Module-4 Classification

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 shape.

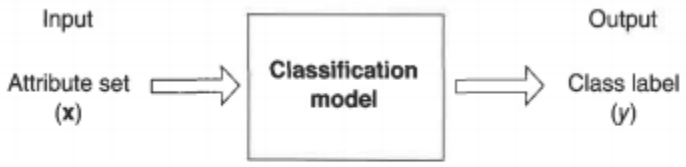


Figure 4.2. Classification as the task of mapping an input attribute set $\mathbf{x}$ into its class label y.

Definition4.1(Classification). Classification is the task of learning a target function / 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.

4.1 Decision Tree Induction

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

4.1.1 How a Decision Tree Works:

- Figure 4.4 shows the decision tree for the mammal classification problem. The tree has three types of nodes:
- **A root node** that has no incoming edges and zero or more outgoing edges.
- **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 characteristics.

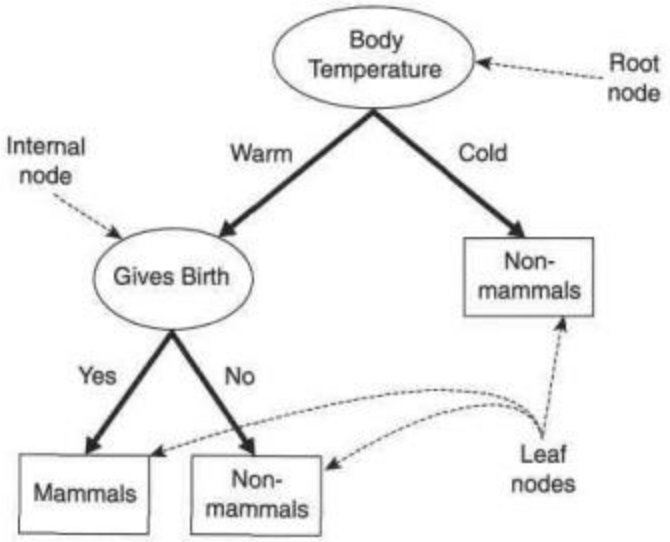


Figure 4.4. A decision tree for the mammal classification problem.

- For example, the root node shown in Figure 4.4 uses the attribute Body 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.

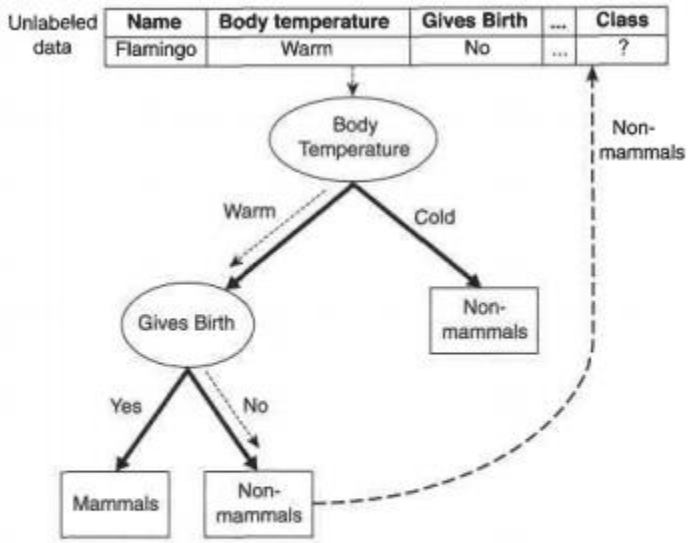


Figure 4.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the `Non-mammal` class.

4.1.2 How to Build a Decision Tree

- In principle, there are exponentially many decision trees that can be constructed from a given set of attributes. While some of the trees are more accurate 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, decision 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 optimum decisions about which attribute to use for partitioning the data. One such algorithm is hunts algorithm, which is the basis of many existing decision tree induction algorithms.
- This section presents a high-level discussion of Hunt's algorithm and illustrates some of its design issues.

Hunt's Algorithm

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

Step 1: If all the records in $\mathbf{D_t}$ belong to the same class $\mathbf{y_t}$, then **t** is a leaf node labeled as $\mathbf{y_t}$.
Step 2: If $\mathbf{D_t}$ contains records that belong to more than one class, **an attribute test condition** is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in $\mathbf{D_t}$ are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

- To illustrate how the algorithm works, consider the problem of predicting whether a loan applicant will repay her loan obligations or become delinquent, subsequently defaulting on her loan.
- A training set for this problem can be constructed by examining the records of previous borrowers. In the example shown in Figure 4.6, each record contains the personal information of a borrower along with a class label indicating whether the borrower has defaulted on loan payments.

	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 4.7a), 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 Hone Owner test condition as shown in Figure 4.7(b).
- For now, we will assume that this is the best criterion for splitting the data at this point. Hunt's algorithm is then applied recursively to each child of the root node. From the training set given in Figure 4.6, notice that all borrowers who are home owners successfully repaid their loans. The left child of the root is therefore a leaf node labeled Defaulted = No (see Figure 4.7(b)). For the right child, we need to continue applying the

recursive step of Hunt's algorithm until all the records belong to the same class. The trees resulting from each recursive step are shown in Figures 4.7(c) and (d).

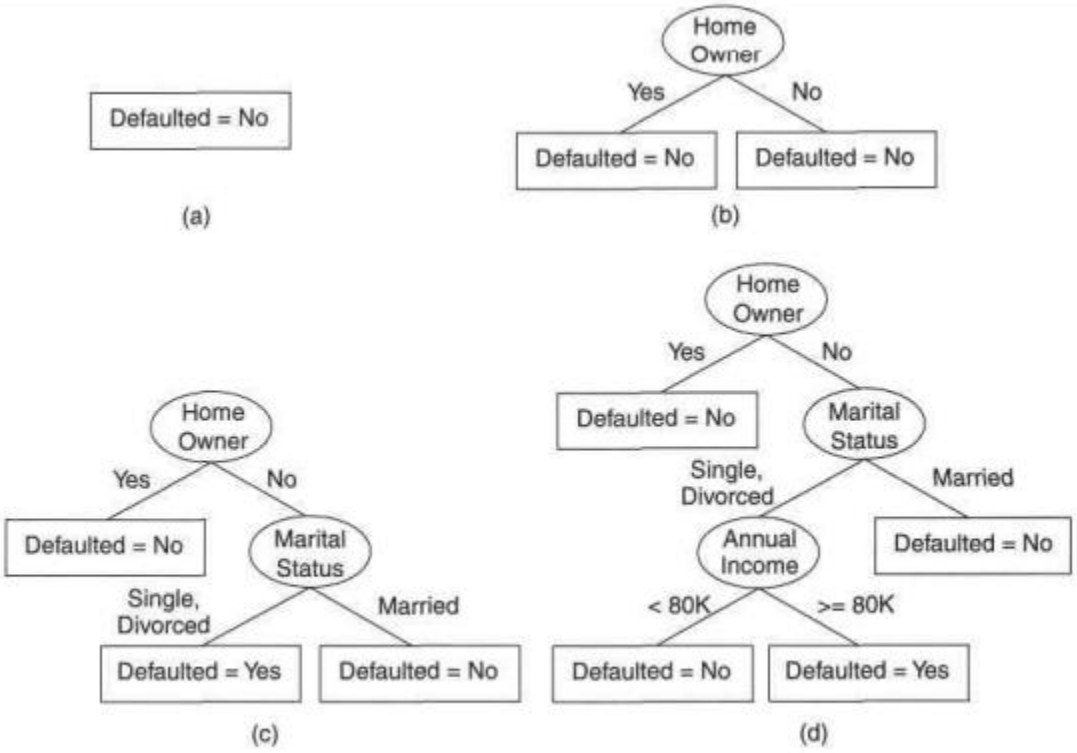


Figure 4.7. Hunt's algorithm for inducing decision trees.

- Hunt's algorithm will work if every combination of attribute values is present in the training data and each combination has a unique class label. These assumptions are too stringent for use in most practical situations. Additional conditions are needed to handle the following cases:

1. It is possible for some of the child nodes created in Step 2 to be empty; i.e., there are no records associated with these nodes. This can happen if none of the training records have the combination of attribute values associated with such nodes. In this case the node is declared a leaf node with the same class label as the majority class of training records associated with its parent node.

2. In Step 2, if all the records associated with D; have identical attribute values (except for the class label), then it is not possible to split these records any further. In this case, the node is declared a leaf node with the same class label as the majority class of training records associated with this node.

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.

4.1.3 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, as shown in Figure 4.8.

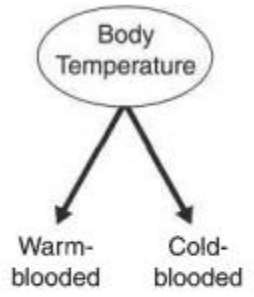


Figure 4.8. Test condition for binary attributes.

- **Nominal Attributes:** Since a nominal attribute can have many values, its test condition can be expressed in two ways, as shown in Figure 4.9. For a multiway split (Figure 4.9(a)), the number of outcomes depends on the number of distinct values for the corresponding attribute. For example, if an attribute such as marital status has three distinct values-single, married, or divorced-its test condition will produce a three-way split. Figure 4.9(b) illustrates three different ways of grouping the attribute values for marital status into two subsets.

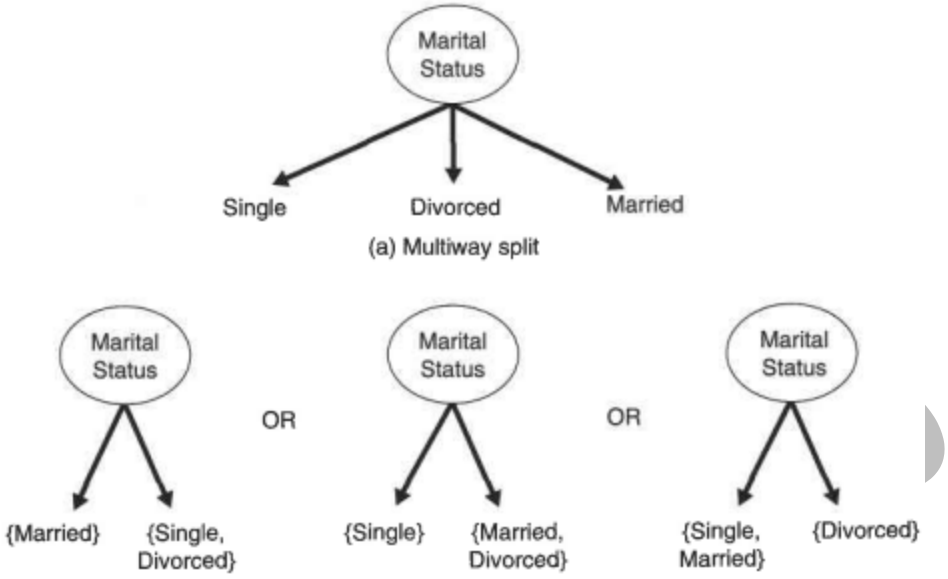


Figure 4.9. Test conditions for nominal attributes.

- **Ordinal Attributes:** Ordinal attributes can also produce binary or multi way splits. Ordinal attribute values can be grouped as long as the grouping does not violate the order property of the attribute values. Figure 4.10 illustrates various ways of splitting training records based on the Shirt Size attribute. The groupings shown in Figures 4.10(a) and (b) preserve the order among the attribute values, whereas the grouping shown in Figure 4.10(c) violates this property because it combines the attribute values Small and Large into the same partition while Medium and Extra Large are combined into another partition.

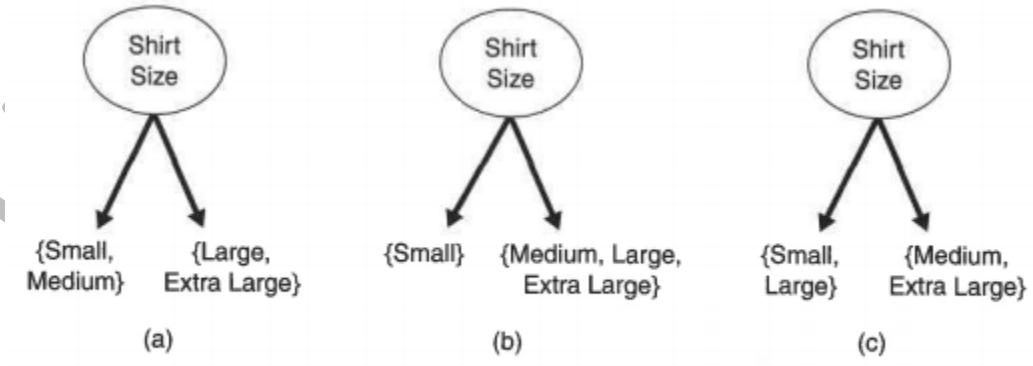


Figure 4.10. Different ways of grouping ordinal attribute values.

- **Continuous Attributes:** For continuous attributes, the test condition can be expressed as a comparison test **(A <v) or (A≥v)** with binary outcomes, or a range query with outcomes of the form $v_i \leq v_{i+1}$ **for i=1,...,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. 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.

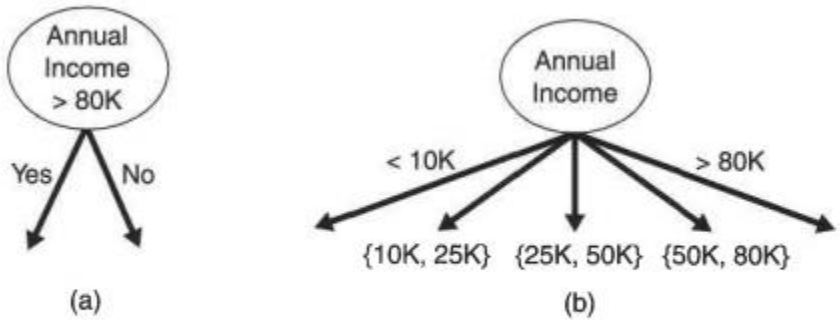


Figure 4.11. Test condition for continuous attributes.

4.1.4 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 $\mathbf{p_i}$. In a two-class problem, the class distribution at any node can be written as $\mathbf{(p_0,p_t),}$ where P1=:1- $\mathbf{p_0}$.
- To illustrate, consider the test conditions shown in Figure 4.12. The class distribution before splitting is (0.5,0.5) because there are an equal number of records from each class.
- If we split the data using the Gender attribute, then the class distributions of the child nodes are (0.6,0.4) and (0.4,0.6), respectively. Although the classes are no longer evenly distributed, the child nodes still contain records from both classes. Splitting on the second attribute, Car Type, will result in purer partitions.

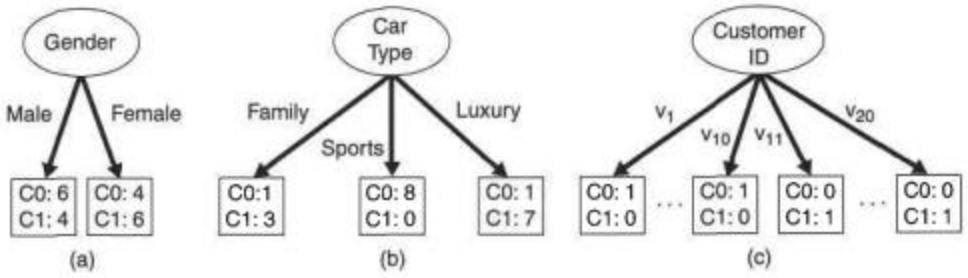


Figure 4.12. Multiway versus binary splits.

- 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. For example, a node with class distribution (0,1) has zero impurity, whereas a node with uniform class distribution (0.5,0.5) has the highest impurity.
- Examples of impurity measures include

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

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

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

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

- Figure 4.13 compares the values of the impurity measures for binary classification 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 different impurity measures.

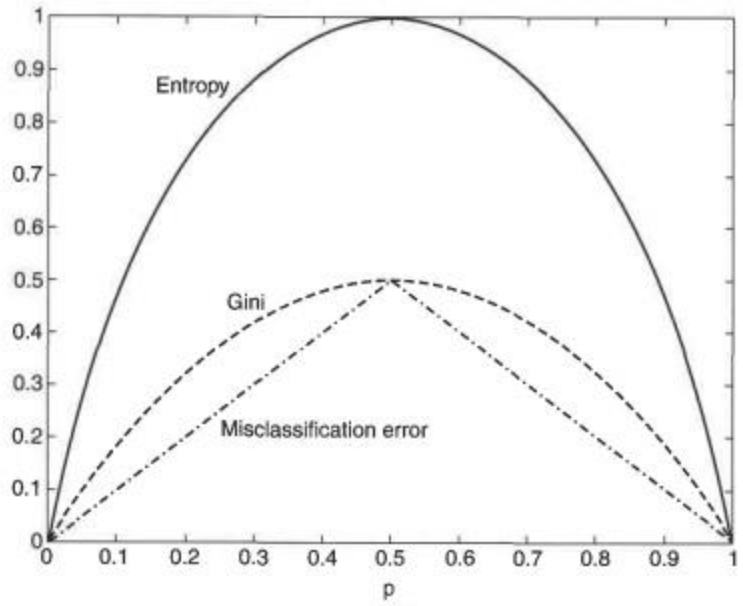


Figure 4.13. Comparison among the impurity measures for binary classification problems.

Node N_1	Count
Class=0	0
Class=1	6

$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$
$\text{Entropy} = -(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$
$\text{Error} = 1 - \max[0/6, 6/6] = 0$

Node N_2	Count
Class=0	1
Class=1	5

$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$
$\text{Entropy} = -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$
$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$

Node N_3	Count
Class=0	3
Class=1	3

$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$
$\text{Entropy} = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$
$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$

- The preceding examples, along with Figure 4.13, illustrate the consistency among different impurity measures. Based on these calculations, node **N_1** has the lowest impurity value, followed by **N_2** and **N_3**.
- To determine how well a test condition performs, we need to compare the degree of impurity of the parent node (before splitting) with the degree of impurity of the child nodes (after splitting). The larger their difference, the better the test condition. The gain, Δ, is a criterion that can be used to determine the goodness of a split:

$$\Delta = I(\text{parent}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j), \qquad (4.6)$$

where I(.) is the impurity measure of a given node, **N** is the total number of records at the parent node, k is the number of attribute values, and **$N(v_j)$** is the number of records associated with the child node**,vj**.

- Decision tree induction algorithms often choose a test condition that maximizes the gain Δ. Since **I(parent)** is the same for all test conditions, maximizing the gain is equivalent to minimizing the weighted average impurity measures of the child nodes. Finally, when entropy is used as the impurity measure in Equation 4.6, the difference in entropy is known as the **information gain**, **Δinfo**.

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) x 0.4898 + (5/12) x 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.

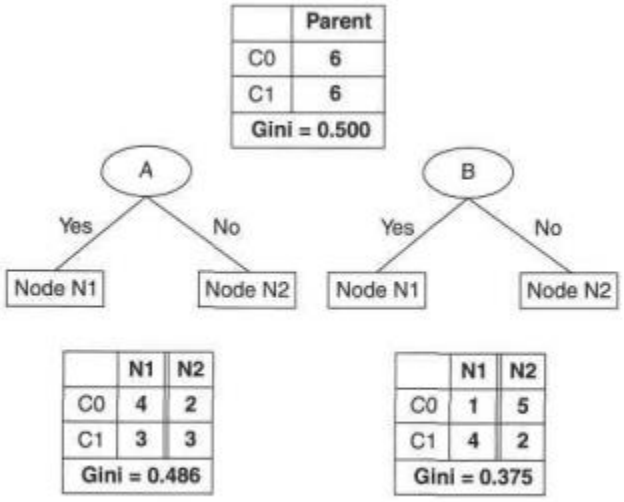
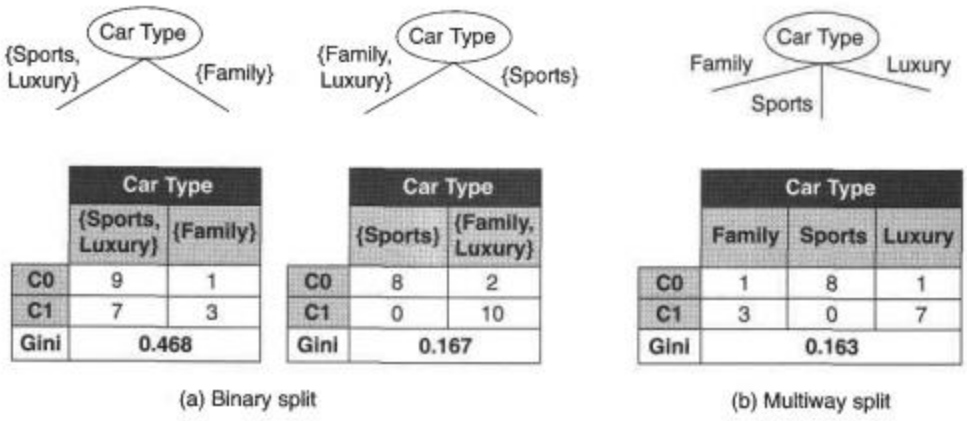


Figure 4.14. Splitting binary attributes.

Splitting of Nominal Attributes

- As previously noted, a nominal attribute can produce either binary or multiway splits, as shown in Figure 4.15.



	Car Type	
	{Sports, Luxury}	{Family}
C0	9	1
C1	7	3
Gini	0.468	

	Car Type	
	{Sports}	{Family, Luxury}
C0	8	2
C1	0	10
Gini	0.167	

	Car Type		
	Family	Sports	Luxury
C0	1	8	1
C1	3	0	7
Gini	0.163		

(a) Binary split

(b) Multiway split

Figure 4.15. Splitting nominal attributes.

- The computation of the Gini index for a binary split is similar to that shown for determining binary attributes. For the first binary grouping of the Car Type attribute, the Gini index of {Sports, Luxury) is 0.4922 and the Gini index of {Fa:rily} 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 {Fanily, Luxury}, the weighted average Gini index is 0.167. The second grouping has a lower Gini index because its corresponding subsets are much purer.

- For the multiway split, the Gini index is computed for every attribute value. Since Gini({raniry}) =0.375, Gini({Sports}) = 0, and Gini({Luxury}) =0.219, the overall Gini index for the multiway split is equal to

$$4/20 \times 0.375 + 8/20 \times 0 + 8/20 \times 0.219 = 0.163.$$

- The multiway split has a smaller Gini index compared to both two-way splits. This result is not surprising because the two-way split actually merges some of the outcomes of a multiway split, and thus, results in less pure subsets.

Splitting of Continuous Attributes

- Consider the example shown in Figure 4.16, in which the test condition **Annual Income ≤ v** is used to split the training records for the loan default classification problem.
- A brute-force method for finding **v** is to consider every value of the attribute in the **N** records as a candidate split position. For each candidate **v**, the data set is scanned once to count the number of records with annual income less than or greater than **v**.

Class	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	230

	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		0.300		0.343		0.375		0.400		0.420	

Figure 4.16. Splitting continuous attributes.

- We then compute the Gini index for each candidate and choose the one that gives the lowest value. This approach is computationally expensive because it requires **O(N)** operations to compute the Gini index at each candidate split position. Since there are -l[candidates, the overall complexity of this task is **O(N^2).**
- To reduce the complexity, the training records are sorted based on their annual income, a computation that requires **O(N log N)** time.
- Candidate split positions are identified by taking the midpoints between two adjacent sorted values: 55, 65, 72, and so on.
- However, unlike the brute-force approach, we do not have to examine all ly' records when evaluating the Gini index of a candidate split position.
- For the first candidate. **ν =55,** none of the records has annual income less than $55K.
- As a result, the Gini index for the descendent node with Annual income < $55K is zero.
- On the other hand, the number of records with annual income greater than or equal to $55K is 3 (for class Yes) and 7 (for class No), respectively. Thus, the Gini index for this node is 0.420. The overall Gini index for this candidate split position is equal to
  0 x 0 + 1 x 0.420 =0.420.
- For the second candidate. **ν** =65. we can determine its class distribution by updating the distribution of the previous candidate. More specifically, the new distribution is obtained by examining the class label of the record with the lowest annual income (i.e., $60K). Since the class label for this record is No, the count for class No is increased from 0 to 1 (for Annual Income ≤ $65K) and is decreased from 7 to 6 (for Annual Incone > $65K). The distribution for class Yes remains unchanged. The new weighted-average Gini index for this candidate split position is 0.400.
- This procedure is repeated until the Gini index values for all candidates are computed, as shown in Figure 4.16. The best split position corresponds to the one that produces the smallest Gini index, i.e., **ν** =97.
- This procedure is less expensive because it requires a constant amount of time to update the class distribution at each candidate split position. It can be further optimized by considering only candidate split positions located between two adjacent records with different class labels. For example, because the first three sorted records (with annual

incomes $60K, $70K, and $75K) have identical class labels, the best split position should not reside between $60K and $75K. Therefore, the candidate split positions at a : $55K, $65K, $72K, $87K, $92K, $110K, $I22K, $772K, and $230K are ignored because they are located between two adjacent records with the same class labels. This approach allows us to reduce the number of candidate split positions from 11 to 2.

Gain Ratio

- Impurity measures such as entropy and Gini index tend to favor attributes that have a large number of distinct values.
- Figure 4.12 shows three alternative test conditions for partitioning the data set given in Exercise 2.

Table 4.7. Data set for Exercise 2.

Customer ID	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

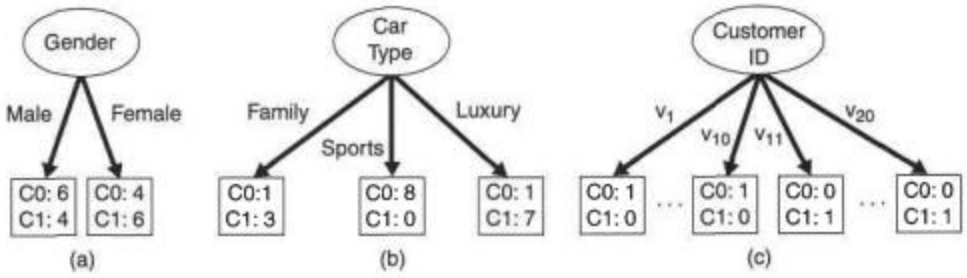


Figure 4.12. Multiway versus binary splits.

- Comparing the first test condition, Gender, with the second, Car Type, it is easy to see that Car Type seems to provide a better way of splitting the data since it produces purer descendent nodes.
- 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.
- 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 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:

$$\text{Gain ratio} = \frac{\Delta_{\text{info}}}{\text{Split Info}}. \tag{4.7}$$

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.

4.1.5 Algorithm for Decision Tree Induction

A skeleton decision tree induction algorithm called Tree Growth is shown in Algorithm 4.7. 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 tree (Steps 11 and 12) until the stopping criterion is met (Step 1). The details of this algorithm are explained below:

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.
```

1. The createNode() function extends the decision tree by creating a new node. A node in the decision tree has either a test condition, denoted as node.test-cond, or a class label, denoted as node.label.

2. The find-best-split0 function determines which attribute should be selected as the test condition for splitting the training records. As previously noted, the choice of test condition depends on which impurity measure is used to determine the goodness of a split. Some widely used measures include entropy, the Gini index, and the $\mathbf{x}^2$statistic.

3. The Classify() function determines the class label to be assigned to a leaf node. For each leaf node **t**, let **p(i|t)** denote the fraction of training records from class **i** associated with the node **t**. In most cases? the leaf node is assigned to the class that has the majority number of training records:

$$leaf.label = \underset{i}{\text{argmax}}\ p(i|t), \qquad (4.8)$$

where the argmax operator returns the argument i that maximizes the expression p(i|t). Besides providing the information needed to determine the class label of a leaf node, the fraction p(i|t) can also be used to estimate the probability that a record assigned to the leaf node t belongs to class i.

4. The stopping-cond() function is used to terminate the tree-growing process by testing whether all the records have either the same class label or the same attribute values. Another way to

terminate the recursive function is to test whether the number of records have fallen below some minimum threshold.

After building the decision tree, a tree-pruning step can be performed to reduce the size of the decision tree. Decision trees that are too large are susceptible to a phenomenon known as **overfitting.**

4.1.6 An Example: Web Robot Detection

- Web usage mining is the task of applying data mining techniques to extract useful patterns from Web access logs.
- These patterns can reveal interesting characteristics of site visitors; e.g., people who repeatedly visit a Web site and view the same product description page are more likely to buy the product if certain incentives such as rebates or free shipping are offered.
- In Web usage mining, it is important to distinguish accesses made by human users from those due to Web robots.
- A Web robot (also known as a Web crawler) is a software program that automatically locates and retrieves information from the Internet by following the hyperlinks embedded in Web pages.
- These programs are deployed by search engine portals to gather the documents necessary for indexing the Web.
- Web robot accesses must be discarded before applying Web mining techniques to analyze human browsing behavior
- This section describes how a decision tree classifier can be used to distinguish between accesses by human users and those by Web robots.
- The input data was obtained from a Web server log, a sample of which is shown in Figure 4.17(a). Each line corresponds to a single page request made by a Web client (a user or a Web robot).
- The fields recorded in the Web log include the IP address of the client, timestamp of the request, Web address of the requested document, size of the document, and the client's identity (via the user agent field).
- A Web session is a sequence of requests made by a client during a single visit to a Web site. Each Web session can be modeled as a directed graph, in which the nodes correspond to Web pages and the edges correspond to hyperlinks connecting one Web page to another.
- Figure 4.17(b) shows a graphical representation of the first Web session given in the Web server log. To classify the Web sessions, features are constructed to describe the characteristics of each session. Figure 4.17(c) shows some of the features used for the Web robot detection task. Among the notable features include the depth and breadth of the traversal. Depth determines the maximum distance of a requested page, where

distance is measured in terms of the number of hyperlinks away from the entry point of the Web site.

- For example, the home page http://www.cs.umn.edu/~kumar is assumed to be at depth 0, whereas http : / /www. cs. umn. edu/kumar/MINDS/MINDS-papers.html is located at depth 2. Based on the Web graph shown in Figure 4.17(b), the depth attribute for the first session is equal to two. The breadth attribute measures the width of the corresponding Web graph. For example, the breadth of the Web session shown in Figure 4.17(b) is equal to two.
- The data set for classification contains 2916 records, with equal numbers of sessions due to Web robots (class 1) and human users (class 0). 10% of the data were reserved for training while the remaining 90% were used for testing.
- The induced decision tree model is shown in Figure 4.18. The tree has an error rate equal to 3.8% on the training set and 5.3% on the test set.
- The model suggests that Web robots can be distinguished from human users in the following way:

1. Accesses by Web robots tend to be broad but shallow, whereas accesses by human users tend to be more focused (narrow but deep).

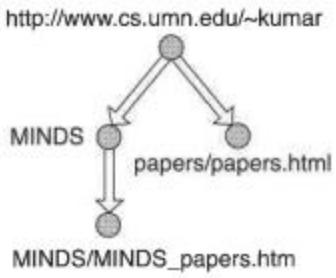
2. Unlike human users, Web robots seldom retrieve the image pages associated with a Web document.

3. Sessions due to Web robots tend to be long and contain a large number of requested pages.

4. Web robots are more likely to make repeated requests for the same document since the Web pages retrieved by human users are often cached by the browser.

Session	IP Address	Timestamp	Request Method	Requested Web Page	Protocol	Status	Number of Bytes	Referrer	User Agent
1	160.11.11.11	08/Aug/2004 10:15:21	GET	http://www.cs.umn.edu/~kumar	HTTP/1.1	200	6424		Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:34	GET	http://www.cs.umn.edu/~kumar/MINDS	HTTP/1.1	200	41378	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:41	GET	http://www.cs.umn.edu/~kumar/MINDS/MINDS_papers.htm	HTTP/1.1	200	1018516	http://www.cs.umn.edu/~kumar/MINDS	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:16:11	GET	http://www.cs.umn.edu/~kumar/papers/papers.html	HTTP/1.1	200	7463	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
2	35.9.2.2	08/Aug/2004 10:16:15	GET	http://www.cs.umn.edu/~steinbac	HTTP/1.0	200	3149		Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040616

(a) Example of a Web server log.



(b) Graph of a Web session.

Attribute Name	Description
totalPages	Total number of pages retrieved in a Web session
ImagePages	Total number of image pages retrieved in a Web session
TotalTime	Total amount of time spent by Web site visitor
RepeatedAccess	The same page requested more than once in a Web session
ErrorRequest	Errors in requesting for Web pages
GET	Percentage of requests made using GET method
POST	Percentage of requests made using POST method
HEAD	Percentage of requests made using HEAD method
Breadth	Breadth of Web traversal
Depth	Depth of Web traversal
MultiIP	Session with multiple IP addresses
MultiAgent	Session with multiple user agents

(c) Derived attributes for Web robot detection.

Figure 4.17. Input data for Web robot detection.

```
Decision Tree:
depth = 1:
| breadth> 7 : class 1
| breadth<= 7:
| | breadth <= 3:
| | | ImagePages> 0.375: class 0
| | | ImagePages<= 0.375:
| | | | totalPages<= 6: class 1
| | | | totalPages> 6:
| | | | | breadth <= 1: class 1
| | | | | breadth > 1: class 0
| | width > 3:
| | | MultiIP = 0:
| | | | ImagePages<= 0.1333: class 1
| | | | ImagePages> 0.1333:
| | | | breadth <= 6: class 0
| | | | breadth > 6: class 1
| | | MultiIP = 1:
| | | | TotalTime <= 361: class 0
| | | | TotalTime > 361: class 1
depth> 1:
| MultiAgent = 0:
| | depth > 2: class 0
| | depth < 2:
| | | MultiIP = 1: class 0
| | | MultiIP = 0:
| | | | breadth <= 6: class 0
| | | | breadth > 6:
| | | | | RepeatedAccess <= 0.322: class 0
| | | | | RepeatedAccess > 0.322: class 1
| MultiAgent = 1:
| | totalPages <= 81: class 0
| | totalPages > 81: class 1
```

Figure 4.18. Decision tree model for Web robot detection.

4.1.7 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 classification models. In other words, it does not require any prior assumptions regarding the type of probability distributions satisfied by the class and other attributes
2. Finding an optimal decision tree is an NP-complete problem. Many decision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space. For example, the algorithm presented in Section 4.1.5 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 interpret. The accuracies of the trees are also comparable to other classification 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 True. Accurate modeling of such a function requires a full decision tree with 2^d nodes, where d is the number of Boolean attributes.
6. Decision tree algorithms are quite robust to the presence of noise, especially when methods for avoiding overfitting.
7. The presence of redundant attributes does not adversely affect the accuracy of decision trees. An attribute is redundant if it is strongly correlated 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 decision trees by eliminating the irrelevant attributes during preprocessing.
8. Since most decision tree algorithms employ a top-down, recursive partitioning 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 representation 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 illustrated 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 algorithms use a divide-and-conquer partitioning strategy, the same test condition can be applied to different parts of the attribute space, thus leading to the subtree replication problem

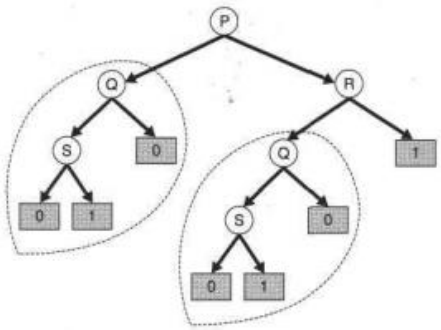


Figure 4.19. Tree replication problem. The same subtree can appear at different branches.

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).

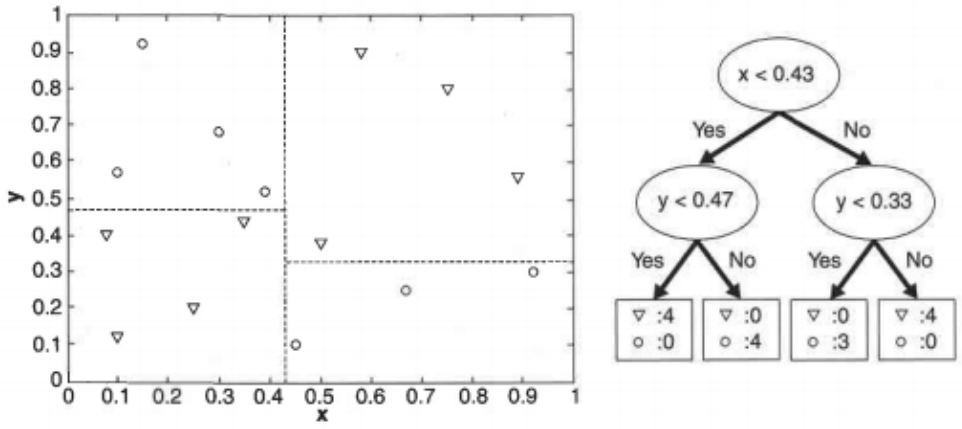


Figure 4.20. Example of a decision tree and its decision boundaries for a two-dimensional data set.

The border between two neighboring regions of different classes is known as a decision boundary. Since the test condition involves only a single attribute, the decision boundaries are rectilinear; i.e., parallel to the "coordinate axes." This limits the expressiveness of the decision tree representation for modeling complex relationships among continuous attributes. Figure 4.21 illustrates a data set that cannot be classified effectively by a decision tree algorithm that uses test conditions involving only a single attribute at a time. An oblique decision tree can be used to overcome this limitation because it allows test conditions that involve more than one attribute. The data set given in Figure 4.21can be easily represented by an oblique decision tree containing a single node with test condition

$x+y<1$

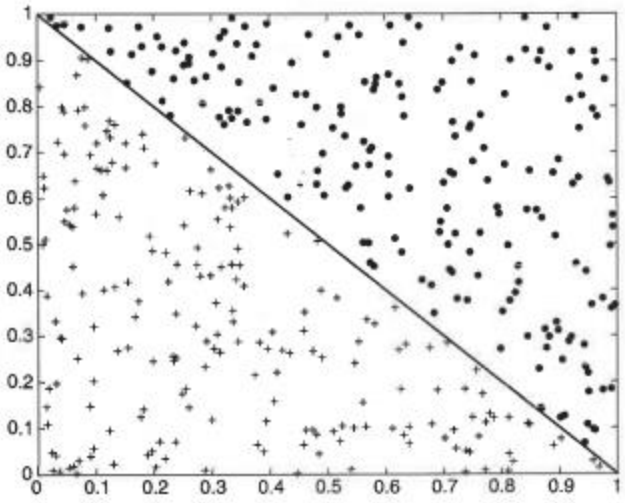


Figure 4.21. Example of data set that cannot be partitioned optimally using test conditions involving single attributes.

Constructive induction provides another way to partition the data into homogeneous) nonrectangular regions. This approach creates composite attributes representing an arithmetic or logical combination of the existing attributes. The new attributes provide a better discrimination of the classes and are augmented to the data set prior to decision tree induction. Unlike the oblique decision tree approach, constructive induction is less expensive because it identifies all the relevant combinations of attributes once, prior to constructing the decision tree.

11. Studies have shown that the choice of impurity measure has little effect on the performance of decision tree induction algorithms. This is because many impurity measures are quite consistent with each other.

4.2 Methods for Comparing Classifiers

- It is often useful to compare the performance of different classifiers to determine which classifier works better on a given data set. However, depending on the size of the data, the observed difference in accuracy between two classifiers may not be statistically significant.
- This section examines some of the statistical tests available to compare the performance of different models and classifiers.
- For illustrative purposes, consider a pair of classification models, M_A and M_B.
- Suppose M_A achieves 85% accuracy when evaluated on a test set containing 30 records, while M_B achieves 75% accuracy on a different test set containing 5000 records. Based on this information, is M_A a better model than M_B?

- The preceding example raises two key questions regarding the statistical significance of the performance metrics:
  1. Although M_A has a higher accuracy than M_B, it was tested on a smaller test set. How much confidence can we place on the accuracy for M_A?
  2. Is it possible to explain the difference in accuracy as a result of variations in the composition of the test sets?
- The first question relates to the issue of estimating the confidence interval of a given model accuracy. The second question relates to the issue of testing the statistical significance of the observed deviation. These issues are investigated in the remainder of this section.

4.2.1 Estimating a Confidence Interval for Accuracy

- To determine the confidence interval, we need to establish the probability distribution that governs the accuracy measure.
- This section describes an approach for deriving the confidence interval by modeling the classification task as a binomial experiment.
- Following is a list of characteristics of a binomial experiment:
  1. The experiment consists of N independent trials, where each trial has two possible outcomes: success or failure.
  2. The probability of success, p, in each trial is constant.
- An example of a binomial experiment is counting the number of heads that turn up when a coin is flipped N times. If X is the number of successes observed in N trials, then the probability that X takes a particular value is given by a binomial distribution with mean Np and variance Np(1 - p):

$$P(X = v) = \binom{N}{p} p^v (1-p)^{N-v}.$$

For example, if the coin is fair ($p = 0.5$) and is flipped fifty times, then the probability that the head shows up 20 times is

$$P(X = 20) = \binom{50}{20} 0.5^{20}(1-0.5)^{30} = 0.0419.$$

If the experiment is repeated many times, then the average number of heads expected to show up is $50 \times 0.5 = 25$, while its variance is $50 \times 0.5 \times 0.5 = 12.5$.

- The task of predicting the class labels of test records can also be considered as a binomial experiment.
- Given a test set that contains N records, let X be the number of records correctly predicted by a model and p be the true accuracy of the model. By modeling the prediction

task as a binomial experiment, X has a binomial distribution with mean Np and variance Np(1 - p).

- It can be shown that the empirical accuracy? acc =X/N , also has a binomial distribution with mean p and variance p(1-p)/N . Although the binomial distribution can be used to estimate the confidence interval for acc, it is often approximated by a normal distribution when N is sufficiently large. Based on the normal distribution, the following confidence interval for acc can be derived:

$$P\left(-Z_{\alpha/2} \leq \frac{acc - p}{\sqrt{p(1-p)/N}} \leq Z_{1-\alpha/2}\right) = 1 - \alpha, \qquad (4.12)$$

where $Z_{\alpha/2}$ and $Z_{1-\alpha/2}$ are the upper and lower bounds obtained from a standard normal distribution at confidence level $(1-\alpha)$. Since a standard normal distribution is symmetric around $Z = 0$, it follows that $Z_{\alpha/2} = Z_{1-\alpha/2}$. Rearranging this inequality leads to the following confidence interval for p:

$$\frac{2 \times N \times acc + Z^2_{\alpha/2} \pm Z_{\alpha/2}\sqrt{Z^2_{\alpha/2} + 4Nacc - 4Nacc^2}}{2(N + Z^2_{\alpha/2})}. \qquad (4.13)$$

The following table shows the values of $Z_{\alpha/2}$ at different confidence levels:

$1-\alpha$	0.99	0.98	0.95	0.9	0.8	0.7	0.5
$Z_{\alpha/2}$	2.58	2.33	1.96	1.65	1.28	1.04	0.67

Example 4.4. Consider a model that has an accuracy of 80% when evaluated on 100 test records. What is the confidence interval for its true accuracy at a 95% confidence level? The confidence level of 95% corresponds to $Z_{\alpha/2} = 1.96$ according to the table given above. Inserting this term into Equation 4.13 yields a confidence interval between 71.1% and 86.7%. The following table shows the confidence interval when the number of records, N, increases:

N	20	50	100	500	1000	5000
Confidence Interval	0.584 – 0.919	0.670 – 0.888	0.711 – 0.867	0.763 – 0.833	0.774 – 0.824	0.789 – 0.811

Note that the confidence interval becomes tighter when N increases. ■

4.2.2 Comparing the Performance of Two Models

Consider a pair of models, M1 and M2, that are evaluated on two independent test sets, D1 and D2. Let n1 denote the number of records in D1 and n2 denote the number of records in D2. In addition, suppose the error rate for M1 on Dl is el and the error rate for M2 on D2 is e2. Our goal is to test whether the observed difference between e1 and e2 is statisticaliy significant. Assuming

that n1 and n2 are sufficiently large, the error rates e1 and e2 can be approximated using normal distributions. If the observed difference in the error rate is denoted as d =e1-e2 then d is also normally distributed with mean d1,its true difference, and variance, σ_{d}^{2}. . The variance of d can be computed as follows:

$$\sigma_d^2 \simeq \hat{\sigma}_d^2 = \frac{e_1(1-e_1)}{n_1} + \frac{e_2(1-e_2)}{n_2}, \tag{4.14}$$

where $e_1(1-e_1)/n_1$ and $e_2(1-e_2)/n_2$ are the variances of the error rates. Finally, at the $(1-\alpha)\%$ confidence level, it can be shown that the confidence interval for the true difference d_t is given by the following equation:

$$d_t = d \pm z_{\alpha/2}\hat{\sigma}_d. \tag{4.15}$$

Example 4.5. Consider the problem described at the beginning of this section. Model M_A has an error rate of $e_1 = 0.15$ when applied to $N_1 = 30$ test records, while model M_B has an error rate of $e_2 = 0.25$ when applied to $N_2 = 5000$ test records. The observed difference in their error rates is $d = |0.15 - 0.25| = 0.1$. In this example, we are performing a two-sided test to check whether $d_t = 0$ or $d_t \neq 0$. The estimated variance of the observed difference in error rates can be computed as follows:

$$\hat{\sigma}_d^2 = \frac{0.15(1-0.15)}{30} + \frac{0.25(1-0.25)}{5000} = 0.0043$$

or $\hat{\sigma}_d = 0.0655$. Inserting this value into Equation 4.15, we obtain the following confidence interval for d_t at 95% confidence level:

$$d_t = 0.1 \pm 1.96 \times 0.0655 = 0.1 \pm 0.128.$$

As the interval spans the value zero, we can conclude that the observed difference is not statistically significant at a 95% confidence level. ■

4.2.3 Comparing the Performance of Two Classifiers

- Suppose we want to compare the performance of two classifiers using the k-fold cross-validation approach. Initially, the data set D is divided into k equal-sized partitions. We then apply each classifier to construct a model from k - 1 of the partitions and test it on the remaining partition.
- This step is repeated k times, each time using a different partition as the test set.

- Let Mij denote the model induced by classification technique Li during the j^{th} iteration. Note that each pair of models M1j and, M2j are tested on the same partition j.
- Let e_{1j} and e_{2j} be their respective error rates. The difference between their error rates during the j^{th} fold can be written as $dj = e_{1j} - e_{2j}$.
- If k is sufficiently large, then dj is normally distributed with mean d_t^{cv}, which is the true difference in their error rates, and variance σ^{cv}. Unlike the previous approach, the overall variance in the observed differences is estimated using the following formula:

$$\widehat{\sigma}^2_{d^{cv}} = \frac{\sum_{j=1}^{k}(d_j - \overline{d})^2}{k(k-1)}, \tag{4.16}$$

where $\overline{d}$ is the average difference. For this approach, we need to use a t-distribution to compute the confidence interval for d_t^{cv}:

$$d_t^{cv} = \overline{d} \pm t_{(1-\alpha),k-1}\widehat{\sigma}_{d^{cv}}.$$

The coefficient $t_{(1-\alpha),k-1}$ is obtained from a probability table with two input parameters, its confidence level $(1-\alpha)$ and the number of degrees of freedom, $k-1$. The probability table for the t-distribution is shown in Table 4.6.

Example 4.6. Suppose the estimated difference in the accuracy of models generated by two classification techniques has a mean equal to 0.05 and a standard deviation equal to 0.002. If the accuracy is estimated using a 30-fold cross-validation approach, then at a 95% confidence level, the true accuracy difference is

$$d_t^{cv} = 0.05 \pm 2.04 \times 0.002. \tag{4.17}$$

Table 4.6. Probability table for t-distribution.

	$(1-\alpha)$				
$k-1$	0.99	0.98	0.95	0.9	0.8
1	3.08	6.31	12.7	31.8	63.7
2	1.89	2.92	4.30	6.96	9.92
4	1.53	2.13	2.78	3.75	4.60
9	1.38	1.83	2.26	2.82	3.25
14	1.34	1.76	2.14	2.62	2.98
19	1.33	1.73	2.09	2.54	2.86
24	1.32	1.71	2.06	2.49	2.80
29	1.31	1.70	2.04	2.46	2.76

Since the confidence interval does not span the value zero, the observed difference between the techniques is statistically significant. ■

4.3 Rule Based Classifier

- A rule-based classifier is a technique for classifying records using a collection of "if . . .then. . ." rules. Table 5.1 shows an example of a model generated by a rule-based classifier for the vertebrate classification problem.
- The rules for the model are represented in a disjunctive normal form, .R : (r1∨r2∨...r_k), where R is known as the rule set and r;'s are the classification rules or disjuncts.

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

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

- Each classification rule can be expressed in the following way:

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

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

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

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

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

$$\text{Coverage}(r) = \frac{|A|}{|D|}$$

$$\text{Accuracy}(r) = \frac{|A \cap y|}{|A|},$$

- Where |A| is the number of records that satisfy the rule antecedent, |A∩y| is the number of records that satisfy both the antecedent and consequent, and |D| is the total number of records.
- Consider the following example to illustrate accuracy and coverage.

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

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

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

Table 5.2. The vertebrate data set.

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

4.3.1 How a Rule-Based Classifier Works

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

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

- The first vertebrate, which is a lemur, is warm-blooded and gives birth to its young. It triggers the rule r3, and thus, is classified as a mammal.
- The second vertebrate, which is a turtle, triggers the rules r4 and r5.
- None of the rules are applicable to a dogfish shark.
- In this case, we need to ensure that the classifier can still make a reliable prediction even though a test record is not covered by any rule. The previous example illustrates two important properties of the rule set generated by a rule-based classifier.

    1. **Mutually Exclusive Rules:** The rules in a rule set R are mutually exclusive if no two rules in .R are triggered by the same record. This property ensures that every record is covered by at most one rule in R. An example of a mutually exclusive rule set is shown in Table 5.3.
    2. **Exhaustive Rules:** A rule set R has exhaustive coverage if there is a rule for each combination of attribute values. This property ensures that every record is covered by at least one rule in -R. Assuming that Body Temperature and Gives Birth are binary variables, the rule set shown in Table 5.3 has exhaustive coverage

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

- Together, these properties ensure that every record is covered by exactly one rule. Unfortunately, many rule-based classifiers, including the one shown in Table 5.1, do not have such properties. If the rule set is not exhaustive, then a default rule, $rd: () \rightarrow yd$, must be added to cover the remaining cases.y_d is the default class.
- If the rule set is not mutually exclusive, then a record can be covered by several rules, some of which may predict conflicting classes. There are two ways to overcome this problem.
    1. **Ordered Rules** In this approach, the rules in a rule set are ordered in decreasing order of their priority, which can be defined in many ways (e.g., based on accuracy, coverage, total description length, or the order in which the rules are generated). An ordered rule set is also known as a decision list. When a test record is presented, it is

classified by the highest-ranked rule that covers the record. This avoids the problem of having conflicting classes predicted by multiple classification rules.

2. **Unordered Rules**: This approach allows a test record to trigger multiple classification rules and considers the consequent of each rule as a vote for a particular class. The votes are then tallied to determine the class label of the test record. The record is usually assigned to the class that receives the highest number of votes. In some cases, the vote may be weighted by the rule's accuracy. Using unordered rules to build a rule-based classifier has both advantages and disadvantages. Unordered rules are less susceptible to errors caused by the wrong rule being selected to classify a test record.

4.3.2 Rule Ordering Schemes

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

Rule-Based Ordering

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

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

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

(Aquatic Creature=semi)) ==> Amphibians

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

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

(Skin Cover=none) ==> Amphibians

Class-Based Ordering

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

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

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

(Aquatic Creature=semi)) ==> Amphibians

(Skin Cover=none) ==> Amphibians

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

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

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

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

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

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

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

4.3.3 How to Build a Rule-Based Classifier

- To build a rule-based classifier, we need to extract a set of rules that identifies key relationships between the attributes of a data set and the class label.
- There are two broad classes of methods for extracting classification rules: (1) direct methods, which extract classification rules directly from data, and (2) indirect methods, which extract classification rules from other classification models, such as decision trees and neural networks

4.3.4 Direct Methods for Rule Extraction

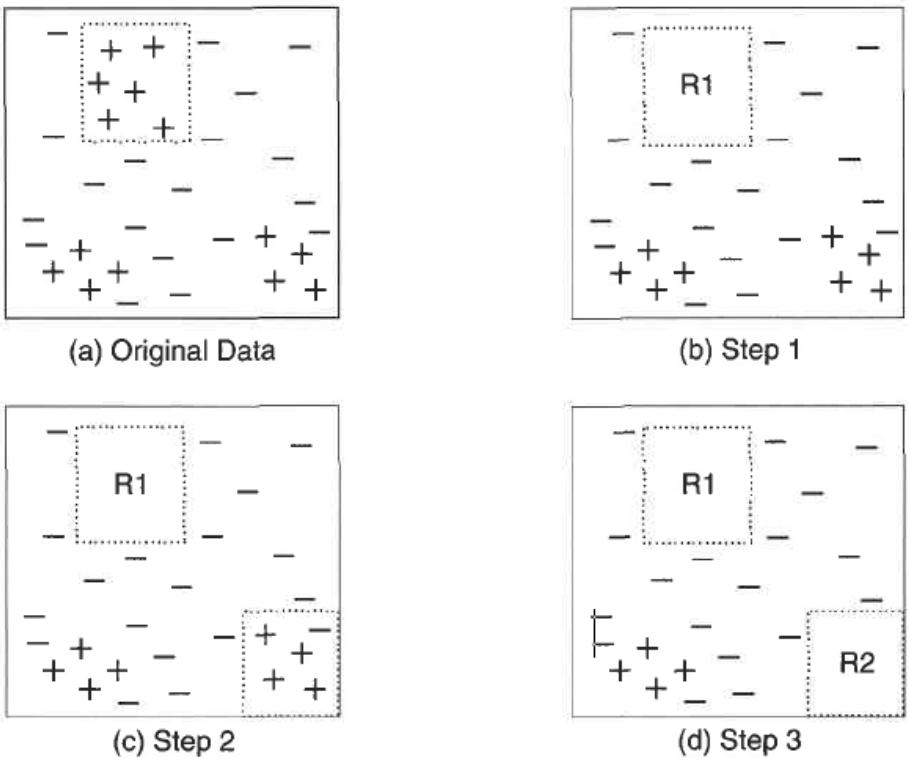
- The sequential covering algorithm is often used to extract rules directly from data. The algorithm extracts the rules one class at a time for data sets that contain more than two classes. For the vertebrate classification problem, the sequential covering algorithm may generate rules for classifying birds first, followed by rules for classifying mammals, amphibians, reptiles, and finally, fishes (see Figure 5.1).
- The criterion for deciding which class should be generated first depends on a number of factors, such as the class prevalence (i.e., fraction of training records that belong to a particular class) or the cost of misclassifying records from a given class.
- A summary of the sequential covering algorithm is given in Algorithm 5.1. The algorithm starts with an empty decision list, .R. The Learn-One- Rule function is then used to extract the best rule for class y that covers the current set of training records. During rule extraction, all training records for class y are considered to be positive examples, while those that belong to other classes are considered to be negative examples.
- A rule is desirable if it covers most of the positive examples and none (or very few) of the negative examples. Once such a rule is found, the training records covered by the rule are eliminated. The new rule is added to the bottom of the decision list R.

- This procedure is repeated until the stopping criterion is met. The algorithm then proceeds to generate rules for the next class.

Algorithm 5.1 Sequential covering algorithm.

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

Figure 5.2 demonstrates how the sequential covering algorithm works for a data set that contains a collection of positive and negative examples. The rule , R1, whose coverage is shown in Figure 5.2(b), is extracted first because it covers the largest fraction of positive examples. All the training records covered by Rl are subsequently removed and the algorithm proceeds to look for the next best rule. which is R2.



(a) Original Data (b) Step 1 (c) Step 2 (d) Step 3

Figure 5.2. An example of the sequential covering algorithm.

Learn-One-Rule Function

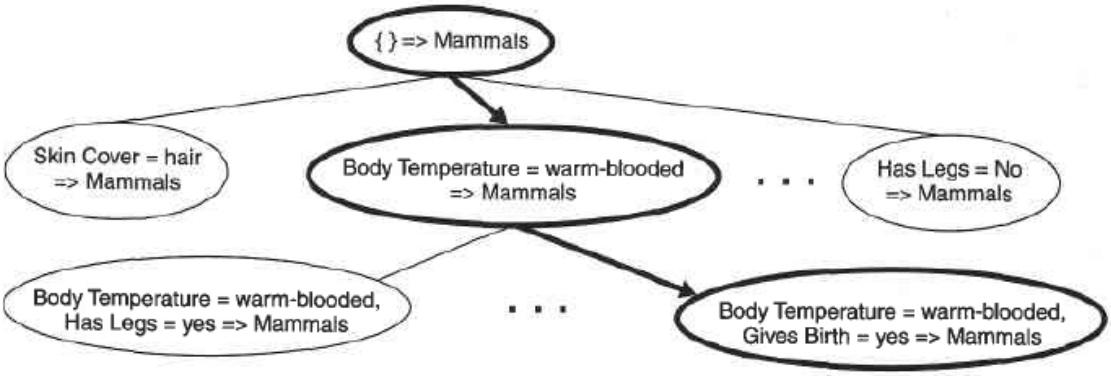
The objective of the Learn-One-Rule function is to extract a classification rule that covers many of the positive examples and none (or very few) of the negative examples in the training set. However, finding an optimal rule is computationally expensive given the exponential size of the search space.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

Rule Growing Strategy

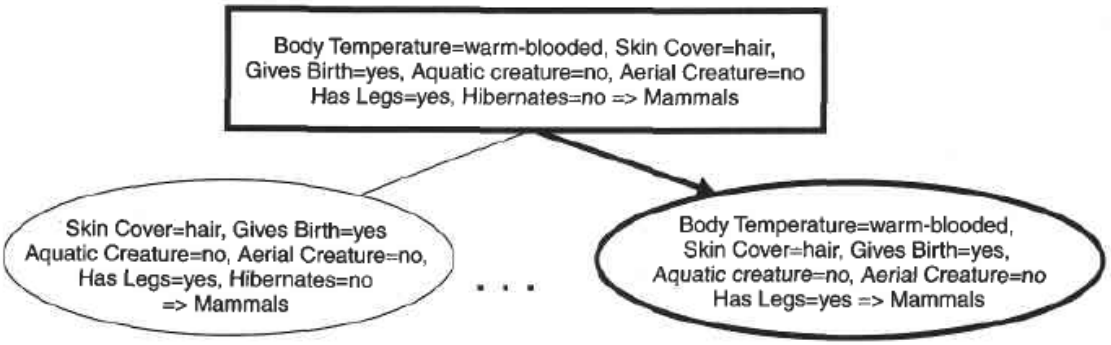
- There are two common strategies for growing a Classification rule: general-to-specific or specific-to-general. Under the general to- specific strategy, an initial rule r:$\{\} \rightarrow y$ is created, where the left-hand side is an empty set and the right-hand side contains the target class. The rule has poor quality because it covers all the examples in the training set.
- New conjuncts are subsequently added to improve the rule's quality. Figure 5.3(a) shows the **general-to-specific** rule-growing strategy for the vertebrate classification problem. The conjunct Body Tenperature=warn-blooded is initially chosen to form the rule

antecedent. The algorithm then explores all the possible candidates and greedily chooses the next conjunct, Gives Birth=yes, to be added into the rule antecedent. This process continues until the stopping criterion is met (e.g., when the added conjunct does not improve the quality of the rule).

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



(a) General-to-specific



(b) Specific-to-general

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

Rule Evaluation

An evaluation metric is needed to determine which conjunct should be added (or removed) during the rule-growing process. Accuracy is an obvious choice because it explicitly measures the fraction of training examples classified correctly by the rule. However, a potential limitation of accuracy is that it does not take into account the rule's coverage. For example, consider a training set that contains 60 positive examples and 100 negative examples. Suppose we are given the following two candidate rules:
Rule r1: covers 50 positive examples and 5 negative examples.
Rule r2: covers 2 positive examples and no negative examples.
The accuracies for r1 and r2 are 90.9% and 100%, respectively. However,r1 is the better rule despite its lower accuracy. The high accuracy for r2 is potentially spurious because the coverage of the rule is too low.
The following approaches can be used to handle this problem.

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

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

where **k** is the number of classes, $\mathbf{f_i}$ is the observed frequency of class **i** examples that are covered by the rule, and $\mathbf{e_i}$ is the expected frequency of a rule that makes random predictions. Note that ,**R** has a chi-square distribution with k - 1 degrees of freedom. A large **R** value suggests that the number of correct predictions made by the rule is significantly larger than that expected by random guessing. For example, since $\mathbf{r_1}$ covers 55 examples, the expected frequency for the positive class is $\mathbf{e_+}$: 55x60/160=20.625, while the expected frequency for the negative class is $\mathbf{e_-}$ = 55 x 100/160=34.375. Thus, the likelihood ratio for r1 is

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

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

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

This statistic therefore suggests that r1 is a better rule than r2.

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

$$\text{Laplace} = \frac{f_+ + 1}{n + k},$$
$$\text{m-estimate} = \frac{f_+ + kp_+}{n + k},$$

where n is the number of examples covered by the rule, $\mathbf{f}_+$ is the number of positive examples covered by the rule, **k** is the total number of classes, and $\mathbf{p}_+$ is the prior probability for the positive class. Note that the m-estimate is equivalent to the Laplace measure by choosing $\mathbf{p}_+=1/k$. Depending on the rule coverage, these measures capture the trade-off between rule accuracy and the prior probability of the positive class. If the rule does not cover any training example, then the Laplace measure reduces to l/ k, which is the prior probability of the positive class assuming a uniform class distribution. The m-estimate also reduces to the prior probability (p_+) when n =0. However, if the rule coverage is large, then both measures asymptotically approach the rule accuracy, $\mathbf{f}_+ / \mathbf{n}$. Going back to the previous example, the Laplace measure for r1 is 51/57 =89.47% To,which is quite close to its accuracy. Conversely, the Laplace measure for r2 (75%) is significantly lower than its accuracy because r2 has a much lower coverage.

3. An evaluation metric that takes into account the support count of the rule can be used. One such metric is the **FOILs information gain.** The support count of a rule corresponds to the number of positive examples covered by the rule. Suppose the rule $r: A \rightarrow +$ covers $\mathbf{p_0}$ positive examples and ns negative examples. After adding a new conjunct B, the extended rule $r': A \wedge B \rightarrow +$ covers $\mathbf{p_1}$ positive examples and n1 negative examples. Given this information, the FOIL's information gain of the extended rule is defined as follows:

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

Since the measure is proportional to p1 and p1/(p1+n1), it prefers rules that have high support count and accuracy. The FOIL's information gains for rules r1 and r2 given in the preceding example are 63.87 and 2.83, respectively. Therefore, r1 is a better rule than r2.

Rule Pruning

The rules generated by the Learn-One-Rule function can be pruned to improve their generalization errors.

Rationale for Sequential Covering

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

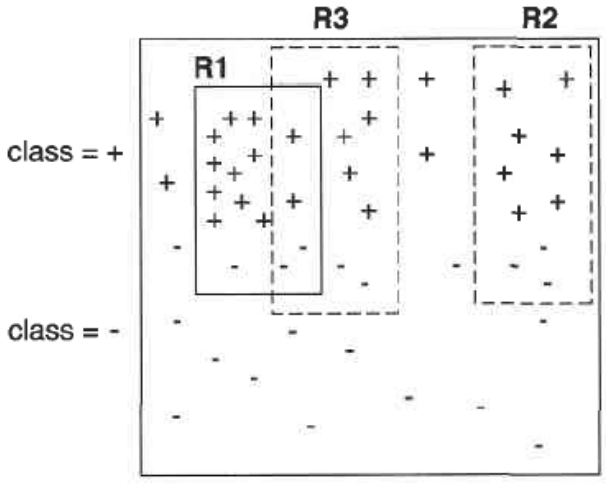


Figure 5.4. Elimination of training records by the sequential covering algorithm. $R1$, $R2$, and $R3$ represent regions covered by three different rules.

RIPPER Algorithm

- To illustrate the direct method, we consider a widely used rule induction algorithm called RIPPER. This algorithm scales almost linearly with the number of training examples and is particularly suited for building models from data\ sets with imbalanced class distributions. RIPPER also works well with noisy data sets because it uses a validation set to prevent model overfitting.
- For two-class problems, RIPPER chooses the majority class as its default class and learns the rules for detecting the minority class. For multiclass problems, the classes are ordered according to their frequencies. Let (y_1,y_2 . . . ,y_c) be the ordered classes, where y1 is the least frequent class and y_c is the most frequent class. During the first iteration, instances that belong to y_1 are labeled as positive examples, while those that belong to other classes are labeled as negative examples. The sequential covering method is used to generate rules that discriminate between the positive and negative examples. Next, RIPPER extracts rules that distinguish y_2 frorn other remaining classes. This process is repeated until we are left with $\mathbf{y_c}$, which is designated as the default class.
- **Rule Growing:** RIPPER employs a general-to-specific strategy to grow a rule and the FOIL's information gain measure to choose the best conjunct to be added into the rule antecedent. It stops adding conjuncts when the rule starts covering negative examples. The new rule is then pruned based on its performance on the validation set. The following metric is computed todetermine whether pruning is needed: (p-n) /(p+n), where p (n) is the number of positive (negative) examples in the validation set covered by the rule. This metric is monotonically related to the rule's accuracy on the validation set. If the metric improves after pruning, then the conjunct is removed. Pruning is done starting from the last conjunct added to the rule. For example, given a rule $ABCD \rightarrow$a, RIPPER checks whether D should be pruned first, followed by CD, BCD, etc. While the original rule covers only positive examples, the pruned rule may cover some of the negative examples in the training set.
- **Building the rule set:** After generating a rule, all the positive and negative examples covered by the rule are eliminated. The rule is then added into the rule set as long as it does not violate the stopping condition, which is based on the minimum description length principle. If the new rule increases the total description length of the rule set by at least d bits, then RIPPER stops adding rules into its rule set (by default, d is chosen to be 64 bits). Another stopping condition used by RIPPER is that the error rate of the rule on the validation set must not exceed 50%.

4.3.5 Indirect method for Rule Evaluation

- This section presents a method for generating a rule set from a decision tree. In principle, every path from the root node to the leaf node of a decision tree can be expressed as a classification rule.
- The test conditions encountered along the path form the conjuncts of the rule antecedent, while the class label at the leaf node is assigned to the rule consequent. Figure 5.5 shows an example of a rule set generated from a decision tree. Notice that the rule set is exhaustive and contains mutually exclusive rules. However, some of the rules can be simplified as shown in the next example.
- r3 is retained to cover the remaining instances of the positive class. Although the rules obtained after simplification are no longer mutually exclusive, they are less complex and are easier to interpret. In the following, we describe an approach used by the C4.5rules algorithm to generate a rule set from a decision tree. Figure 5.6 shows the decision tree and resulting classification rules obtained for the data set given in Table 5.2.

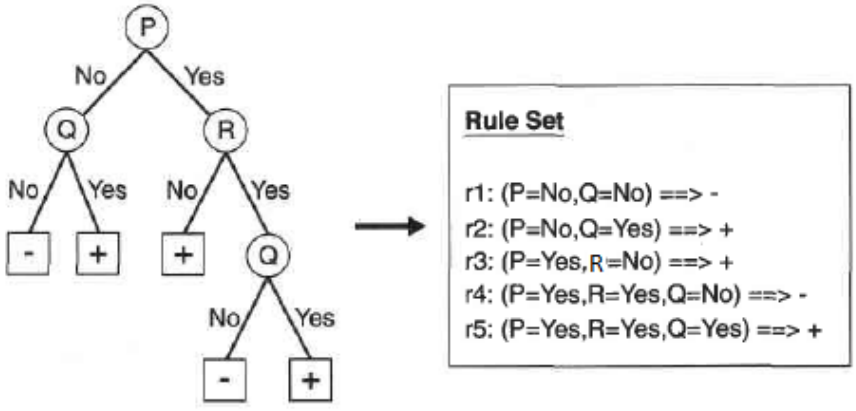


Figure 5.5. Converting a decision tree into classification rules.

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

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

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

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

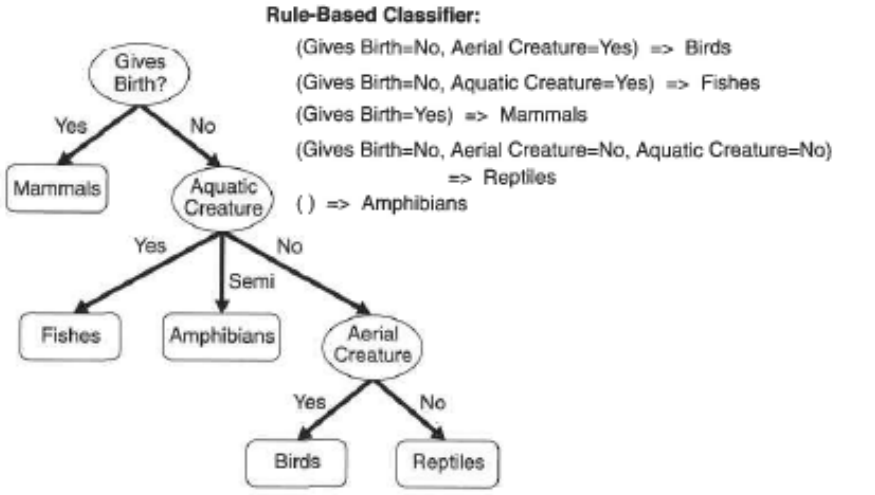


Figure 5.6. Classification rules extracted from a decision tree for the vertebrate classification problem.

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

Rule Ordering: After generating the rule set, C4.5rules uses the class-based ordering scheme to order the extracted rules. Rules that predict the same class are grouped together into the same subset. The total description length for each subset is computed, and the classes are arranged in increasing order of their total description length. The class that has the smallest description length is given the highest priority because it is expected to contain the best set of rules.

4.3.6 Characteristics of Rule-Based Classifiers

A rule-based classifier has the following characteristics:

- The expressiveness of a rule set is almost equivalent to that of a decision tree because a decision tree can be represented by a set of mutually exclusive and exhaustive rules. Both rule-based and decision tree classifier create rectilinear partitions of the attribute space and assign a class to each partition.

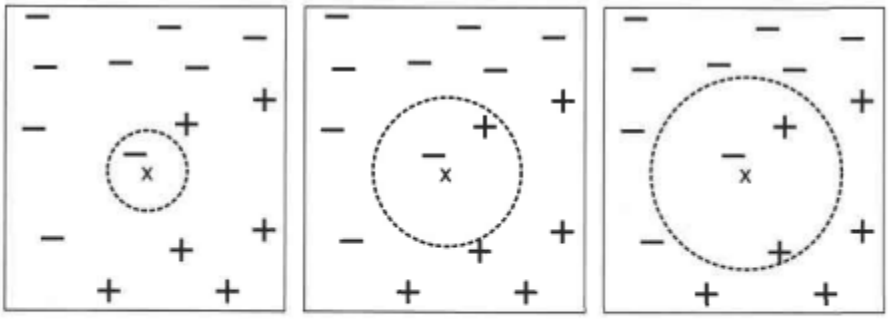
- Rule-based classifiers are generally used to produce descriptive models that are easier to interpret, but gives comparable performance to the decision tree classifier.
- The class-based ordering approach adopted by many rule-based classifiers (such as RIPPER) is well suited for handling data sets with imbalanced class distributions.

4.4 Nearest-Neighbor classifiers

- Nearest-Neighbor classifiers The classification framework shown in Figure 4.3 involves a two-step process: (1) an inductive step for constructing a classification model from data, and (2) a deductive step for applying the model to test examples.
- Decision tree and rule-based classifiers are examples of eager learners because they are designed to learn a model that maps the input attributes to the class label as soon as the training data becomes available.
- An opposite strategy would be to delay the process of modeling the training data until it is needed to classify the test examples. Techniques that employ this strategy are known as lazy learners.
- An example of a lazy learner is the Rote classifier, which memorizes the entire training data and performs classification only if the attributes of a test instance match one of the training examples exactly.
- One way to make this approach more flexible is to find all the training examples that are relatively similar to the attributes of the test example. These examples, which are known as nearest neighbors, can be used to determine the class label of the test example.
- The justification for using nearest neighbors is best exemplified by the following saying: "If it walks like a duck, quacks like a duck, and looks like a duck, then it's probably a duck."
- A nearest neighbor classifier represents each example as a data point in a d-dimensional space, where d is the number of attributes.
- Figure 5.7 illustrates the 1-, 2-, and 3-nearest neighbors of a data point located at the center of each circle. The data point is classified based on the class labels of its neighbors. In the case where the neighbors have more than one label, the data point is assigned to the majority class of its nearest neighbors.
- In Figure 5.7(a), the l-nearest neighbor of the data point is a negative example. Therefore the data point is assigned to the negative class. If the number of nearest neighbors is three, as shown in Figure 5.7(c), then the neighborhood contains two positive examples and one negative example. Using the majority voting scheme, the data point is assigned to the positive class. In the case where there is a tie

between the classes (see Figure 5.7(b)), we may randomly choose one of them to classify the data point.

- The preceding discussion underscores the importance of choosing the right value for k. If k is too small, then the nearest-neighbor classifier may be susceptible to overfitting because of noise in the training data. On the other hand, if k is too large, the nearest-neighbor classifier may misclassify the test instance because its list of nearest neighbors may include data points that are located far away from its neighborhood (see Figure 5.8)



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Figure 5.7. The 1-, 2-, and 3-nearest neighbors of an instance.

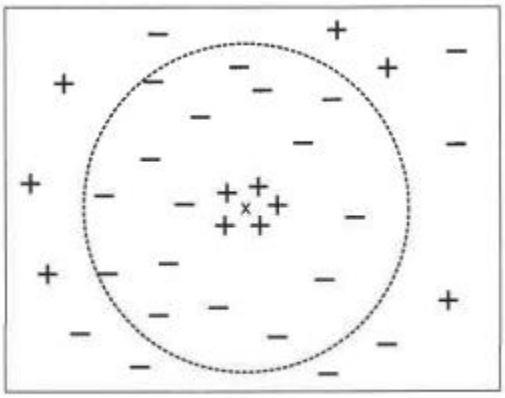


Figure 5.8. k-nearest neighbor classification with large k.

4.4.1 Algorithm

- A high-level summary of the nearest-neighbor classification method is given in Algorithm 5.2. The algorithm computes the distance (or similarity) between each test example z=(x',y') and all the training examples (x, y) Є D to determine its nearest-neighbor list, D_z.

- Such computation can be costly if the number of training examples is large. However, efficient indexing techniques are available to reduce the amount of computations needed to find the nearest neighbors of a test example.

Algorithm 5.2 The k-nearest neighbor classification algorithm.

1: Let k be the number of nearest neighbors and D be the set of training examples.
2: **for** each test example $z = (\mathbf{x}', y')$ **do**
3: Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example, $(\mathbf{x}, y) \in D$.
4: Select $D_z \subseteq D$, the set of k closest training examples to z.
5: $y' = \underset{v}{\text{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
6: **end for**

- Once the nearest-neighbor list is obtained, the test example is classified based on the majority class of its nearest neighbors:

$$\text{Majority Voting: } y' = \underset{v}{\text{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i), \qquad (5.7)$$

where **v** is a class label, $\mathbf{y_i}$ is the class label for one of the nearest neighbors, and **I(.)** is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

- In the majority voting approach, every neighbor has the same impact on the classification. This makes the algorithm sensitive to the choice of **k**, as shown in Figure 5.7. One way to reduce the impact of k is to weight the influence of each nearest neighbor **x;** according to its distance: $\boldsymbol{w_i = 1/d(x', x_i)^2}$ .As a result, training examples that are located far away from z have a weaker impact on the classification compared to those that are located close to z. Using the distance-weighted voting scheme, the class label can be determined as follows:

$$\text{Distance-Weighted Voting: } y' = \underset{v}{\text{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \times I(v = y_i). \qquad (5.8)$$

4.4.2 Characteristics of Nearest-Neighbor Classifiers

- Nearest-neighbor classification is part of a more general technique known as instance-based learning, which uses specific training instances to make predictions without having to maintain an abstraction (or model) derived from data. Instance-based learning algorithms require a proximity measure to determine the similarity or distance between instances and a classification function that returns the predicted class of a test instance based on its proximity to other instances.

- Lazy learners such as nearest-neighbor classifiers do not require model building. However, classifying a test example can be quite expensive because we need to compute the proximity values individually between the test and training examples. In contrast, eager learners often spend the bulk of their computing resources for model building. Once a model has been built, classifying a test example is extremely fast.
- Nearest-neighbor classifiers make their predictions based on local information, whereas decision tree and rule-based classifiers attempt to find a global model that fits the entire input space. Because the classification decisions are made locally, nearest-neighbor classifiers (with small values of k) are quite susceptible to noise.
- Nearest-neighbor classifiers can produce arbitrarily shaped decision boundaries. Such boundaries provide a more flexible model representation compared to decision tree and rule-based classifiers that are often constrained to rectilinear decision boundaries. The decision boundaries of nearest-neighbor classifiers also have high variability because they depend on the composition of training examples. Increasing the number of nearest neighbors may reduce such variability.
- Nearest-neighbor classifiers can produce wrong predictions unless the appropriate proximity measure and data preprocessing steps are taken. For example, suppose we want to classify a group of people based on attributes such as height (measured in meters) and weight (measured in pounds). The height attribute has a low variability, ranging from 1.5 m to 1.85 m, whereas the weight attribute may vary from 90 lb. to 250 lb. If the scale of the attributes are not taken into consideration, the proximity measure may be dominated by differences in the weights of a person.

4.5 Bayesian Classifiers

- In many applications the relationship between the attribute set and the class variable is non-deterministic. In other words, the class label of a test record cannot be predicted with certainty even though its attribute set is identical to some of the training examples.
- This situation may arise because of noisy data or the presence of certain confounding factors that affect classification but are not included in the analysis. For example, consider the task of predicting whether a person is at risk for heart disease based on the person's diet and workout frequency.
- Although most people who eat healthily and exercise regularly have less chance of developing heart disease, they may still do so because of other factors such as heredity excessive smoking, and alcohol abuse. Determining whether a person's diet is healthy or the workout frequency is sufficient is also subject to interpretation, which in turn may introduce uncertainties into the learning problem. This section

presents an approach for modeling probabilistic relationships between the attribute set and the class variable.

- The section begins with an introduction to the Bayes theorem, a statistical principle for combining prior knowledge of the classes with new evidence gathered from data. The use of the Bayes theorem for solving classification problems will be explained, followed by a description of two implementations of Bayesian classifiers: naive Bayes and the Bayesian belief network.

4.5.1 Bayes Theorem

Consider a football game between two rival teams: Team 0 and Team 1. Suppose Team 0 wins 65% of the time and Team 1 wins the remaining matches. Among the games won by Team 0, only 30%To of them come from playing on Team 1's football field. On the other hand,,75% To of the victories for Team 1 are obtained while playing at home. If Team f is to host the next match between the two teams, which team will most likely emerge as the winner?

- Let X and Y be a pair of random variables. Their joint probability, P(X=x,Y=y), refers to the probability that variable X will take on the value x and variable Y will take on the value y.
- A conditional probability is the probability that a random variable will take on a particular value given that the outcome for another random variable is known. For example, the conditional probability P(Y=y|X=x) refers to the probability that the variable Y will take on the value y, given that the variable X is observed to have the value x.
- The joint and conditional probabilities for X and Y are related in the following way:

$$P(X,Y) = P(Y|X) \times P(X) = P(X|Y) \times P(Y). \quad (5.9)$$

Rearranging the last two expressions in Equation 5.9 leads to the following formula, known as the Bayes theorem:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}. \quad (5.10)$$

- The Bayes theorem can be used to solve the prediction problem stated at the beginning of this section. For notational convenience, let X be the random variable that represents the team hosting the match and Y be the random variable that represents the winner of the match. Both X and Y can take on values from the set {0,1}. We can summarize the information given in the problem as follows:

Probability Team 0 wins is $P(Y = 0) = 0.65$.
Probability Team 1 wins is $P(Y = 1) = 1 - P(Y = 0) = 0.35$.
Probability Team 1 hosted the match it won is $P(X = 1|Y = 1) = 0.75$.
Probability Team 1 hosted the match won by Team 0 is $P(X = 1|Y = 0) = 0.3$.

Our objective is to compute $P(Y = 1|X = 1)$, which is the conditional probability that Team 1 wins the next match it will be hosting, and compares it against $P(Y = 0|X = 1)$. Using the Bayes theorem, we obtain

$$\begin{aligned} P(Y = 1|X = 1) &= \frac{P(X = 1|Y = 1) \times P(Y = 1)}{P(X = 1)} \\ &= \frac{P(X = 1|Y = 1) \times P(Y = 1)}{P(X = 1, Y = 1) + P(X = 1, Y = 0)} \\ &= \frac{P(X = 1|Y = 1) \times P(Y = 1)}{P(X = 1|Y = 1)P(Y = 1) + P(X = 1|Y = 0)P(Y = 0)} \\ &= \frac{0.75 \times 0.35}{0.75 \times 0.35 + 0.3 \times 0.65} \\ &= 0.5738, \end{aligned}$$

where the law of total probability (see Equation C.5 on page 722) was applied in the second line. Furthermore, $P(Y = 0|X = 1) = 1 - P(Y = 1|X = 1) = 0.4262$. Since $P(Y = 1|X = 1) > P(Y = 0|X = 1)$, Team 1 has a better chance than Team 0 of winning the next match.

4.5.2 Using the Bayes Theorem for Classification

- Let X denote the attribute set and Y denote the class variable.
- If the class variable has a non-deterministic relationship with the attributes, then we can treat X and Y as random variables and capture their relationship probabilistically using P(Y|X).
- This conditional probability is also known as the posterior probability for Y, as opposed to its prior probability, P(Y). During the training phase, we need to learn the posterior probabilities P(Y|X) for every combination of X and Y based on information gathered from the training data.
- By knowing these probabilities, a test record X' can be classified by finding the class Y' that maximizes the posterior probability, P(Y'lX').
- To illustrate this approach, consider the task of predicting whether a loan borrower will default on their payments. Figure 5.9 shows a training set with the following attributes: Hone Owner, Marital Status, and Annual Income. Loan borrowers who defaulted on their payments are classified as Yes, while those who repaid their loans are classified as No.

	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 5.9. Training set for predicting the loan default problem.

- Suppose we are given a test record with the following attribute set: X =(Home Owner= No, Marital Status =Married, Annual Incom=:$120K).
- To classify the record, we need to compute the posterior probabilities P(Yes|X) and P(No|X) based on information available in the training data. If P(Yes|X) > P(No|X), then the record is classified as Yes; otherwise, it is classified as No.
- Estimating the posterior probabilities accurately for every possible combination of class label and attribute value is a difficult problem because it requires a very large training set, even for a moderate number of attributes.
- The Bayes theorem is useful because it allows us to express the posterior probability in terms of the prior probability P(Y), the class-conditional probability P(X|Y), and the evidence, P(X):

$$P(Y|\mathbf{X}) = \frac{P(\mathbf{X}|Y) \times P(Y)}{P(\mathbf{X})}.$$

- When comparing the posterior probabilities for different values of Y, the denominator term, P(X), is always constant, and thus, can be ignored.
- The prior probability P(Y) can be easily estimated from the training set by computing the fraction of training records that belong to each class.
- To estimate the class-conditional probabilities P(X|Y), we present two implementations of Bayesian classification methods: **the naive Bayes classifier and the Bayesian belief network.**

4.5.3 Naive Bayes Classifier

- A naive Bayes classifier estimates the class-conditional probability by assuming that the attributes are conditionally independent, given the class label y.
- The conditional independence assumption can be formally stated as follows

$$P(\mathbf{X}|Y=y) = \prod_{i=1}^{d} P(X_i|Y=y), \qquad (5.12)$$

where each attribute set X : {X1, X2,..., Xd} consists of d attributes.

4.5.3.1 Conditional Independence

- Let X, Y, and Z denote three sets of random variables. The variables in X are said to be conditionally independent of Y, given Z, 1f the following condition holds:

P(X|Y, Z) = P(X|Z)

- An example of conditional independence is the relationship between a person's arm length and his or her reading skills. One might observe that people with longer arms tend to have higher levels of reading skills. This relationship can be explained by the presence of a confounding factor, which is age. A young child tends to have short arms and lacks the reading skills of an adult. If the age of a person is fixed, then the observed relationship between arm length and reading skills disappears. Thus, we can conclude that arm length and reading skills are conditionally independent when the age variable is fixed. The conditional independence between X and Y can also be written into a form that looks similar to Equation 5.12:

$$\begin{aligned} P(\mathbf{X},\mathbf{Y}|\mathbf{Z}) &= \frac{P(\mathbf{X},\mathbf{Y},\mathbf{Z})}{P(\mathbf{Z})} \\ &= \frac{P(\mathbf{X},\mathbf{Y},\mathbf{Z})}{P(\mathbf{Y},\mathbf{Z})} \times \frac{P(\mathbf{Y},\mathbf{Z})}{P(\mathbf{Z})} \\ &= P(\mathbf{X}|\mathbf{Y},\mathbf{Z}) \times P(\mathbf{Y}|\mathbf{Z}) \\ &= P(\mathbf{X}|\mathbf{Z}) \times P(\mathbf{Y}|\mathbf{Z}), \qquad (5.14) \end{aligned}$$

where Equation 5.13 was used to obtain the last line of Equation 5.14.

4.5.3.2 How a Naive Bayes Classifier works

- With the conditional independence assumption, instead of computing the class-conditional probability for every combination of X, we only have to estimate the conditional probability of each Xi, given Y.
- The latter approach is more practical because it does not require a very large training set to obtain a good estimate of the probability.
- To classify a test record, the naive Bayes classifier computes the posterior probability for each class Y:

$$P(Y|\mathbf{X}) = \frac{P(Y)\prod_{i=1}^{d} P(X_i|Y)}{P(\mathbf{X})}. \qquad (5.15)$$

- Since P(X) is fixed for every Y, it is sufficient to choose the class that maximizes the numerator term. In the next two subsections, we describe several approaches for estimating the conditional probabilities for $P(Y)\prod_{i=1}^{d} P(X_i|Y)$. categorical and continuous attributes.

4.5.3.3Estimating Conditional Probabilities for Categorical attributes

- For a categorical attribute X_i, the conditional probability **$P(X_i=x_i|Y=y)$** is estimated according to the fraction of training instances in class g that take on a particular attribute value x_i.
- For example, in the training set given in Figure 5.9, three out of the seven people who repaid their loans also own a home. As a result, the conditional probability for **P(Home Owner=Yes|no)** is equal to 3/7. Similarly, the conditional probability for defaulted borrowers who are single is given by **P(Marital Status =Single|Yes) =2/3.**

4.5.3.4 Estimating Conditional Probabilities for Continuous Attributes

There are two ways to estimate the class-conditional probabilities for continuous attributes in naive Bayes classifiers:

1. We can discretize each continuous attribute and then replace the continuous attribute value with its corresponding discrete interval. This approach transforms the continuous attributes into ordinal attributes. The conditional probability **P(Xi|Y=y)** is estimated by computing the fraction of training records belonging to class **y** that falls within the corresponding interval for $\mathbf{X_i}$. The estimation error depends on the discretization strategy as well as the number of discrete intervals. If the number of intervals is too large, there are too few training records in each interval to provide a reliable estimate for **P(Xi|Y).** On the other hand, if the number of intervals is too small, then some intervals may aggregate records from different classes and we may miss the correct decision boundary.

2. We can assume a certain form of probability distribution for the continuous variable and estimate the parameters of the distribution using the training data. A Gaussian distribution is usually chosen to represent the class-conditional probability for continuous attributes. The distribution is characterized by two parameters, its mean,**μ**, and variance**,** **⌐−2**. For each class **yj**, the class-conditional probability for attribute **Xi** is

$$P(X_i = x_i|Y = y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp^{-\frac{(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}. \qquad (5.16)$$

The parameter μ_{ij} can be estimated based on the sample mean of X_i ($\bar{x}$) for all training records that belong to the class y_j. Similarly, σ_{ij}^2 can be estimated from the sample variance (s^2) of such training records. For example, consider the annual income attribute shown in Figure 5.9. The sample mean and variance for this attribute with respect to the class `No` are

$$\bar{x} = \frac{125 + 100 + 70 + \ldots + 75}{7} = 110$$
$$s^2 = \frac{(125 - 110)^2 + (100 - 110)^2 + \ldots + (75 - 110)^2}{7(6)} = 2975$$
$$s = \sqrt{2975} = 54.54.$$

Given a test record with taxable income equal to $120K, we can compute its class-conditional probability as follows:

$$P(\texttt{Income=120}|\texttt{No}) = \frac{1}{\sqrt{2\pi}(54.54)} \exp^{-\frac{(120-110)^2}{2\times 2975}} = 0.0072.$$

Note that the preceding interpretation of class-conditional probability is somewhat misleading. The right-hand side of Equation 5.16 corresponds to a **probability density function**, **f(Xi,μij,⌐ij)**Since the function is continuous, the probability that the random variable Xl takes a particular value is zero. Instead, we should compute the conditional probability that X_i lies within some interval, x_i and xi+€ , where € is a small constant:

$$P(x_i \leq X_i \leq x_i + \epsilon | Y = y_j) = \int_{x_i}^{x_i+\epsilon} f(X_i; \mu_{ij}, \sigma_{ij}) dX_i \approx f(x_i; \mu_{ij}, \sigma_{ij}) \times \epsilon. \quad (5.17)$$

Since ϵ appears as a constant multiplicative factor for each class, it cancels out when we normalize the posterior probability for $P(Y|\mathbf{X})$. Therefore, we can still apply Equation 5.16 to approximate the class-conditional probability $P(X_i|Y)$.

4.5.3.5 Example of the Naïve Bayes Classifier

- Consider the data set shown in Figure 5.10(a). We can compute the class conditional probability for each categorical attribute, along with the sample mean and variance for the continuous attribute using the methodology described in the previous subsections.
- These probabilities are summarized in Figure 5.10(b). To predict the class label of a test record **X= (Home Owner=No, Marital Status =Married, Income =$120K)**, we need to compute the posterior probabilities **P(No|X) and P(Yes|X)**.
- Recall from our earlier discussion that these posterior probabilities can be estimated by computing the product between the prior probability **P(Y)** and the class-conditional probabilities,$\boldsymbol{\pi_i P(X_i|Y)}$ which corresponds to the numerator of the right-hand side term in Equation 5.15.
- The prior probabilities of each class can be estimated by calculating the fraction of training records that belong to each class. Since there are three records that belong to the class Yes and seven records that belong to the 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

(a)

P(Home Owner=Yes|No) = 3/7
P(Home Owner=No|No) = 4/7
P(Home Owner=Yes|Yes) = 0
P(Home Owner=No|Yes) = 1
P(Marital Status=Single|No) = 2/7
P(Marital Status=Divorced|No) = 1/7
P(Marital Status=Married|No) = 4/7
P(Marital Status=Single|Yes) = 2/3
P(Marital Status=Divorced|Yes) = 1/3
P(Marital Status=Married|Yes) = 0

For Annual Income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

(b)

Figure 5.10. The naïve Bayes classifier for the loan classification problem.

No, $P(\texttt{Yes}) = 0.3$ and $P(\texttt{No}) = 0.7$. Using the information provided in Figure 5.10(b), the class-conditional probabilities can be computed as follows:

$$\begin{aligned} P(\mathbf{X}|\texttt{No}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{No}) \times P(\texttt{Status} = \texttt{Married}|\texttt{No}) \\ &\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{No}) \\ &= 4/7 \times 4/7 \times 0.0072 = 0.0024. \end{aligned}$$

$$\begin{aligned} P(\mathbf{X}|\texttt{Yes}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{Yes}) \times P(\texttt{Status} = \texttt{Married}|\texttt{Yes}) \\ &\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{Yes}) \\ &= 1 \times 0 \times 1.2 \times 10^{-9} = 0. \end{aligned}$$

- Putting them together, the posterior probability for class No is **P(No|X)=α*7/10*.0024=0.0016α** where **α**=l/P(X) is a constant term. Using a similar approach, we can show that the posterior probability for class Yes is zero because its class-conditional probability is zero. Since P(No|X) > P(Yes|X), the record is classified as No.

4.5.3.6 M-estimate of conditional Probability

- The preceding example illustrates a potential problem with estimating posterior probabilities from training data. If the class-conditional probability for one of the attributes is zero, then the overall posterior probability for the class vanishes. This approach of estimating class-conditional probabilities using simple fractions may seem too brittle.
- In a more extreme case, if the training examples do not cover many of the attribute values, we may not be able to classify some of the test records. For example, if **P(Marital Status=Divorced|No) is zero instead of 1/7**, then a record with attribute set

X= (Home Owner =yes, Marital Status =Divorced, Income =$120K) has the following class-conditional probabilities:

P(X|No) = 3/7 x 0 x 0.0072=0

P(Xlves) =0 x1/3 x 7.2 x 1.2 x10^{-9}=0

- The naive Bayes classifier will not be able to classify the record. This problem can be addressed by using the m-estimate approach for estimating the conditional probabilities

$$P(x_i|y_j) = \frac{n_c + mp}{n + m}, \qquad (5.18)$$

- where **n** is the total number of instances from **class y_j,**
- **n_c** is the number of training examples from class **y_j** that take on the value **x_i**,
- **m** is a parameter known as the equivalent sample size, and p is a user-specified parameter.
- If there is no training set available then **$P(x_i|y_i)=p$**
- Therefore **p** can be regarded as the prior probability of observing the attribute value **x_i** among records with class **y_j**. The equivalent sample size determines the tradeoff between the prior probability p and the observed probability nc/n.
- In the example given in the previous section, the conditional probability **P(Status =Married|Yes)=0** because none of the training records for the class has the particular attribute value. Using the m-estimate approach with **m=3 and p=1/3,** the conditional probability is no longer zero:

$$P(\texttt{Marital Status} = \texttt{Married}|\texttt{Yes}) = (0 + 3 \times 1/3)/(3 + 3) = 1/6.$$

If we assume $p = 1/3$ for all attributes of class Yes and $p = 2/3$ for all attributes of class No, then

$$\begin{aligned} P(\mathbf{X}|\texttt{No}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{No}) \times P(\texttt{Status} = \texttt{Married}|\texttt{No}) \\ &\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{No}) \\ &= 6/10 \times 6/10 \times 0.0072 = 0.0026. \end{aligned}$$

$$\begin{aligned} P(\mathbf{X}|\texttt{Yes}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{Yes}) \times P(\texttt{Status} = \texttt{Married}|\texttt{Yes}) \\ &\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{Yes}) \\ &= 4/6 \times 1/6 \times 1.2 \times 10^{-9} = 1.3 \times 10^{-10}. \end{aligned}$$

The posterior probability for class No is $P(\texttt{No}|\mathbf{X}) = \alpha \times 7/10 \times 0.0026 = 0.0018\alpha$, while the posterior probability for class Yes is $P(\texttt{Yes}|\mathbf{X}) = \alpha \times 3/10 \times 1.3 \times 10^{-10} = 4.0 \times 10^{-11}\alpha$. Although the classification decision has not changed, the m-estimate approach generally provides a more robust way for estimating probabilities when the number of training examples is small.

4.5.3.7 Characteristics of Naive Bayes Classifiers

- They are robust to isolated noise points because such points are averaged out when estimating conditional probabilities from data. Naive Bayes classifiers can also handle missing values by ignoring the example during model building and classification.
- They are robust to irrelevant attributes. If Xi is an irrelevant attribute, then P(Xi|Y) becomes almost uniformly distributed
- Correlated attributes can degrade the performance of naive Bayes classifiers because the conditional independence assumption no longer holds for such attributes. For example, consider the following probabilities:

$$P(A=0|Y=0)=0.4, \quad P(A=1|Y=0)=0.6,$$
$$P(A=0|Y=1)=0.6, \quad P(A=1|Y=1)=0.4,$$

where **A** is a binary attribute and **Y** is a binary class variable. Suppose there is another binary attribute **B** that is perfectly correlated with **A** when **Y=0**, but is independent of **A when Y=1**. For simplicity, assume that the class-conditional probabilities for **B are the same as for A**. Given a record with attributes,**A=0,B=0** we can compute its posterior probabilities as follows:

$$\begin{aligned} P(Y=0|A=0,B=0) &= \frac{P(A=0|Y=0)P(B=0|Y=0)P(Y=0)}{P(A=0,B=0)} \\ &= \frac{0.16 \times P(Y=0)}{P(A=0,B=0)}. \end{aligned}$$

$$\begin{aligned} P(Y=1|A=0,B=0) &= \frac{P(A=0|Y=1)P(B=0|Y=1)P(Y=1)}{P(A=0,B=0)} \\ &= \frac{0.36 \times P(Y=1)}{P(A=0,B=0)}. \end{aligned}$$

If $P(Y=0)=P(Y=1)$, then the naïve Bayes classifier would assign the record to class 1. However, the truth is,

$$P(A=0,B=0|Y=0)=P(A=0|Y=0)=0.4,$$

because A and B are perfectly correlated when $Y=0$. As a result, the posterior probability for $Y=0$ is

$$\begin{aligned} P(Y=0|A=0,B=0) &= \frac{P(A=0,B=0|Y=0)P(Y=0)}{P(A=0,B=0)} \\ &= \frac{0.4 \times P(Y=0)}{P(A=0,B=0)}, \end{aligned}$$

which is larger than that for $Y=1$. The record should have been classified as class 0.

4.5.3.8 Bayes error rate

- Example: Consider the task of identifying alligators and crocodiles based on their respective lengths. The average length of an adult crocodile is about 15 feet, while the average length of an adult alligator is about 12 feet. Assuming that their length x follows a Gaussian distribution with a standard deviation equal to 2 feet, we can express their class-conditional probabilities as follows:

$$P(X|\texttt{Crocodile}) = \frac{1}{\sqrt{2\pi}\cdot 2}\exp\left[-\frac{1}{2}\left(\frac{X-15}{2}\right)^2\right] \quad (5.19)$$

$$P(X|\texttt{Alligator}) = \frac{1}{\sqrt{2\pi}\cdot 2}\exp\left[-\frac{1}{2}\left(\frac{X-12}{2}\right)^2\right] \quad (5.20)$$

- Figure 5.11 shows a comparison between the class-conditional probabilities for a crocodile and an alligator. Assuming that their prior probabilities are the same, the ideal decision boundary is located at some length x^ such that

$$P(X=\hat{x}|\texttt{Crocodile}) = P(X=\hat{x}|\texttt{Alligator}).$$

Using Equations 5.19 and 5.20, we obtain

$$\left(\frac{\hat{x}-15}{2}\right)^2 = \left(\frac{\hat{x}-12}{2}\right)^2,$$

which can be solved to yield $\hat{x} = 13.5$. The decision boundary for this example is located halfway between the two means. ■

- When the prior probabilities are different, the decision boundary shifts toward the class with lower prior probability. Furthermore, the minimum error rate attainable by any classifier on the given data can also be computed. The error rate of the classifier is given by the sum of the area under the posterior probability curve for crocodiles and the area under the posterior probability curve for alligators :

$$\text{Error} = \int_0^{\hat{x}} P(\texttt{Crocodile}|X)dX + \int_{\hat{x}}^{\infty} P(\texttt{Alligator}|X)dX.$$

The total error rate is known as the **Bayes error rate**.

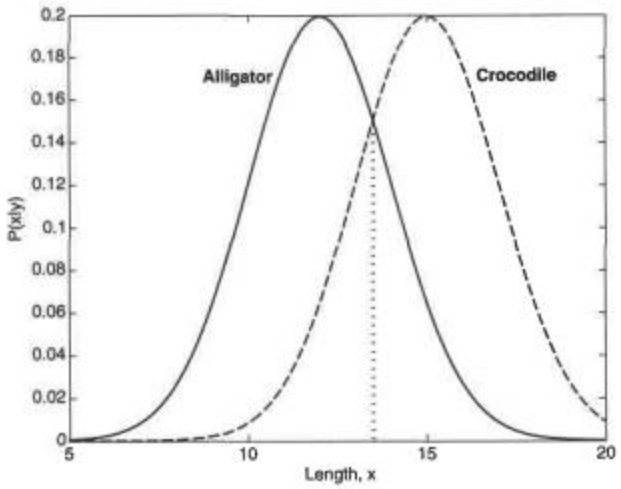


Figure 5.11. Comparing the likelihood functions of a crocodile and an alligator.

4.5.4 Bayesian Belief Networks

- The conditional independence assumption made by naive Bayes classifiers may seem too rigid, especially for classification problems in which the attributes are somewhat correlated. This section presents a more flexible approach for modeling the class-conditional probabilities P(X|Y).
- Instead of requiring all the attributes to be conditionally independent given the class, this approach allows us to specify which pair of attributes are conditionally independent. We begin with a discussion on how to represent and build such a probabilistic model, followed by an example of how to make inferences from the model.

4.5.4.1 Model Representation

- A Bayesian belief network (BBN), or simply, Bayesian network, provides a graphical representation of the probabilistic relationships among a set of random variables.
- There are two key elements of a Bayesian network:

  1. A directed acyclic graph (dag) encoding the dependence relationships among a set of variables.

  2. A probability table associating each node to its immediate parent node
- Consider three random variables, A, B, and C, in which A and B are independent variables and each has a direct influence on a third variable, C. The relationships among the variables can be summarized into the directed acyclic graph shown in Figure 5.12(a).
- Each node in the graph represents a variable, and each arc asserts the dependence relationship between the pair of variables. If there is a directed arc from X to Y, then X is the parent of Y and Y is the child of X.

- Furthermore, if there is a directed path in the network from X to Z, then X is an ancestor of Z, whlle Z is a descendant of X.

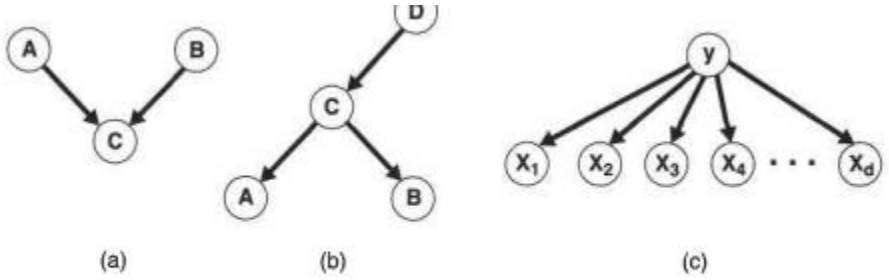


Figure 5.12. Representing probabilistic relationships using directed acyclic graphs.

- For example, in the diagram shown in Figure 5.12(b), A is a descendant of D and D is an ancestor of B. Both B and D arc also non-descendants of A. An important property of the Bayesian network can be stated as follows:

  Property 1 (Conditional Independence): *A node in a Bayesian network is conditionally independent of its non-descendants, if its parents are known.*

- In the diagram shown in Figure 5.12(b), A is conditionally independent of both B and D given C because the nodes for B and D are non-descendants of node A. The conditional independence assumption made by a naive Bayes classifier can also be represented using a Bayesian network, as shown in Figure 5.12(c), where y is the target class and {X1,X2,...,Xd} is the attribute set.
- Besides the conditional independence conditions imposed by the network topology, each node is also associated with a probability table.

  1. If a node X does not have any parents, then the table contains only the prior probability P(X).

  2. If a node X has only one parent, Y, then the table contains the conditional probability P(X|Y).

  3. If a node X has multiple parents, {Y1,Y2, . . . ,Yk}, then the table contains the conditional probability P(X|Y1,Y2,. . ., Yk.)

4.5.4.2 Model Building

- Model building in Bayesian networks involves two steps:
  - creating the structure of the network, and

- estimating the probability values in the tables associated with each node. The network topology can be obtained by encoding the subjective knowledge of domain experts. Algorithm 5.3 presents a systematic procedure for inducing the topology of a Bayesian network.

- Algorithm 5.3 guarantees a topology that does not contain any cycles. The proof for this is quite straightforward. If a cycle exists, then there must be at least one arc connecting the lower-ordered nodes to the higher-ordered nodes, and at least another arc connecting the higher-ordered nodes to the lower ordered nodes

Algorithm 5.3 Algorithm for generating the topology of a Bayesian network.

1: Let $T = (X_1, X_2, \ldots, X_d)$ denote a total order of the variables.
2: **for** $j = 1$ to d **do**
3: Let $X_{T(j)}$ denote the j^{th} highest order variable in T.
4: Let $\pi(X_{T(j)}) = \{X_{T(1)}, X_{T(2)}, \ldots, X_{T(j-1)}\}$ denote the set of variables preceding $X_{T(j)}$.
5: Remove the variables from $\pi(X_{T(j)})$ that do not affect X_j (using prior knowledge).
6: Create an arc between $X_{T(j)}$ and the remaining variables in $\pi(X_{T(j)})$.
7: **end for**

Example 5.4. Consider the variables shown in Figure 5.13. After performing Step 1, let us assume that the variables are ordered in the following way: (E, D, HD, Hb, CP, BP). From Steps 2 to 7, starting with variable D, we obtain the following conditional probabilities:

- $P(D|E)$ is simplified to $P(D)$.
- $P(HD|E, D)$ cannot be simplified.
- $P(Hb|HD, E, D)$ is simplified to $P(Hb|D)$.
- $P(CP|Hb, HD, E, D)$ is simplified to $P(CP|Hb, HD)$.
- $P(BP|CP, Hb, HD, E, D)$ is simplified to $P(BP|HD)$.

Based on these conditional probabilities, we can create arcs between the nodes (E, HD), (D, HD), (D, Hb), (HD, CP), (Hb, CP), and (HD, BP). These arcs result in the network structure shown in Figure 5.13. ■

4.5.4.3 Example of interfacing using BBN

- Figure 5.13 shows an example of a Bayesian network for modeling patients with heart disease or heartburn problems. Each variable in the diagram is assumed to be binary-valued. The parent nodes for heart disease (HD) correspond to risk factors that may affect the disease, such as exercise (E) and diet (D). The child nodes for heart disease correspond to symptoms of the disease, such as chest pain (CP) and high blood pressure (BP).

- For example, the diagram shows that heartburn (Hb) may result from an unhealthy diet and may lead to chest pain.
- The nodes associated with the risk factors contain only the prior probabilities, whereas the nodes for heart disease, heartburn, and their corresponding symptoms contain the conditional probabilities. To save space, some of the probabilities have been omitted from the diagram.

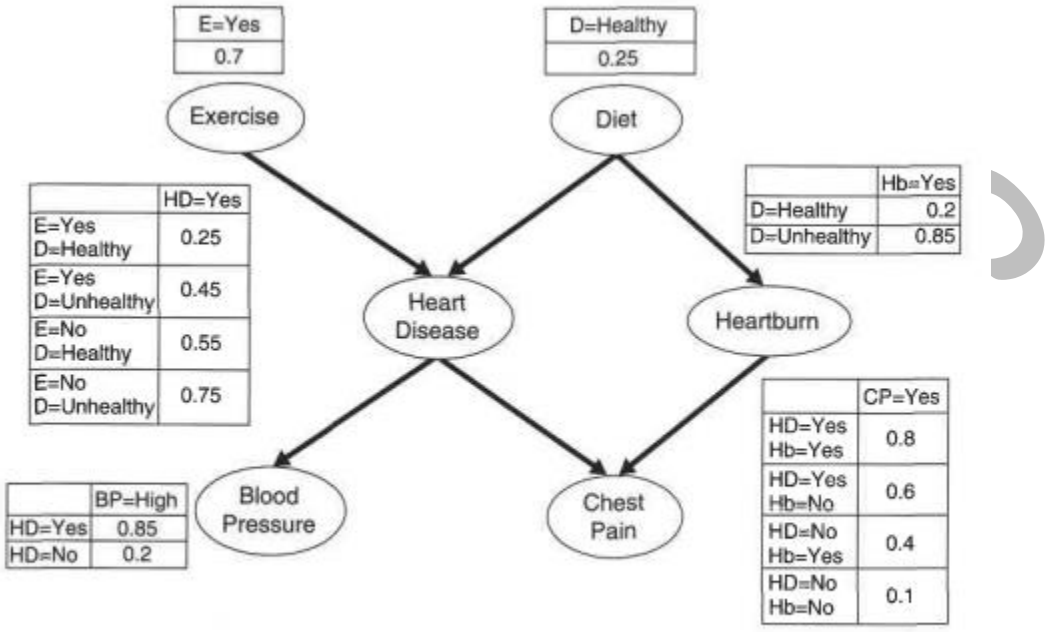


Figure 5.13. A Bayesian belief network for detecting heart disease and heartburn in patients.

- The omitted probabilities can be recovered from

abilities can be recovered by noting that $P(X = \overline{x}) = 1 - P(X = x)$ and $P(X = \overline{x}|Y) = 1 - P(X = x|Y)$, where $\overline{x}$ denotes the opposite outcome of x. For example, the conditional probability

$$\begin{aligned} & P(\texttt{Heart Disease} = \texttt{No} | \texttt{Exercise} = \texttt{No}, \texttt{Diet} = \texttt{Healthy}) \\ = \; & 1 - P(\texttt{Heart Disease} = \texttt{Yes} | \texttt{Exercise} = \texttt{No}, \texttt{Diet} = \texttt{Healthy}) \\ = \; & 1 - 0.55 = 0.45. \end{aligned}$$

- Suppose we are interested in using the BBN shown in Figure 5.13 to diagnose whether a person has heart disease. The following cases illustrate how the diagnosis can be made under different scenarios.

Case 1: No Prior Information

Without any prior information, we can determine whether the person is likely to have heart disease by computing the prior probabilities P(HD=Yes) and P(HD=No). To simplify the notation, let α ∈ {Yes,No} denote the binary values of Exercise and β∈{Healthy,Unhealthy} denote the binary values of Diet.

$$\begin{aligned} P(\texttt{HD} = \texttt{Yes}) &= \sum_{\alpha}\sum_{\beta} P(\texttt{HD} = \texttt{Yes}|E = \alpha, D = \beta)P(E = \alpha, D = \beta) \\ &= \sum_{\alpha}\sum_{\beta} P(\texttt{HD} = \texttt{Yes}|E = \alpha, D = \beta)P(E = \alpha)P(D = \beta) \\ &= 0.25 \times 0.7 \times 0.25 + 0.45 \times 0.7 \times 0.75 + 0.55 \times 0.3 \times 0.25 \\ &\quad + 0.75 \times 0.3 \times 0.75 \\ &= 0.49. \end{aligned}$$

Since $P(\texttt{HD} = \texttt{no}) = 1 - P(\texttt{HD} = \texttt{yes}) = 0.51$, the person has a slightly higher chance of not getting the disease.

Case 2: High Blood Pressure

If the person has high blood pressure we can make a diagnosis about heart disease by comparing the posterior probabilities, P(HD =Yes|BP=High) against P(HD=No|BPHigh). To do this, we must compute P(BP+ High):

$$\begin{aligned} P(\texttt{BP} = \texttt{High}) &= \sum_{\gamma} P(\texttt{BP} = \texttt{High}|\texttt{HD} = \gamma)P(\texttt{HD} = \gamma) \\ &= 0.85 \times 0.49 + 0.2 \times 0.51 = 0.5185. \end{aligned}$$

where $\gamma \in \{\texttt{Yes}, \texttt{No}\}$. Therefore, the posterior probability the person has heart disease is

$$\begin{aligned} P(\texttt{HD} = \texttt{Yes}|\texttt{BP} = \texttt{High}) &= \frac{P(\texttt{BP} = \texttt{High}|\texttt{HD} = \texttt{Yes})P(\texttt{HD} = \texttt{Yes})}{P(\texttt{BP} = \texttt{High})} \\ &= \frac{0.85 \times 0.49}{0.5185} = 0.8033. \end{aligned}$$

Similarly, $P(\texttt{HD} = \texttt{No}|\texttt{BP} = \texttt{High}) = 1 - 0.8033 = 0.1967$. Therefore, when a person has high blood pressure, it increases the risk of heart disease.

Case 3: High Blood Pressure, Healthy Diet, and Regular Exercise

Suppose we are told that the person exercises regularly and eats a healthy diet. How does the new information affect our diagnosis? With the new information, the posterior probability that the person has heart disease is

$$
\begin{aligned}
& P(\texttt{HD} = \texttt{Yes} | \texttt{BP} = \texttt{High}, D = \texttt{Healthy}, E = \texttt{Yes}) \\
= & \left[\frac{P(\texttt{BP} = \texttt{High} | \texttt{HD} = \texttt{Yes}, D = \texttt{Healthy}, E = \texttt{Yes})}{P(\texttt{BP} = \texttt{High} | D = \texttt{Healthy}, E = \texttt{Yes})} \right] \\
& \times P(\texttt{HD} = \texttt{Yes} | D = \texttt{Healthy}, E = \texttt{Yes}) \\
= & \frac{P(\texttt{BP} = \texttt{High} | \texttt{HD} = \texttt{Yes}) P(\texttt{HD} = \texttt{Yes} | D = \texttt{Healthy}, E = \texttt{Yes})}{\sum_\gamma P(\texttt{BP} = \texttt{High} | \texttt{HD} = \gamma) P(\texttt{HD} = \gamma | D = \texttt{Healthy}, E = \texttt{Yes})} \\
= & \frac{0.85 \times 0.25}{0.85 \times 0.25 + 0.2 \times 0.75} \\
= & \ 0.5862,
\end{aligned}
$$

while the probability that the person does not have heart disease is

$$P(\texttt{HD} = \texttt{No} | \texttt{BP} = \texttt{High}, D = \texttt{Healthy}, E = \texttt{Yes}) = 1 - 0.5862 = 0.4138.$$

The model therefore suggests that eating healthily and exercising regularly may reduce a person's risk of getting heart disease

4.5.4.4 Characterstics of BBN

- BBN provides an approach for capturing the prior knowledge of a particular domain using a graphical model. The network can also be used to encode causal dependencies among variables.
- Constructing the network can be time consuming and requires a large amount of effort. However, once the structure of the network has been determined, adding a new variable is quite straightforward.
- Bayesian networks are well suited to dealing with incomplete data. Instances with missing attributes can be handled by summing or integrating the probabilities over all possible values of the attribute.
- Because the data is combined probabilistically with prior knowledge, the method is quite robust to model overfitting.

Question bank

1. Explain the working of decision tree induction
2. Explain the hunts algorithm with example
3. Explain the methods for expressing attribute list conditions
4. Explain the measure for selecting the best split
5. Explain the algorithm for decision tree induction
6. Explain the characteristics of decision tree induction

7. Compare the performance of two classifiers
8. Explain the estimation of confidence interval for accuracy
9. Construct the decision tree for the following table1 using hunts algorithm.

10. Table1. The training data set

Instance	A	B	C	Class
1	T	T	T	+
2	F	T	T	-
3	T	F	T	+
4	F	F	T	-
5	T	T	F	+
6	F	T	F	+
7	T	F	F	+
8	F	F	F	-

11. Construct the decision tree for the following table1 using hunts algorithm.

12. Table1. The training data set

Instance	A	B	C	Class
1	0	0	0	+
2	0	0	1	+
3	0	1	0	+
4	0	1	1	-
5	1	0	0	+
6	1	0	0	+
7	1	1	0	-
8	1	0	1	+
9	1	1	0	-
10	1	1	0	-

13.

Consider a training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R_1: $A \longrightarrow +$ (covers 4 positive and 1 negative examples),
R_2: $B \longrightarrow +$ (covers 30 positive and 10 negative examples),
R_3: $C \longrightarrow +$ (covers 100 positive and 90 negative examples),

determine which is the best and worst candidate rule according to:

(a) Rule accuracy.
(b) FOIL's information gain.
(c) The likelihood ratio statistic.
(d) The Laplace measure.
(e) The m-estimate measure (with $k = 2$ and $p_+ = 0.2$).

14. Explain the working of rule based classifier
15. Explain rule based and class based classifier
16. Explain sequential covering algorithm with example
17. Explain rule growing strategy
18. Explain evaluation of rules
19. Explain Ripper Algorithm
20. Explain Indirect method for rule extraction
21. Explain the K nearest neighbor classification algorithm
22. Characteristics of K nearest neighbor classification
23. Explain Bayes Theorem
24. Explain Naïve Bayes classifier
25. Characterstics of Naïve Bayes classifier
26. Explain briefly Bayes error rate
27. Explain the Bayesian belief networks model representation
28. Explain the algorithm for generating the topology of a bayesian network
29. Given the Bayesian network shown in Figure 5.48, compute the following probabilities:
    (a) P(B = good, F = empty, G :=empty, S=yes).
    (b) P(B=bad, F= empty,G =not empty, S=no).
    (c) Given that the battery is bad, compute the probability that the car will start.

Table 5.12. Data set for Exercise 11.

Mileage	Engine	Air Conditioner	Number of Records with Car Value=Hi	Number of Records with Car Value=Lo
Hi	Good	Working	3	4
Hi	Good	Broken	1	2
Hi	Bad	Working	1	5
Hi	Bad	Broken	0	4
Lo	Good	Working	9	0
Lo	Good	Broken	5	1
Lo	Bad	Working	1	2
Lo	Bad	Broken	0	2

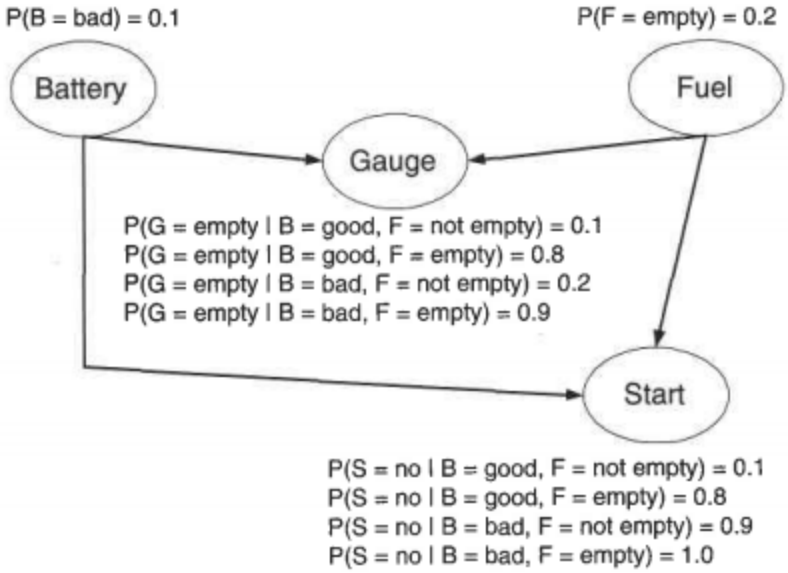


Figure 5.48. Bayesian belief network for Exercise 12.

Note: Study all the problems solved in class