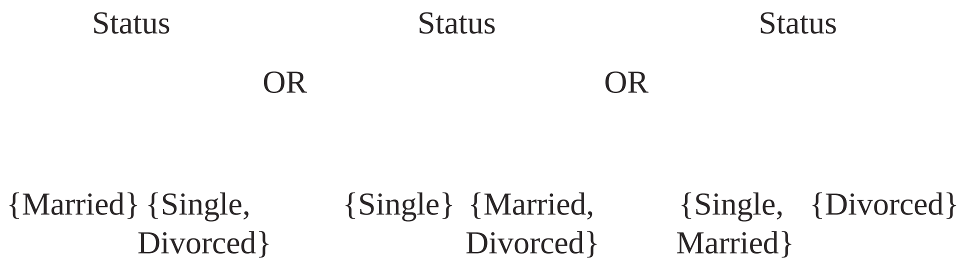
Figure 4.8. Test con

Classification for binary attributes.



(a) Multiway split

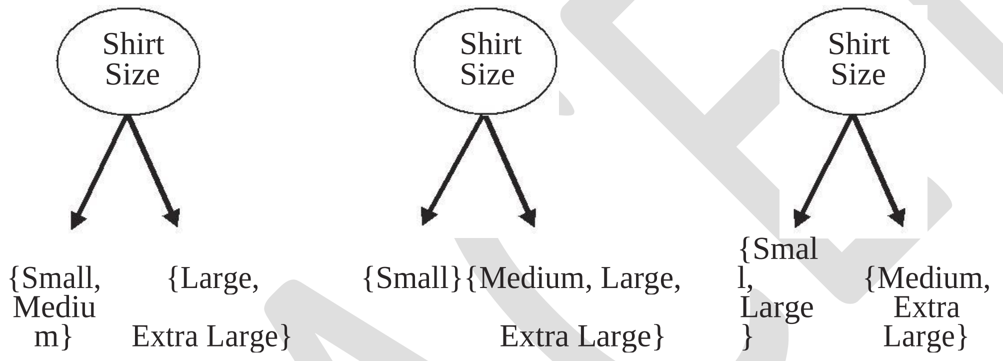




(b) Binary split {by grouping attribute values}

Figure 4.9. Test conditions for nominal attributes.

Nominal Attributes Since a nominal attribute can have many values, its test condition can be expressed in two ways, as shown in



(a) (b) (c)

Continuous Attributes For continuous attributes, the test condition can be expressed as a comparison test ($A < v$) or ($A \geq v$) with binary outcomes, or a range query with outcomes of the form $v_i \leq A < v_{i+1}$, for $i = 1, \ldots, k$. The difference between these approaches is shown in Figure 4.11. For the binary case, the decision tree algorithm must consider all possible split positions v, and it selects the one that produces the best partition. For the multiway split, the algorithm must consider all possible ranges of continuous values. One approach is to apply the discretization strategies described in Section 2.3.6 on page 57. After discretization, a new ordinal value will be assigned to each discretized interval. Adjacent intervals can also be aggregated into wider ranges as long as the order property is preserved.

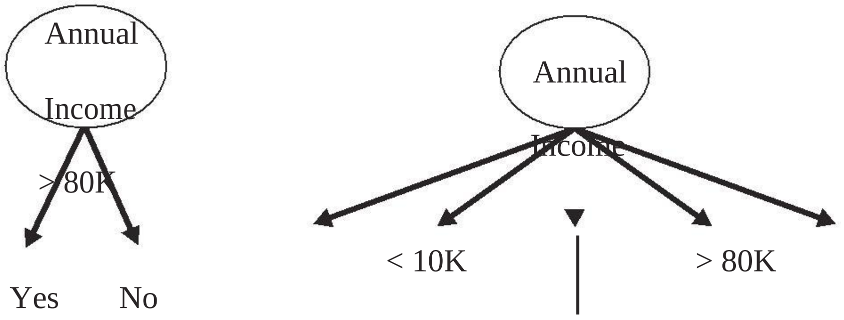


Figure 4.11. Test condition for continuous attributes.

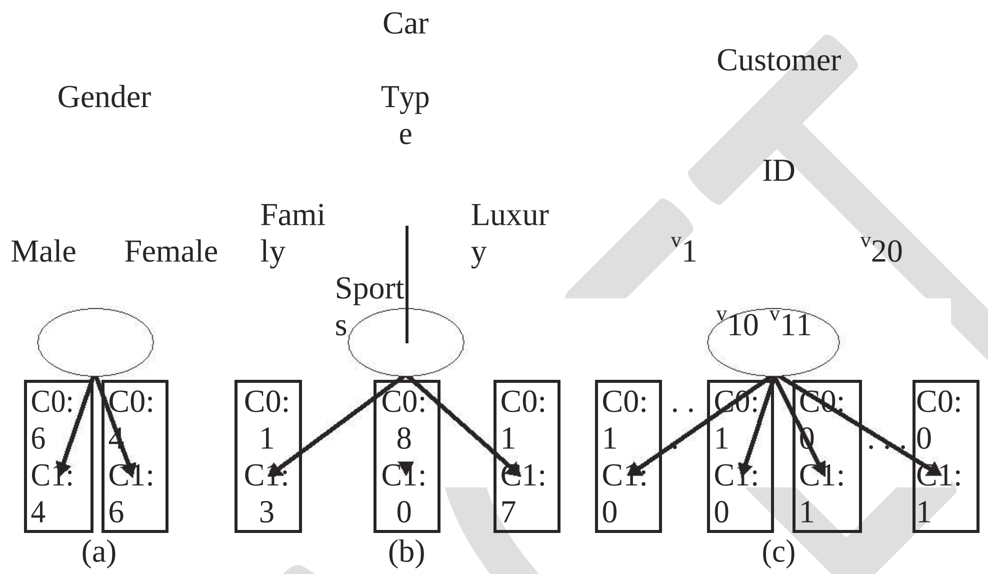


Figure 4.12. Multiway versus binary splits.

4.3.4 Measures for Selecting the Best Split

There are many measures that can be used to determine the best way to split the records. These measures are defined in terms of the class distribution of the records before and after splitting.

Let p(i|t) denote the fraction of records belonging to class i at a given node t. We sometimes omit the reference to node t and express the fraction as p_i. In a two-class problem, the class distribution at any node can be written as (p_0, p_1), where $p_1 = 1 - p_0$. To illustrate,

The measures developed for selecting the best split are often based on the degree of impurity of the child nodes. The smaller the degree of impurity, the more skewed the class distribution.

$$\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t), \qquad (4.3)$$

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2, \qquad (4.4)$$

$$\text{Classification error}(t) = 1 - \max_i [p(i|t)], \qquad (4.5)$$

where c is the number of classes and 0 $\log_2$ 0 = 0 in entropy calculations.

Figure 4.13. Comparison among the impurity measures for binary classification problems.

Figure 4.13 compares the values of the impurity measures for binary classi-fication problems. p refers to the fraction of records that belong to one of the two classes. Observe that all three measures attain their maximum value when the class distribution is uniform (i.e., when p = 0.5). The minimum values for the measures are attained when all the records belong to the same class (i.e., when p equals 0 or 1). We next provide several examples of computing the diff erent impurity measures.

Node N_1	Count
Class=0	0
Class=1	6

Gini = $1 - (0/6)^2 - (6/6)^2 = 0$

Entropy = $-(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0$

Error = $1 - \max[0/6, 6/6] = 0$

Node N_2	Count
Class=0	1
Class=1	5

Gini = $1 - (1/6)^2 - (5/6)^2 = 0.278$

Entropy = $-(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650$

Error = $1 - \max[1/6, 5/6] = 0.167$

Node N_3	Count
Class=0	3
Class=1	3

Gini = $1 - (3/6)^2 - (3/6)^2 = 0.5$

Entropy = $-(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1$

Error = $1 - \max[3/6, 3/6] = 0.5$

Splitting of Binary Attributes

Consider the diagram shown in Figure 4.14. Suppose there are two ways to split the data into smaller subsets. Before splitting, the Gini index is 0.5 since there are an equal number of records from both classes. If attribute A is chosen to split the data, the Gini index for node N1 is 0.4898, and for node N2, it is 0.480. The weighted average of the Gini index for the descendent nodes is (7/12) × 0.4898 + (5/12) × 0.480 = 0.486. Similarly, we can show that the weighted average of the Gini index for attribute B is 0.375. Since the subsets for attribute B have a smaller Gini index, it is preferred over attribute A.

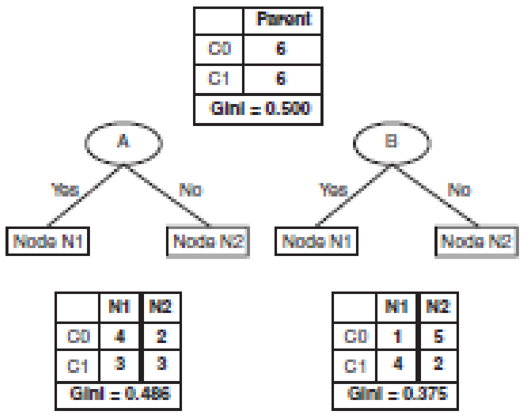


Figure 4.14. Splitting binary attributes.

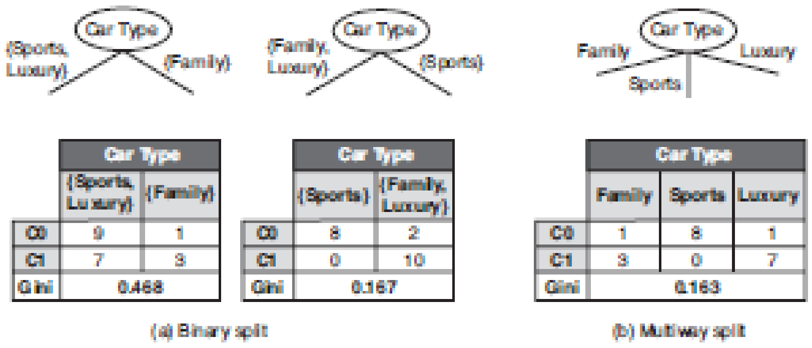


Figure 4.15. Splitting nominal attributes.

Luxury} is 0.4922 and the Gini index of {Family} is 0.3750. The weighted average Gini index for the grouping is equal to

$$16/20 \times 0.4922 + 4/20 \times 0.3750 = 0.468.$$

Similarly, for the second binary grouping of {Sports} and {Family, Luxury}, the weighted average Gini index is 0.167. The second grouping has a lower Gini index because its corresponding subsets are much purer.

Gain Ratio

Impurity measures such as entropy and Gini index tend to favor attributes that have a large number of distinct values. However, if we compare both conditions with Customer ID, the latter appears to produce purer partitions. Yet Customer ID is not a predictive attribute because its value is unique for each record. Even in a less extreme situation, a test condition that results in a large number of outcomes may not be desirable because the number of records associated with each partition is too small to enable us to make any reliable predictions.

There are two strategies for overcoming this problem. The first strategy is to restrict the test conditions to binary splits only. This strategy is employed by decision tree algorithms such as CART. Another strategy is to modify the splitting criterion to take into account the number of outcomes produced by the attribute test condition. For example, in the C4.5 decision tree algorithm, a splitting criterion known as gain ratio is used to determine the goodness of a split. This criterion is defined as follows:

$$\text{Gain ratio} = \frac{\Delta_{\text{info}}}{\text{Split Info}}. \qquad (4.7)$$

Here, Split Info $= -\sum_{i=1}^{k} P(v_i) \log_2 P(v_i)$ and k is the total number of splits. For example, if each attribute value has the same number of records, then $\forall i : P(v_i) = 1/k$ and the split information would be equal to $\log_2 k$. This example suggests that if an attribute produces a large number of splits, its split information will also be large, which in turn reduces its gain ratio.

4.3.5 Algorithm for Decision Tree Induction :A skeleton decision tree induction algorithm called TreeGrowth is shown in Algorithm 4.1. The input to this algorithm consists of the training records E and the attribute set F. The algorithm works by recursively selecting the best attribute to split the data (Step 7) and expanding the leaf nodes of the

Algorithm 4.1 A skeleton decision tree induction algorithm.

```
TreeGrowth (E, F )
 1: if stopping cond(E,F ) = true then
 2:    leaf = createNode().
 3:    leaf.label = Classify(E).
 4:    return leaf .
 5: else
 6:    root = createNode().
 7:    root.test cond = find best split(E, F ).
 8:    let V = {v|v is a possible outcome of root.test cond }.
 9:    for each v ∈ V do
10:       $E_v$ = {e | root.test cond(e) = v and e ∈ E}.
11:       child = TreeGrowth($E_v$ , F ).
12:       add child as descendent of root and label the edge (root → child) as v.
13:    end for
```

```
14: end if
15: return root.
```

4.3.7 Characteristics of Decision Tree Induction

The following is a summary of the important characteristics of decision tree induction algorithms.

1. Decision tree induction is a nonparametric approach for building classifi-cation models. In other words, it does not require any prior assumptions regarding the type of probability distributions satisfied by the class and other attributes (unlike some of the techniques described in Chapter 5).

2. Finding an optimal decision tree is an NP-complete problem. Many de-cision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space. For example, the algorithm pre-sented in Section 4.3.5 uses a greedy, top-down, recursive partitioning strategy for growing a decision tree.

3. Techniques developed for constructing decision trees are computationally inexpensive, making it possible to quickly construct models even when the training set size is very large. Furthermore, once a decision tree has been built, classifying a test record is extremely fast, with a worst-case complexity of O(w), where w is the maximum depth of the tree.

4. Decision trees, especially smaller-sized trees, are relatively easy to inter-pret. The accuracies of the trees are also comparable to other classifica-tion techniques for many simple data sets.

5. Decision trees provide an expressive representation for learning discrete-valued functions. However, they do not generalize well to certain types of Boolean problems. One notable example is the parity function, whose value is 0 (1) when there is an odd (even) number of Boolean attributes
   with the value T rue. Accurate modeling of such a function requires a full decision tree with 2^d nodes, where d is the number of Boolean attributes (see Exercise 1 on page 198).

6. Decision tree algorithms are quite robust to the presence of noise, espe-cially when methods for avoiding overfitting, as described in Section 4.4, are employed.

7. The presence of redundant attributes does not adversely aff ect the ac-curacy of decision trees. An attribute is redundant if it is strongly cor-related with another attribute in the data. One of the two redundant attributes will not be used for splitting once the other attribute has been chosen. However, if the data set contains many irrelevant attributes, i.e., attributes that are not useful for the classification task, then some of the irrelevant attributes may be accidently chosen during the tree-growing process, which results in a decision tree that is larger than necessary. Feature selection techniques can help to improve the accuracy of deci-sion trees by eliminating the irrelevant attributes during preprocessing. We will investigate the issue of too many irrelevant attributes in Section 4.4.3.

8. Since most decision tree algorithms employ a top-down, recursive parti-tioning approach, the number of records becomes smaller as we traverse down the tree. At the leaf nodes, the number of records may be too small to make a statistically significant decision about the class rep-resentation of the nodes. This is known as the data fragmentation problem. One possible solution is to disallow further splitting when the number of records falls below a certain threshold.

9. A subtree can be replicated multiple times in a decision tree, as illus-trated in Figure 4.19.

This makes the decision tree more complex than necessary and perhaps more difficult to interpret. Such a situation can arise from decision tree implementations that rely on a single attribute test condition at each internal node. Since most of the decision tree al-gorithms use a divide-and-conquer partitioning strategy, the same test condition can be applied to diff erent parts of the attribute space, thus leading to the subtree replication problem.

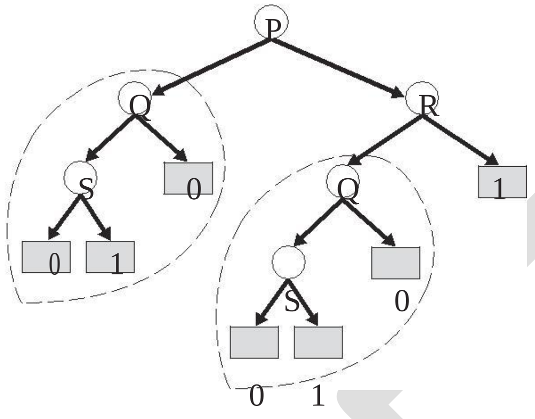


Figure 4.19. Tree replication problem. The same subtree can appear at different branches.

10. The test conditions described so far in this chapter involve using only a single attribute at a time. As a consequence, the tree-growing procedure can be viewed as the process of partitioning the attribute space into disjoint regions until each region contains records of the same class (see Figure 4.20). The border between two neighboring regions of diff erent classes is known as a decision boundary. Since the test condition in-volves only a single attribute, the decision boundaries are rectilinear; i.e

4.4 Model Over fitting

The errors committed by a classification model are generally divided into two types: training errors and generalization errors. Training error, also known as reconstitution error or apparent error, is the number of misclassification errors committed on training records, whereas generalization error is the expected error of the model on previously unseen records. Recall from Section 4.2 that a good classification model must not only fit the training data well, it must also accurately classify records it has never seen before. In other words, a good model must have low training error as well as low generalization error. This is important because a model that fits the training data too well can have a poorer generalization error than a model with a higher training error. Such a situation is known as model over fitting.

4.4.4 Estimation of Generalization Errors

Although the primary reason for overfitting is still a subject of debate, it is generally agreed that the complexity of a model has an impact on model overfitting, as was illustrated in Figure 4.23. The question is, how do we

determine the right model complexity? The ideal complexity is that of a model that produces the lowest generalization error. The problem is that the learning algorithm has access only to the training set during model building (see Figure 4.3). It has no knowledge of the test set, and thus, does not know how well the tree will perform on records it has never seen before. The best it can do is to estimate the generalization error of the induced tree. This section presents several methods for doing the estimation. Using Reconstitution Estimate

The resubstitution estimate approach assumes that the training set is a good representation of the overall data. Consequently, the training error, otherwise known as resubstitution error, can be used to provide an optimistic estimate for the generalization error. Under this assumption, a decision tree induction algorithm simply selects the model that produces the lowest training error rate as its final model. However, the training error is usually a poor estimate of generalization error.

Example 4.1. Consider the binary decision trees shown in Figure 4.27. As-sume that both trees are generated from the same training data and both make their classification decisions at each leaf node according to the majority class. Note that the left tree, T_L, is more complex because it expands some of the leaf nodes in the right tree, T_R. The training error rate for the left tree is $e(T_L) = 4/24 = 0.167$, while the training error rate for the right tree is

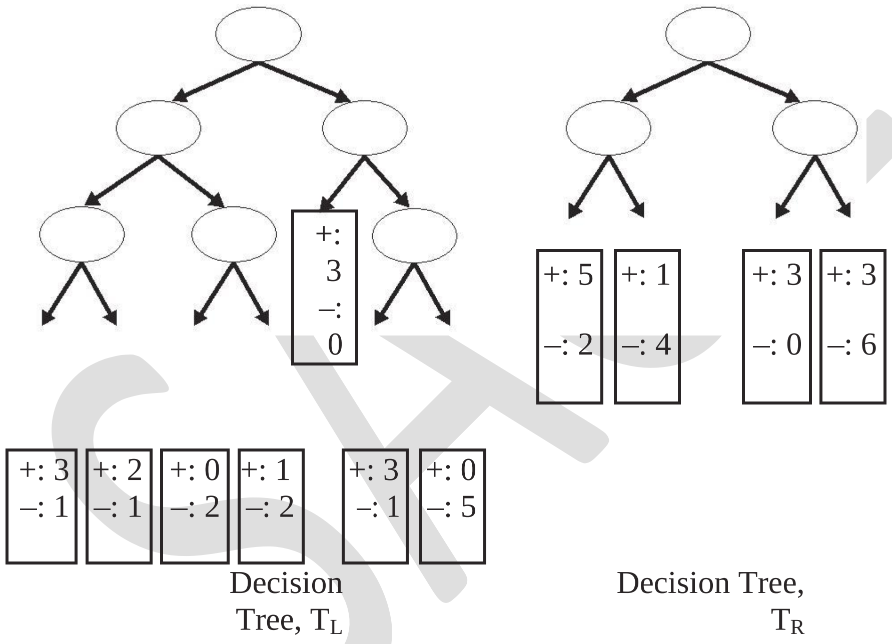


Figure 4.27. Example of two decision trees generated from the same training data.

$e(T_R) = 6/24 = 0.25$. Based on their resubstitution estimate, the left tree is considered better than the right tree. ■

Incorporating Model Complexity

As previously noted, the chance for model overfitting increases as the model becomes more complex. For this reason, we should prefer simpler models, a strategy that agrees with a well-known principle known as Occam's razor or the principle of parsimony:

Definition 4.2. Occam’s Razor: Given two models with the same general-ization errors, the simpler model is preferred over the more complex model.

Occam’s razor is intuitive because the additional components in a complex model stand a greater chance of being fitted purely by chance. In the words of Einstein, “Everything should be made as simple as possible, but not simpler.” Next, we present two methods for incorporating model complexity into the evaluation of classification models.

Pessimistic Error Estimate The first approach explicitly computes gener-alization error as the sum of training error and a penalty term for model com-plexity. The resulting generalization error can be considered its pessimistic error estimate. For instance, let n(t) be the number of training records classi-fied by node t and e(t) be the number of misclassified records. The pessimistic error estimate of a decision tree T , e_g (T), can be computed as follows:

$$e_g(T) = \frac{\sum_{i=1}^{k}[e(t_i) + \Omega(t_i)]}{\sum_{i=1}^{k} n(t_i)} = \frac{e(T) + \Omega(T)}{N_t},$$

where k is the number of leaf nodes, e(T) is the overall training error of the decision tree, N_t is the number of training records, and $\Omega(t_i)$ is the penalty term associated with each node t_i.

Example 4.2. Consider the binary decision trees shown in Figure 4.27. If the penalty term is equal to 0.5, then the pessimistic error estimate for the left tree is

$$e_g(T_L) = \frac{4 + 7 \times 0.5}{24} = \frac{7.5}{24} = 0.3125$$

and the pessimistic error estimate for the right tree is

$$e_g(T_R) = \frac{6 + 4 \times 0.5}{24} = \frac{8}{24} = 0.3333.$$

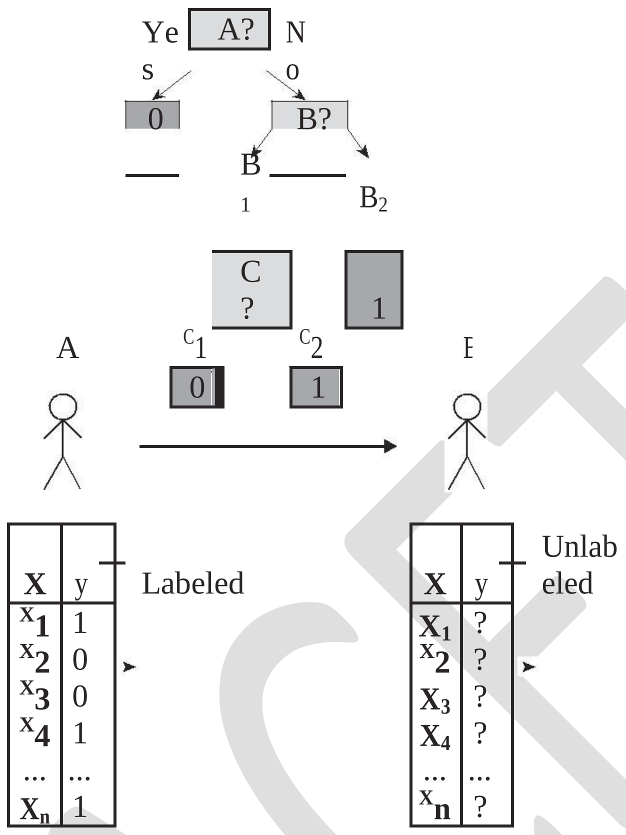


Figure 4.28. The minimum description length (MDL) principle.

Thus, the left tree has a better pessimistic error rate than the right tree. For binary trees, a penalty term of 0.5 means a node should always be expanded into its two child nodes as long as it improves the classification of at least one training record because expanding a node, which is equivalent to adding 0.5 to the overall error, is less costly than committing one training error.

If **Ω**(t) = 1 for all the nodes t, the pessimistic error estimate for the left tree is e_g (T_L) = 11/24 = 0.458, while the pessimistic error estimate for the right tree is e_g (T_R) = 10/24 = 0.417. The right tree therefore has a better pessimistic error rate than the left tree. Thus, a node should not be expanded into its child nodes unless it reduces the misclassification error for more than one training record. ■

Minimum Description Length Principle Another way to incorporate model complexity is based on an information-theoretic approach known as the minimum description length or MDL principle. To illustrate this principle, consider the example shown in Figure 4.28. In this example, both A and B are given a set of records with known attribute values x. In addition, person A knows the exact class label for each record, while person B knows none of this information. B can obtain the classification of each record by requesting that A transmits the class labels sequentially. Such a message would require Θ(n) bits of information, where n is the total number of records.

Alternatively, A may decide to build a classification model that summarizes the relationship between x and y. The model can be encoded in a compact form before being transmitted to B. If the model is 100% accurate, then the cost of transmission is equivalent to the cost of encoding the model. Otherwise, A must also transmit information about which record is classified incorrectly by the model. Thus, the overall cost of transmission is

$$\text{Cost(model, data)} = \text{Cost(model)} + \text{Cost(data|model)}, \qquad (4.9)$$

where the first term on the right-hand side is the cost of encoding the model, while the second term represents the cost of encoding the mislabeled records. According to the MDL principle, we should seek a model that minimizes the overall cost function. An example showing how to compute the total descrip-tion length of a decision tree is given by Exercise 9 on page 202.

4.4.5 Handling Over fitting in Decision Tree Induction

In the previous section, we described several methods for estimating the gen-eralization error of a classification model. Having a reliable estimate of gener-alization error allows the learning algorithm to search for an accurate model without overfitting the training data. This section presents two strategies for avoiding model over fitting in the context of decision tree induction.

Prepruning (Early Stopping Rule) In this approach, the tree-growing algorithm is halted before generating a fully grown tree that perfectly fits the entire training data. To do this, a more restrictive stopping condition must be used; e.g., stop expanding a leaf node when the observed gain in impurity measure (or improvement in the estimated generalization error) falls below a certain threshold. The advantage of this approach is that it avoids generating overly complex subtrees that overfit the training data. Nevertheless, it is difficult to choose the right threshold for early termination. Too high of a threshold will result in underfitted models, while a threshold that is set too low may not be sufficient to overcome the model overfitting problem. Furthermore,

Post-pruning In this approach, the decision tree is initially grown to its maximum size. This is followed by a tree-pruning step, which proceeds to trim the fully grown tree in a bottom-up fashion. Trimming can be done by replacing a subtree with (1) a new leaf node whose class label is determined from the majority class of records affiliated with the subtree, or (2) the most frequently used branch of the subtree. The tree-pruning step terminates when no further improvement is observed. Post-pruning tends to give better results than prepruning because it makes pruning decisions based on a fully grown tree, unlike prepruning, which can suff er from premature termination of the tree-growing process. However, for post-pruning, the additional computations needed to grow the full tree may be wasted when the subtree is pruned.

Figure 4.29 illustrates the simplified decision tree model for the Web robot detection example given in Section 4.3.6. Notice that the subtrees rooted at Classificationdepth = 1 have been replaced by one of the branches involving the attribute ImagePages. This approach is also known as subtree rising. The depth > 1 and MultiAgent = 0 subtree has been replaced by a leaf node assigned to class 0. This approach is known as subtree replacement. The subtree for depth > 1 and MultiAgent = 1 remains intact.

4.5 Evaluating the Performance of a Classifier

It described several methods for estimating the generalization error of a model during training. The estimated error helps the learning algorithm to do model selection; i.e., to find a model of the right complexity that is not susceptible to overfitting. Once the model has been constructed, it can be applied to the test set to predict the class labels of previously unseen records.

It is often useful to measure the performance of the model on the test set because such a measure provides an unbiased estimate of its generalization error. The accuracy or error rate computed from the test set can also be used to compare the relative performance of diff erent classifiers on the same domain. However, in order to do this, the class labels of the test records must be known. This section reviews some of the methods commonly used to evaluate the performance of a classifier.

4.5.1 Holdout Method

In the holdout method, the original data with labeled examples is partitioned into two disjoint sets, called the training and the test sets, respectively. A classification model is then induced from the training set and its performance is evaluated on the test set. The proportion of data reserved for training and for testing is typically at the discretion of the analysts (e.g., 50-50 or two-thirds for training and one-third for testing). The accuracy of the classifier can be estimated based on the accuracy of the induced model on the test set.

The holdout method has several well-known limitations. First, fewer la-beled examples are available for training because some of the records are with-held for testing. As a result, the induced model may not be as good as when all the labeled examples are used for training. Second, the model may be highly dependent on the composition of the training and test sets. The smaller the training set size, the larger the variance of the model. On the other hand, if the training set is too large, then the estimated accuracy computed from the smaller test set is less reliable. Such an estimate is said to have a wide con-fidence interval. Finally, the training and test sets are no longer independent of each other. Because the training and test sets are subsets of the original data, a class that is overrepresented in one subset will be underrepresented in the other, and vice versa.

4.5.2 Random Subsampling

The holdout method can be repeated several times to improve the estimation of a classifier's performance. This approach is known as random subsampling. Let acc_i be the model accuracy during the i^{th} iteration. The overall accuracy is given by acc^{sub} = ^{k} acc^{i}/k. Random sub sampling still encounters some i=1 of the problems associated with the holdout method because it does not utilize as much data as possible for training. It also has no control over the number of times each record is used for testing and training. Consequently, some records might be used for training more often than others.

4.5.3 Cross-Validation

An alternative to random subsampling is cross-validation. In this approach, each record is used the same number of times for training and exactly once for testing. To illustrate this method, suppose we partition the data into two equal-sized subsets. First, we choose one of the subsets for training and the other for testing. We then swap the roles of the subsets so that the previous training set becomes the test set and vice versa. This approach is called a two-fold cross-validation.

The total error is obtained by summing up the errors for both runs. In this example, each record is used exactly once for training and once for testing. The k-fold cross-validation method generalizes this approach by segmenting the data into k equal-sized partitions. During each run, one of the partitions is chosen for testing, while the rest of them are used for training. This procedure is repeated k times so that each partition is used for testing exactly once. Again, the total error is found by summing up the errors for all k runs. A special case of the k-fold cross-validation method sets k = N , the size of the data set. In this so-called leave-one-out approach, each test set contains only one record. This approach has the advantage of utilizing as much data as possible for training. In addition, the test sets are mutually exclusive and they eff ectively cover the entire data set. The drawback of this approach is that it is computationally expensive to repeat the procedure N times. Furthermore, since each test set contains only one record, the variance of the estimated performance metric tends to be high.

4.5.4 Bootstrap

The methods presented so far assume that the training records are sampled without replacement. As a result, there are no duplicate records in the training and test sets. In the bootstrap approach, the training records are sampled with replacement; i.e., a record already chosen for training is put back into the original pool of records so that it is equally likely to be redrawn. If the original data has N records, it can be shown that, on average, a bootstrap sample of size N contains about 63.2% of the records in the original data. This approximation follows from the fact that the probability a record is chosen by a bootstrap sample is $1 - (1 - 1/N)^N$. When N is sufficiently large, the probability asymptotically approaches $1 - e^{-1} = 0.632$. Records that are not included in the bootstrap sample become part of the test set. The model induced from the training set is then applied to the test set to obtain an estimate of the accuracy of the bootstrap sample, $\_{i}$. The sampling procedure is then repeated b times to generate b bootstrap samples.

There are several variations to the bootstrap sampling approach in terms of how the overall accuracy of the classifier is computed. One of the more widely used approaches is the .632 bootstrap, which computes the overall accuracy by combining the accuracies of each bootstrap sample ($\_{i}$) with the accuracy computed from a training set that contains all the labeled examples in the original data (acc_s):

$$-$$