

Module 4

CLASSIFICATION

Issues Regarding Classification and Prediction

Classification by Decision Tree Induction

- ▶ Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.
- ▶ A typical decision tree is shown in Figure 8.2. It represents the concept buys computer, that is, it predicts whether a customer at AllElectronics is likely to purchase a computer.
- ▶ Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals. Some decision tree algorithms produce only binary trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.
- ▶ A decision tree algorithm known as ID3

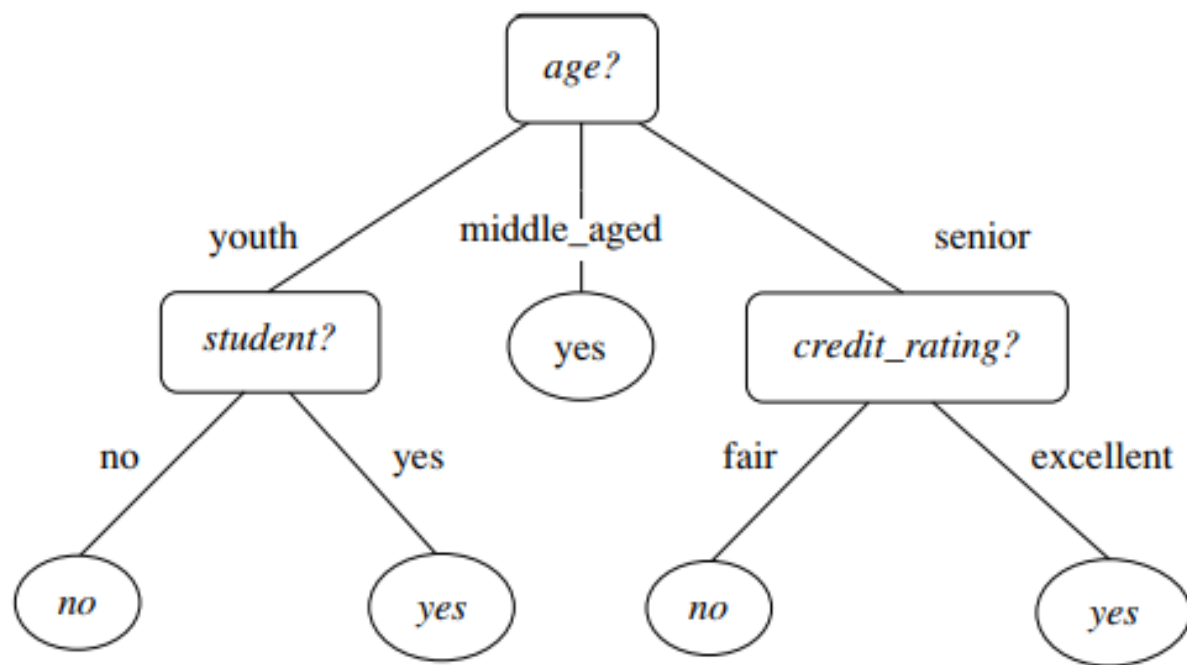
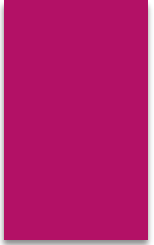


Figure 8.2 A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

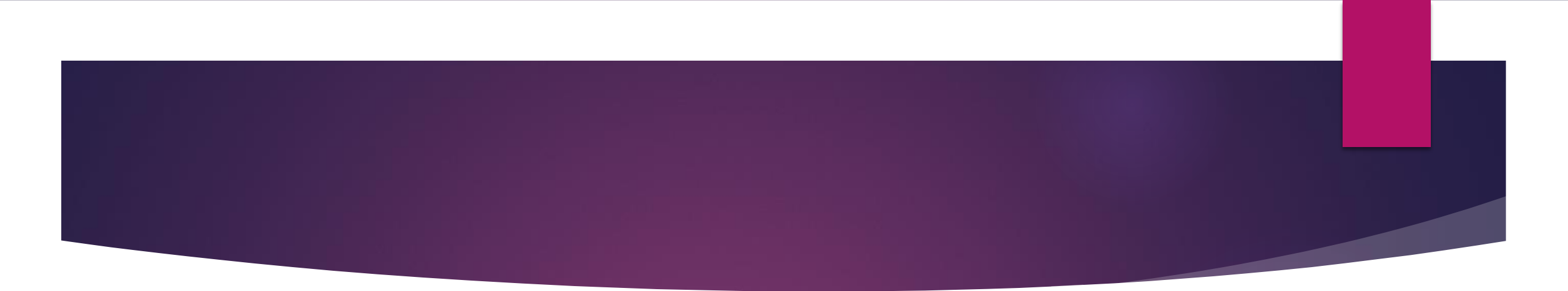


Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, D .

Input:

- Data partition, D , which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that “best” partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

- 
- ▶ When decision trees are built, many of the branches may reflect noise or outliers in the training data. Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.
 - ▶ When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data. Such methods typically use statistical measures to remove the least-reliable branches. An unpruned tree and a pruned version of it are shown in Figure 8.6.
 - ▶ Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.
 - ▶ There are two common approaches to tree pruning: prepruning and postpruning

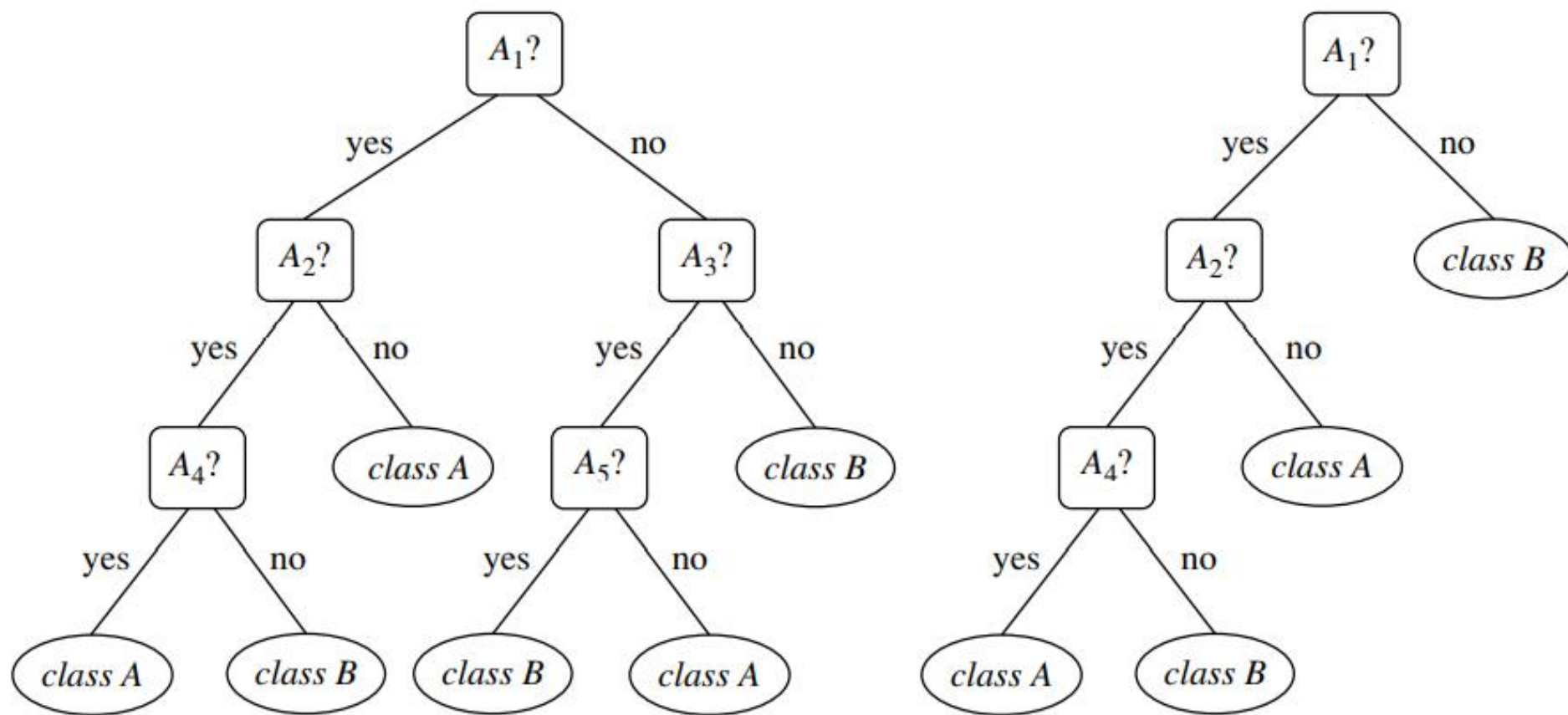
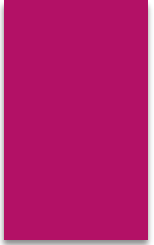
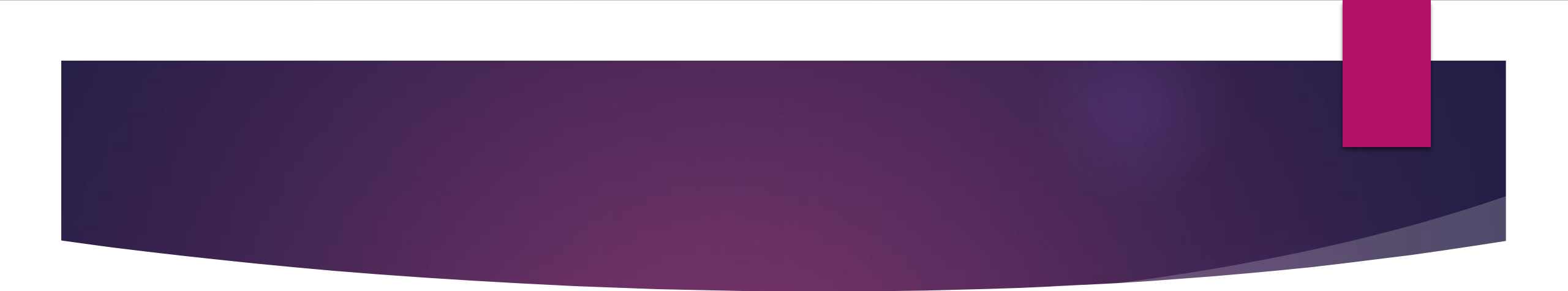


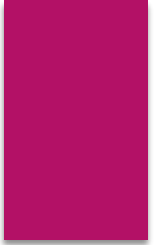
Figure 8.6 An unpruned decision tree and a pruned version of it.

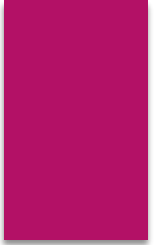
- 
- ✓ In the prepruning approach, a tree is “pruned” by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples. When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on, can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted.
 - ✓ The second and more common approach is postpruning, which removes subtrees from a “fully grown” tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.
 - ✓ For example, notice the subtree at node “A3?” in the unpruned tree of Figure 8.6. Suppose that the most common class within this subtree is “class B.” In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf “class B.”

Bayesian Classification

- ▶ Bayesian classifiers are statistical classifiers. They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.
- ▶ Bayesian classification is based on Bayes' theorem.
- ▶ Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.
- ▶ Naive Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class-conditional independence. It is made to simplify the computations involved and, in this sense, is considered "naive."

- 
- ▶ Let X be a data tuple. In Bayesian terms, X is considered “evidence.” As usual, it is described by measurements made on a set of n attributes. Let H be some hypothesis, such as that the data tuple X belongs to a specified class C . For classification problems, we want to determine $P(H|X)$, the probability that the hypothesis H holds given the “evidence” or observed data tuple X . In other words, we are looking for the probability that tuple X belongs to class C , given that we know the attribute description of X .
 - ▶ $P(H|X)$ is the posterior probability, or a posteriori probability, of H conditioned on X . For example, suppose our world of data tuples is confined to customers described by the attributes age and income, respectively, and that X is a 35-year-old customer with an income of \$40,000. Suppose that H is the hypothesis that our customer will buy a computer. Then $P(H|X)$ reflects the probability that customer X will buy a computer given that we know the customer’s age and income.

- 
- ❖ In contrast, $P(H)$ is the prior probability, or a priori probability, of H . For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter. The posterior probability, $P(H|X)$, is based on more information (e.g., customer information) than the prior probability, $P(H)$, which is independent of X .
 - ❖ Similarly, $P(X|H)$ is the posterior probability of X conditioned on H . That is, it is the probability that a customer, X , is 35 years old and earns \$40,000, given that we know the customer will buy a computer.

- 
- ❖ $P(X)$ is the prior probability of X . Using our example, it is the probability that a person from our set of customers is 35 years old and earns \$40,000.
 - ❖ “How are these probabilities estimated?” $P(H)$, $P(X | H)$, and $P(X)$ may be estimated from the given data, as we shall see below. Bayes’ theorem is useful in that it provides a way of calculating the posterior probability, $P(H | X)$, from $P(H)$, $P(X | H)$, and $P(X)$. Bayes’ theorem is

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}.$$

Rule-Based Classification

1. Using IF-THEN Rules for Classification

Rules are a good way of representing information or bits of knowledge. A rule-based classifier uses a set of IF-THEN rules for classification. An IF-THEN rule is an expression of the form

IF condition THEN conclusion.

An example is rule R1,

R1: IF age = youth AND student = yes THEN buys computer = yes.

The “IF”-part (or left-hand side) of a rule is known as the rule antecedent or precondition. The “THEN”-part (or right-hand side) is the rule consequent. In the rule antecedent, the condition consists of one or more attribute tests (such as age = youth, and student = yes) that are logically ANDed. The rule’s consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer). R1 can also be written as

R1: (age = youth) \wedge (student = yes) \Rightarrow (buys computer = yes).

- If the condition (that is, all of the attribute tests) in a rule antecedent holds true for a given tuple, we say that the rule antecedent is satisfied (or simply, that the rule is satisfied) and that the rule covers the tuple.
- A rule R can be assessed by its coverage and accuracy. Given a tuple, X , from a classlabeled data set, D , let n_{covers} be the number of tuples covered by R ; $n_{correct}$ be the number of tuples correctly classified by R ; and $|D|$ be the number of tuples in D . We can define the coverage and accuracy of R as

$$coverage(R) = \frac{n_{covers}}{|D|}$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}.$$

- That is, a rule's coverage is the percentage of tuples that are covered by the rule (i.e., whose attribute values hold true for the rule's antecedent). For a rule's accuracy, we look at the tuples that it covers and see what percentage of them the rule can correctly classify.

2. Rule Extraction from a Decision Tree

- ▶ Decision tree classifiers are a popular method of classification—it is easy to understand how decision trees work and they are known for their accuracy. Decision trees can become large and difficult to interpret.
- ▶ In this subsection, we look at how to build a rulebased classifier by extracting IF-THEN rules from a decision tree. In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand, particularly if the decision tree is very large.
- ▶ To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent (“IF” part). The leaf node holds the class prediction, forming the rule consequent (“THEN” part).

Extracting classification rules from a decision tree

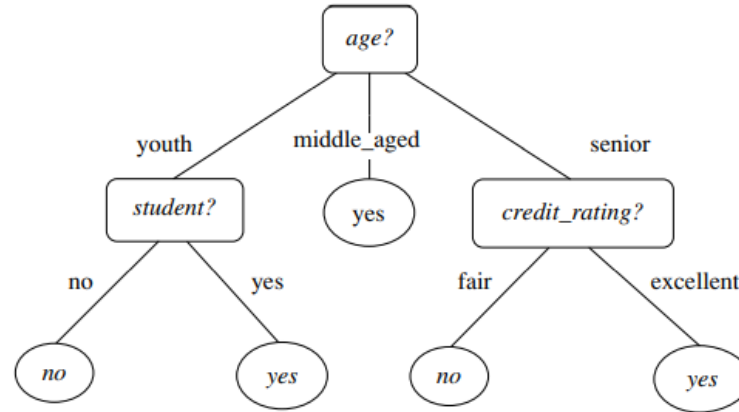


Figure 8.2 A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

Example 6.7 Extracting classification rules from a decision tree. The decision tree of Figure 6.2 can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Figure 6.2 are

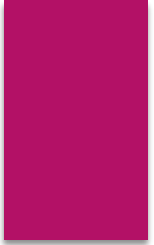
- R1: IF *age* = *youth* AND *student* = *no* THEN *buys_computer* = *no*
- R2: IF *age* = *youth* AND *student* = *yes* THEN *buys_computer* = *yes*
- R3: IF *age* = *middle_aged* THEN *buys_computer* = *yes*
- R4: IF *age* = *senior* AND *credit_rating* = *excellent* THEN *buys_computer* = *yes*
- R5: IF *age* = *senior* AND *credit_rating* = *fair* THEN *buys_computer* = *no*

Classification by Backpropagation

“What is backpropagation?”

Backpropagation is a neural network learning algorithm.

A neural network is a set of connected input/output units in which each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples. Neural network learning is also referred to as connectionist learning due to the connections between units.

- 
- ❖ Neural network involve long training times and are therefore more suitable for applications where this is feasible. They require a number of parameters that are typically best determined empirically such as the network topology or “structure.”
 - ❖ Neural networks have been criticized for their poor interpretability. For example, it is difficult for humans to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network. These features initially made neural networks less desirable for data mining.
 - ❖ Advantages of neural networks, however, include their high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained. They can be used when you may have little knowledge of the relationships between attributes and classes. They are well suited for continuous-valued inputs and outputs, unlike most decision tree algorithms.
 - ❖ They have been successful on a wide array of real-world data, including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text. Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process.
 - ❖ In addition, several techniques have been recently developed for rule extraction from trained neural networks. These factors contribute to the usefulness of neural networks for classification and numeric prediction in data mining.

A Multilayer Feed-Forward Neural Network

- ▶ There are many different kinds of neural networks and neural network algorithms. The most popular neural network algorithm is backpropagation.
- ▶ We will learn about multilayer feed-forward networks, the type of neural network on which the backpropagation algorithm performs.
- ▶ The backpropagation algorithm performs learning on a multilayer feed-forward neural network. It iteratively learns a set of weights for prediction of the class label of tuples. A multilayer feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer.

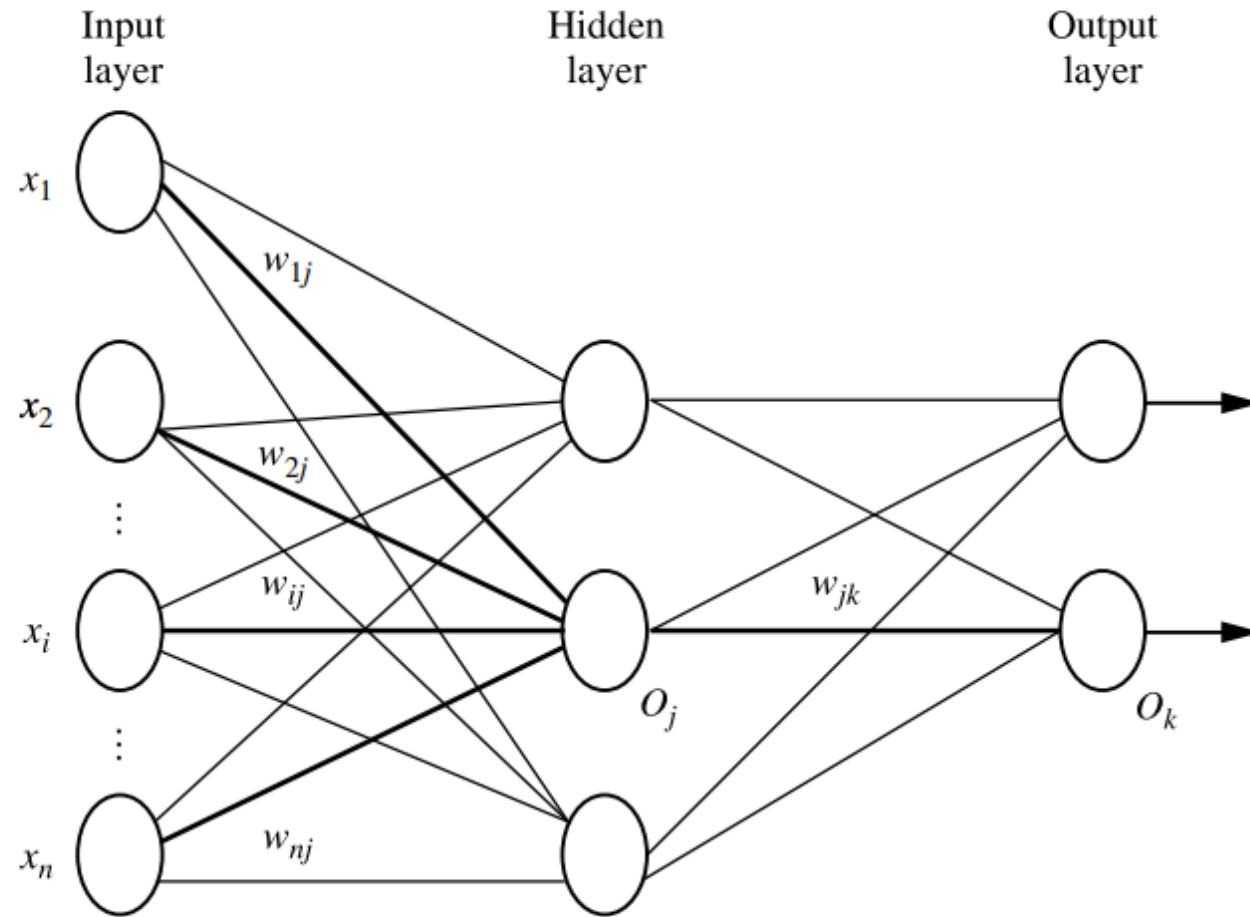


Figure 9.2 Multilayer feed-forward neural network.

- ✓ These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of “neuronlike” units, known as a hidden layer.
- ✓ The outputs of the hidden layer units can be input to another hidden layer, and so on.
- ✓ The number of hidden layers is arbitrary, although in practice, usually only one is used. The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network’s prediction for given tuples.
- ✓ The multilayer neural network shown in Figure has two layers of output units. Therefore, we say that it is a two-layer neural network. (The input layer is not counted because it serves only to pass the input values to the next layer.) Similarly, a network containing two hidden layers is called a three-layer neural network, and so on.
- ✓ It is a feed-forward network since none of the weights cycles back to an input unit or to a previous layer’s output unit. It is fully connected in that each unit provides input to each unit in the next forward layer.
- ✓ Each output unit takes, as input, a weighted sum of the outputs from units in the previous layer. It applies a nonlinear (activation) function to the weighted input

9.2.2 Defining a Network Topology

“How can I design the neural network’s topology?” Before training can begin, the user must decide on the network topology by specifying the number of units in the input layer, the number of hidden layers (if more than one), the number of units in each hidden layer, and the number of units in the output layer.

Normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. Typically, input values are normalized so as to fall between 0.0 and 1.0. Discrete-valued attributes may be encoded such that there is one input unit per domain value. For example, if an attribute A has three possible or known values, namely $\{a_0, a_1, a_2\}$, then we may assign three input units to represent A . That is, we may have, say, I_0, I_1, I_2 as input units. Each unit is initialized to 0. If $A = a_0$, then I_0 is set to 1 and the rest are 0. If $A = a_1$, then I_1 is set to 1 and the rest are 0, and so on.

Neural networks can be used for both classification (to predict the class label of a given tuple) and numeric prediction (to predict a continuous-valued output). For classification, one output unit may be used to represent two classes (where the value 1 represents one class, and the value 0 represents the other). If there are more than two classes, then one output unit per class is used. (See Section 9.7.1 for more strategies on multiclass classification.)

There are no clear rules as to the “best” number of hidden layer units. Network design is a trial-and-error process and may affect the accuracy of the resulting trained network. The initial values of the weights may also affect the resulting accuracy. Once a network has been trained and its accuracy is not considered acceptable, it is common to repeat the training process with a different network topology or a different set of initial weights. Cross-validation techniques for accuracy estimation (described in Chapter 8) can be used to help decide when an acceptable network has been found. A number of automated techniques have been proposed that search for a “good” network structure. These typically use a hill-climbing approach that starts with an initial structure that is selectively modified.

Sample calculations for learning by the backpropagation algorithm. Figure 9.5 shows a multilayer feed-forward neural network. Let the learning rate be 0.9. The initial weight and bias values of the network are given in Table 9.1, along with the first training tuple, $X = (1, 0, 1)$, with a class label of 1.

This example shows the calculations for backpropagation, given the first training tuple, X . The tuple is fed into the network, and the net input and output of each unit are computed. These values are shown in Table 9.2. The error of each unit is computed and propagated backward. The error values are shown in Table 9.3. The weight and bias updates are shown in Table 9.4

Refer PDF for detail explanation

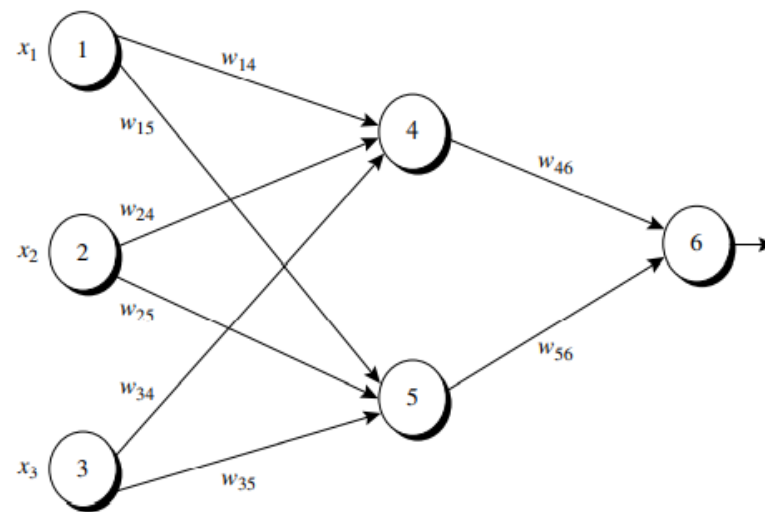


Figure 9.5 Example of a multilayer feed-forward neural network.

Table 9.1 Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Table 9.2 Net Input and Output Calculations

Unit, j	Net Input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Table 9.3 Calculation of the Error at Each Node

Unit, j	Err _{j}
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

Table 9.4 Calculations for Weight and Bias Updating

Weight or Bias	New Value
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

“How can we classify an unknown tuple using a trained network?” To classify an unknown tuple, X , the tuple is input to the trained network, and the net input and output of each unit are computed. (There is no need for computation and/or backpropagation of the error.) If there is one output node per class, then the output node with the highest value determines the predicted class label for X . If there is only one output node, then output values greater than or equal to 0.5 may be considered as belonging to the positive class, while values less than 0.5 may be considered negative.

Several variations and alternatives to the backpropagation algorithm have been proposed for classification in neural networks. These may involve the dynamic adjustment of the network topology and of the learning rate or other parameters, or the use of different error functions.

Support Vector Machines

Support Vector Machines

In this section, we study **support vector machines (SVMs)**, a method for the classification of both linear and nonlinear data. In a nutshell, an **SVM** is an algorithm that works as follows. It uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane (i.e., a “decision boundary” separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using *support vectors* (“essential” training tuples) and *margins* (defined by the support vectors). We will delve more into these new concepts later.

“I’ve heard that SVMs have attracted a great deal of attention lately. Why?” The first paper on support vector machines was presented in 1992 by Vladimir Vapnik and colleagues Bernhard Boser and Isabelle Guyon, although the groundwork for SVMs has been around since the 1960s (including early work by Vapnik and Alexei Chervonenkis on statistical learning theory). Although the training time of even the fastest SVMs can be extremely slow, they are highly accurate, owing to their ability to model complex nonlinear decision boundaries. They are much less prone to overfitting than other methods. The support vectors found also provide a compact description of the learned model. SVMs can be used for numeric prediction as well as classification. They have been applied to a number of areas, including handwritten digit recognition, object recognition, and speaker identification, as well as benchmark time-series prediction tests.

9.3.1 The Case When the Data Are Linearly Separable

To explain the mystery of SVMs, let's first look at the simplest case—a two-class problem where the classes are linearly separable. Let the data set D be given as $(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), \dots, (\mathbf{X}_{|D|}, y_{|D|})$, where \mathbf{X}_i is the set of training tuples with associated class labels, y_i . Each y_i can take one of two values, either $+1$ or -1 (i.e., $y_i \in \{+1, -1\}$), corresponding to the classes *buys_computer = yes* and *buys_computer = no*, respectively. To aid in visualization, let's consider an example based on two input attributes, A_1 and A_2 , as shown in Figure 9.7. From the graph, we see that the 2-D data are **linearly separable** (or “linear,” for short), because a straight line can be drawn to separate all the tuples of class $+1$ from all the tuples of class -1 .

There are an infinite number of separating lines that could be drawn. We want to find the “best” one, that is, one that (we hope) will have the minimum classification error on previously unseen tuples. How can we find this best line? Note that if our data were 3-D (i.e., with three attributes), we would want to find the best separating *plane*. Generalizing to n dimensions, we want to find the best *hyperplane*. We will use “hyperplane” to refer to the decision boundary that we are seeking, regardless of the number of input attributes. So, in other words, how can we find the best hyperplane?

An SVM approaches this problem by searching for the **maximum marginal hyperplane**. Consider Figure 9.8, which shows two possible separating hyperplanes and their associated margins. Before we get into the definition of margins, let's take an intuitive look at this figure. Both hyperplanes can correctly classify all the given data tuples. Intuitively, however, we expect the hyperplane with the larger margin to be more accurate at classifying future data tuples than the hyperplane with the smaller margin. This is why (during the learning or training phase) the SVM searches for the hyperplane with the largest margin, that is, the *maximum marginal hyperplane* (MMH). The associated margin gives the largest separation between classes.

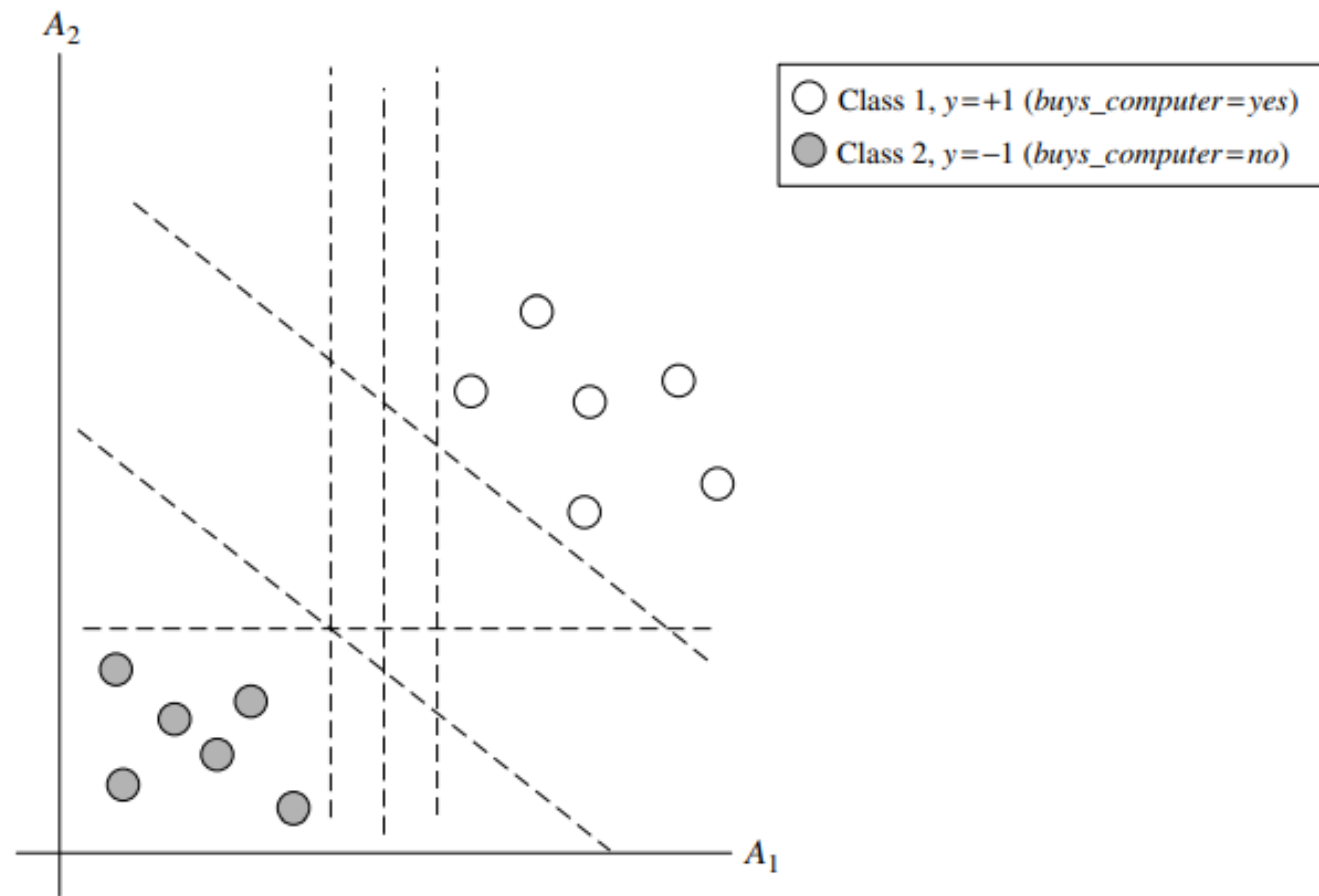


Figure 9.7 The 2-D training data are linearly separable. There are an infinite number of possible separating hyperplanes or “decision boundaries,” some of which are shown here as dashed lines. Which one is best?

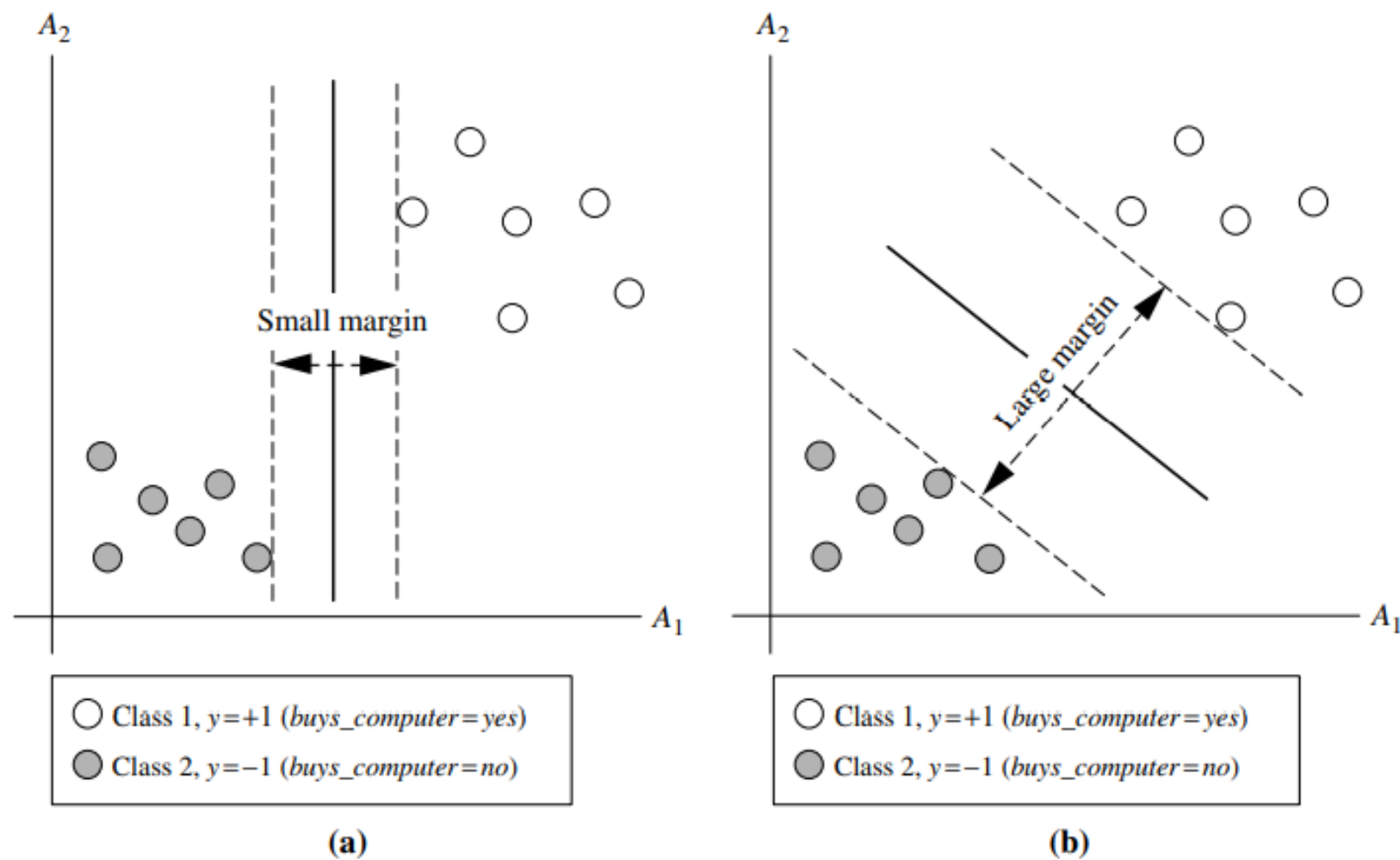


Figure 9.8 Here we see just two possible separating hyperplanes and their associated margins. Which one is better? The one with the larger margin (b) should have greater generalization accuracy.

9.3.2 The Case When the Data Are Linearly Inseparable

In Section 9.3.1 we learned about linear SVMs for classifying linearly separable data, but what if the data are not linearly separable, as in Figure 9.10? In such cases, no straight line can be found that would separate the classes. The linear SVMs we studied would not be able to find a feasible solution here. Now what?

The good news is that the approach described for linear SVMs can be extended to create *nonlinear SVMs* for the classification of *linearly inseparable data* (also called *non-linearly separable data*, or *nonlinear data* for short). Such SVMs are capable of finding nonlinear decision boundaries (i.e., nonlinear hypersurfaces) in input space.

“So,” you may ask, “*how can we extend the linear approach?*” We obtain a nonlinear SVM by extending the approach for linear SVMs as follows. There are two main steps. In the first step, we transform the original input data into a higher dimensional space using a nonlinear mapping. Several common nonlinear mappings can be used in this step, as we will further describe next. Once the data have been transformed into the new higher space, the second step searches for a linear separating hyperplane in the new space. We again end up with a quadratic optimization problem that can be solved using the linear SVM formulation. The maximal marginal hyperplane found in the new space corresponds to a nonlinear separating hypersurface in the original space.

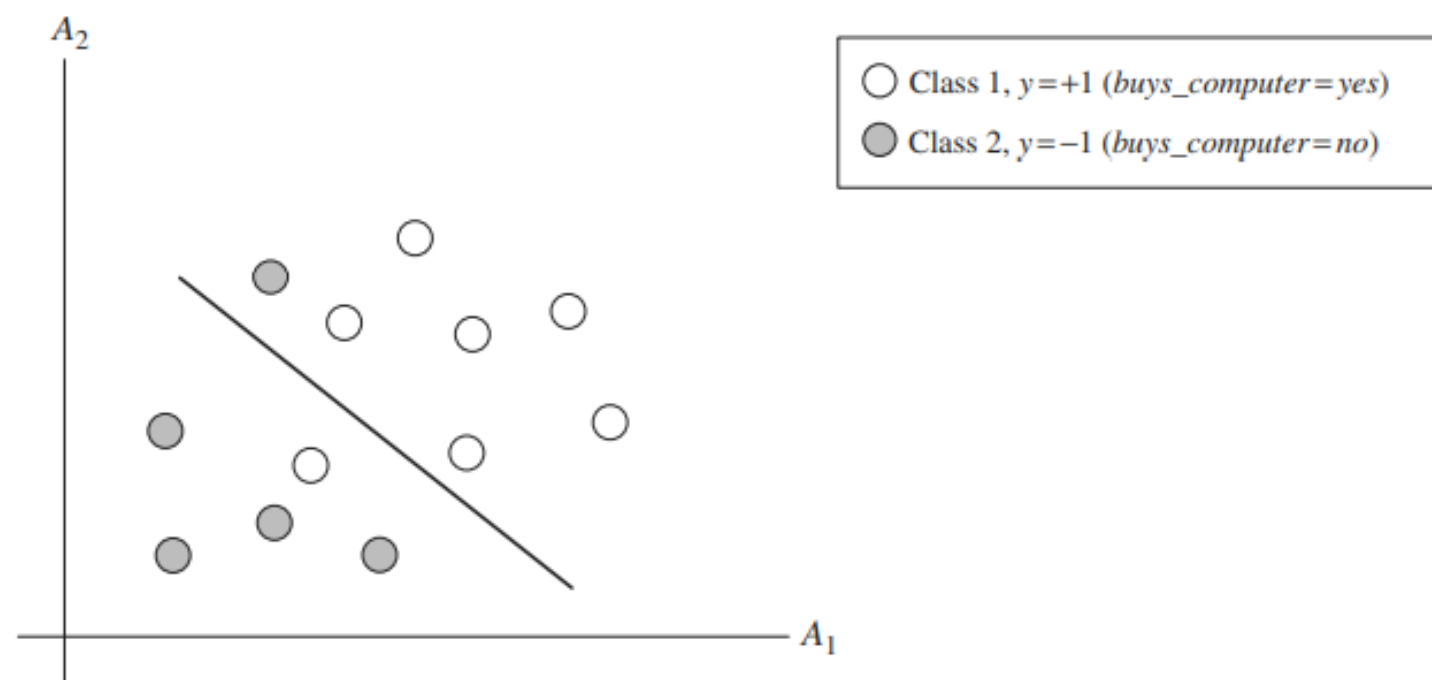


Figure 9.10 A simple 2-D case showing linearly inseparable data. Unlike the linear separable data of Figure 9.7, here it is not possible to draw a straight line to separate the classes. Instead, the decision boundary is nonlinear.

Associative Classification

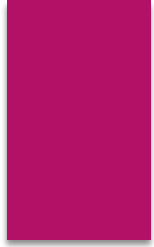
- ▶ Association rules are mined in a two-step process consisting of frequent itemset mining, followed by rule generation.
- ▶ The first step searches for patterns of attribute-value pairs that occur repeatedly in a data set, where each attribute-value pair is considered an item. The resulting attribute value pairs form frequent itemsets. The second step analyzes the frequent itemsets in order to generate association rules.
- ▶ All association rules must satisfy certain criteria regarding their “accuracy” (or confidence) and the proportion of the data set that they actually represent (referred to as support).
- ▶ For example, the following is an association rule mined from a data set, D, shown with its confidence and support

age = youth \wedge credit = OK \Rightarrow buys computer = yes [support = 20%, confidence = 93%]

where “ \wedge ” represents a logical “AND.” We will say more about confidence and support in a minute.

Lazy Learners

The classification methods discussed so far in this chapter—decision tree induction, Bayesian classification, rule-based classification, classification by backpropagation, support vector machines, and classification based on association rule mining—are all examples of *eager learners*. **Eager learners**, when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify. We can think of the learned model as being ready and eager to classify previously unseen tuples.



Imagine a contrasting lazy approach, in which the learner instead waits until the last minute before doing any model construction in order to classify a given test tuple. That is, when given a training tuple, a **lazy learner** simply stores it (or does only a little minor processing) and waits until it is given a test tuple. Only when it sees the test tuple does it perform generalization in order to classify the tuple based on its similarity to the stored training tuples. Unlike eager learning methods, lazy learners do less work when a training tuple is presented and more work when making a classification or prediction. Because lazy learners store the training tuples or “instances,” they are also referred to as **instance-based learners**, even though all learning is essentially based on instances.

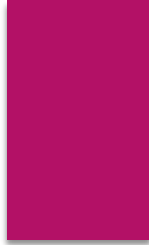
When making a classification or prediction, lazy learners can be computationally expensive. They require efficient storage techniques and are well-suited to implementation on parallel hardware. They offer little explanation or insight into the structure of the data. Lazy learners, however, naturally support incremental learning. They are able to model complex decision spaces having hyperpolygonal shapes that may not be as easily describable by other learning algorithms (such as hyper-rectangular shapes modeled by decision trees). In this section, we look at two examples of lazy learners: *k-nearest-neighbor classifiers* and *case-based reasoning classifiers*.

1. k-Nearest-Neighbor Classifiers

Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n -dimensional space. In this way, all of the training tuples are stored in an n -dimensional pattern space. When given an unknown tuple, a **k -nearest-neighbor classifier** searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k “nearest neighbors” of the unknown tuple.

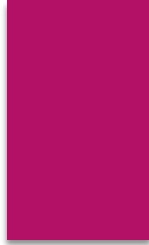
“Closeness” is defined in terms of a distance metric, such as Euclidean distance. The Euclidean distance between two points or tuples, say, $X_1 = (x_{11}, x_{12}, \dots, x_{1n})$ and $X_2 = (x_{21}, x_{22}, \dots, x_{2n})$, is

$$\text{dist}(X_1, X_2) = \sqrt{\sum_{i=1}^n (x_{1i} - x_{2i})^2}. \quad (6.45)$$



In other words, for each numeric attribute, we take the difference between the corresponding values of that attribute in tuple X_1 and in tuple X_2 , square this difference, and accumulate it. The square root is taken of the total accumulated distance count. Typically, we normalize the values of each attribute before using Equation (6.45). This helps prevent attributes with initially large ranges (such as *income*) from outweighing attributes with initially smaller ranges (such as binary attributes). Min-max normalization, for example, can be used to transform a value v of a numeric attribute A to v' in the range $[0, 1]$ by computing

$$v' = \frac{v - \min_A}{\max_A - \min_A}, \quad (6.46)$$



where \min_A and \max_A are the minimum and maximum values of attribute A . Chapter 2 describes other methods for data normalization as a form of data transformation.

For k -nearest-neighbor classification, the unknown tuple is assigned the most common class among its k nearest neighbors. When $k = 1$, the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. Nearest-neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the average value of the real-valued labels associated with the k nearest neighbors of the unknown tuple.

2. Case-Based Reasoning

Case-based reasoning (CBR) classifiers use a database of problem solutions to solve new problems. Unlike nearest-neighbor classifiers, which store training tuples as points in Euclidean space, CBR stores the tuples or “cases” for problem solving as complex symbolic descriptions. Business applications of CBR include problem resolution for customer service help desks, where cases describe product-related diagnostic problems. CBR has also been applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively. Medical education is another area for CBR, where patient case histories and treatments are used to help diagnose and treat new patients.

When given a new case to classify, a case-based reasoner will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. If no identical case is found, then the case-based reasoner will search for training cases having components that are similar to those of the new case. Conceptually, these training cases may be considered as neighbors of the new case. If cases are represented as graphs, this involves searching for subgraphs that are similar to subgraphs within the new case. The case-based reasoner tries to combine the solutions of the neighboring training cases in order to propose a solution for the new case. If incompatibilities arise with the individual solutions, then backtracking to search for other solutions may be necessary. The case-based reasoner may employ background knowledge and problem-solving strategies in order to propose a feasible combined solution.

Other Classification Methods

- ▶ Genetic Algorithms
- ▶ Rough Set Approach
- ▶ Fuzzy Set Approaches

.1 Genetic Algorithms

Genetic algorithms attempt to incorporate ideas of natural evolution. In general, genetic learning starts as follows. An initial **population** is created consisting of randomly generated rules. Each rule can be represented by a string of bits. As a simple example, suppose that samples in a given training set are described by two Boolean attributes, A_1 and A_2 , and that there are two classes, C_1 and C_2 . The rule “*IF A_1 AND NOT A_2 THEN C_2* ” can be encoded as the bit string “100,” where the two leftmost bits represent attributes A_1 and A_2 , respectively, and the rightmost bit represents the class. Similarly, the rule “*IF NOT A_1 AND NOT A_2 THEN C_1* ” can be encoded as “001.” If an attribute has k values, where $k > 2$, then k bits may be used to encode the attribute’s values. Classes can be encoded in a similar fashion.

Based on the notion of survival of the fittest, a new population is formed to consist of the *fittest* rules in the current population, as well as *offspring* of these rules. Typically, the **fitness** of a rule is assessed by its classification accuracy on a set of training samples.

Offspring are created by applying genetic operators such as crossover and mutation. In **crossover**, substrings from pairs of rules are swapped to form new pairs of rules. In **mutation**, randomly selected bits in a rule’s string are inverted.

The process of generating new populations based on prior populations of rules continues until a population, P , evolves where each rule in P satisfies a prespecified fitness threshold.

Genetic algorithms are easily parallelizable and have been used for classification as well as other optimization problems. In data mining, they may be used to evaluate the fitness of other algorithms.

.2 Rough Set Approach

Rough set theory can be used for classification to discover structural relationships within imprecise or noisy data. It applies to discrete-valued attributes. Continuous-valued attributes must therefore be discretized before its use.

Rough set theory is based on the establishment of **equivalence classes** within the given training data. All of the data tuples forming an equivalence class are indiscernible, that is, the samples are identical with respect to the attributes describing the data. Given real-world data, it is common that some classes cannot be distinguished in terms of the available attributes. Rough sets can be used to approximately or “roughly” define such classes. A rough set definition for a given class, C , is approximated by two sets—a **lower approximation** of C and an **upper approximation** of C . The lower approximation of C consists of all of the data tuples that, based on the knowledge of the attributes, are certain to belong to C without ambiguity. The upper approximation of C consists of all of the tuples that, based on the knowledge of the attributes, cannot be described as not belonging to C . The lower and upper approximations for a class C are shown in Figure 6.24, where each rectangular region represents an equivalence class. Decision rules can be generated for each class. Typically, a decision table is used to represent the rules.

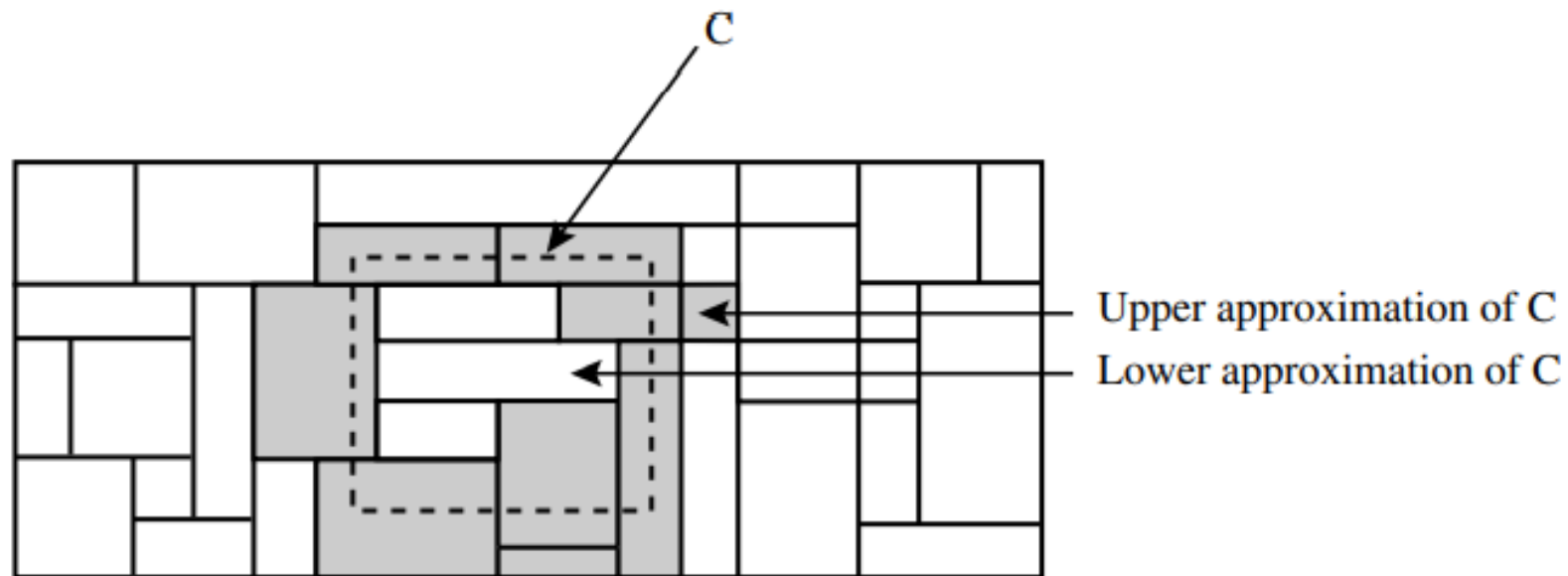


Figure 6.24 A rough set approximation of the set of tuples of the class C using lower and upper approximation sets of C . The rectangular regions represent equivalence classes.

3 Fuzzy Set Approaches

Instead, we can discretize *income* into categories such as $\{low_income, medium_income, high_income\}$, and then apply **fuzzy logic** to allow “fuzzy” thresholds or boundaries to be defined for each category (Figure 6.25). Rather than having a precise cutoff between categories, fuzzy logic uses truth values between 0.0 and 1.0 to represent the degree of membership that a certain value has in a given category. Each category then represents a **fuzzy set**. Hence, with fuzzy logic, we can capture the notion that an income of \$49,000 is, more or less, high, although not as high as an income of \$50,000. Fuzzy logic systems typically provide graphical tools to assist users in converting attribute values to fuzzy truth values.

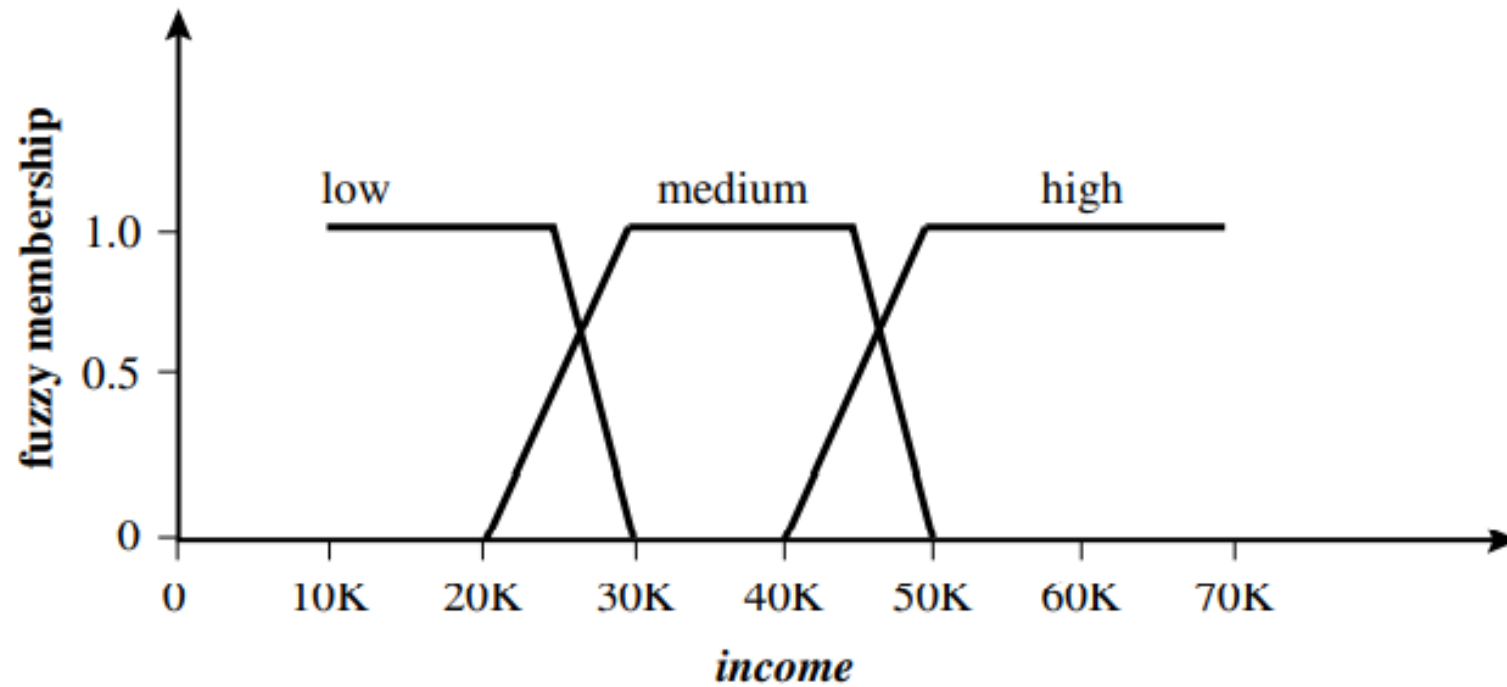


Figure 6.25 Fuzzy truth values for *income*, representing the degree of membership of *income* values with respect to the categories $\{low, medium, high\}$. Each category represents a fuzzy set. Note that a given income value, x , can have membership in more than one fuzzy set. The membership values of x in each fuzzy set do not have to total to 1.

Prediction

“What if we would like to predict a continuous value, rather than a categorical label?”

Numeric prediction is the task of predicting continuous (or ordered) values for given input. For example, we may wish to predict the salary of college graduates with 10 years of work experience, or the potential sales of a new product given its price. By far, the most widely used approach for numeric prediction (hereafter referred to as prediction) is **regression**, a statistical methodology that was developed by Sir Frances Galton (1822–1911), a mathematician who was also a cousin of Charles Darwin. In fact, many texts use the terms “regression” and “numeric prediction” synonymously. However, as we have seen, some classification techniques (such as backpropagation, support vector machines, and k -nearest-neighbor classifiers) can be adapted for prediction. In this section, we discuss the use of regression techniques for prediction.

Example 6.11 Straight-line regression using the method of least squares. Table 6.7 shows a set of paired data where x is the number of years of work experience of a college graduate and y is the

Table 6.7 Salary data.

x years experience	y salary (in \$1000s)
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83

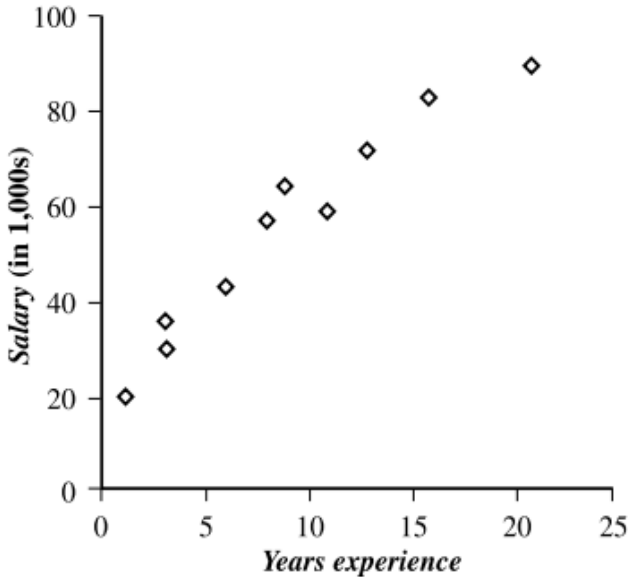


Figure 6.26 Plot of the data in Table 6.7 for Example 6.11. Although the points do not fall on a straight line, the overall pattern suggests a linear relationship between x (years experience) and y (salary).

corresponding salary of the graduate. The 2-D data can be graphed on a *scatter plot*, as in Figure 6.26. The plot suggests a linear relationship between the two variables, x and y . We model the relationship that salary may be related to the number of years of work experience with the equation $y = w_0 + w_1x$.

Given the above data, we compute $\bar{x} = 9.1$ and $\bar{y} = 55.4$. Substituting these values into Equations (6.50) and (6.51), we get

$$w_1 = \frac{(3 - 9.1)(30 - 55.4) + (8 - 9.1)(57 - 55.4) + \cdots + (16 - 9.1)(83 - 55.4)}{(3 - 9.1)^2 + (8 - 9.1)^2 + \cdots + (16 - 9.1)^2} = 3.5$$

$$w_0 = 55.4 - (3.5)(9.1) = 23.6$$

Thus, the equation of the least squares line is estimated by $y = 23.6 + 3.5x$. Using this equation, we can predict that the salary of a college graduate with, say, 10 years of experience is \$58,600. ■

6.11.2 Nonlinear Regression

“How can we model data that does not show a linear dependence? For example, what if a given response variable and predictor variable have a relationship that may be modeled by a polynomial function?” Think back to the straight-line linear regression case above where dependent response variable, y , is modeled as a linear function of a single independent predictor variable, x . What if we can get a more accurate model using a nonlinear model, such as a parabola or some other higher-order polynomial? **Polynomial regression** is often of interest when there is just one predictor variable. It can be modeled by adding polynomial terms to the basic linear model. By applying transformations to the variables, we can convert the nonlinear model into a linear one that can then be solved by the method of least squares.

Example 6.12 Transformation of a polynomial regression model to a linear regression model. Consider a cubic polynomial relationship given by

$$y = w_0 + w_1x + w_2x^2 + w_3x^3. \quad (6.53)$$

To convert this equation to linear form, we define new variables:

$$x_1 = x \quad x_2 = x^2 \quad x_3 = x^3 \quad (6.54)$$

Equation (6.53) can then be converted to linear form by applying the above assignments, resulting in the equation $y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$, which is easily solved by the method of least squares using software for regression analysis. Note that polynomial regression is a special case of multiple regression. That is, the addition of high-order terms like x^2 , x^3 , and so on, which are simple functions of the single variable, x , can be considered equivalent to adding new independent variables. ■

Accuracy and Error measures

- ▶ Now that you may have trained a classifier or predictor, there may be many questions going through your mind.
- ▶ For example, suppose you used data from previous sales to train a classifier to predict customer purchasing behavior. You would like an estimate of how accurately the classifier can predict the purchasing behavior of future customers, that is, future customer data on which the classifier has not been trained. You may even have tried different methods to build more than one classifier (or predictor) and now wish to compare their accuracy. But what is accuracy? How can we estimate it? Are there strategies for increasing the accuracy of a learned model? These questions are addressed in the next few sections.

Confusion Matrix

		Predicted class	
		C_1	C_2
Actual class	C_1	true positives	false negatives
	C_2	false positives	true negatives

Figure 6.28 A confusion matrix for positive and negative tuples.

The confusion matrix is a useful tool for analyzing how well your classifier can recognize tuples of different classes. A confusion matrix for two classes is shown in Figure 6.27. Given m classes, a confusion matrix is a table of at least size m by m .

- ▶ True positives refer to the positive tuples that were correctly labeled by the classifier, while true negatives are the negative tuples that were correctly labeled by the classifier.
- ▶ False positives are the negative tuples that were incorrectly labeled (e.g., tuples of class buys computer = no for which the classifier predicted buys computer = yes).
- ▶ Similarly false negatives are the positive tuples that were incorrectly labeled (e.g., tuples of class buys computer = yes for which the classifier predicted buys computer = no).

Evaluating the Accuracy of a Classifier or Predictor

- ▶ How can we use the above measures to obtain a reliable estimate of classifier accuracy (or predictor accuracy in terms of error)?
- ▶ Holdout, random subsampling, crossvalidation, and the bootstrap are common techniques for asses randomly sampled partitions of the given data. The use of such techniques to estimate accuracy increases the overall computation time, yet is useful for model selection.

6.13.1 Holdout Method and Random Subsampling

The **holdout** method is what we have alluded to so far in our discussions about accuracy. In this method, the given data are randomly partitioned into two independent sets, a *training set* and a *test set*. Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set. The training set is used to derive the model, whose accuracy is estimated with the test set (Figure 6.29). The estimate is pessimistic because only a portion of the initial data is used to derive the model.

Random subsampling is a variation of the holdout method in which the holdout method is repeated k times. The overall accuracy estimate is taken as the average of the accuracies obtained from each iteration. (For prediction, we can take the average of the predictor error rates.)

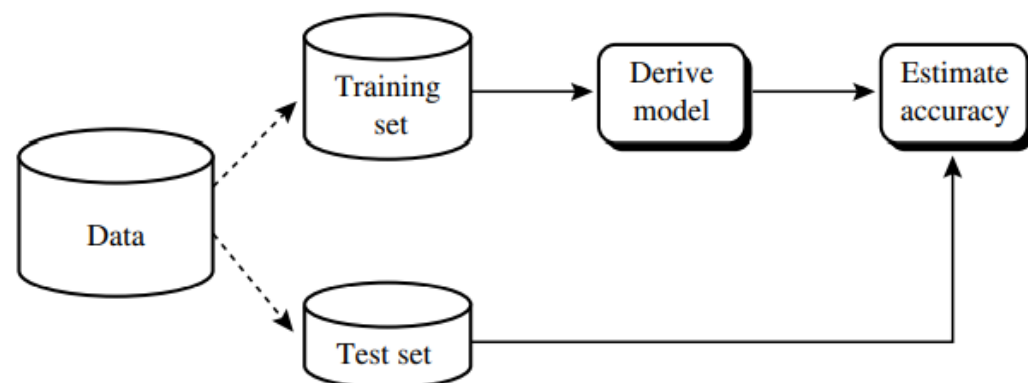


Figure 6.29 Estimating accuracy with the holdout method.

6.13.2 Cross-validation

In **k -fold cross-validation**, the initial data are randomly partitioned into k mutually exclusive subsets or “folds,” D_1, D_2, \dots, D_k , each of approximately equal size. Training and testing is performed k times. In iteration i , partition D_i is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets D_2, \dots, D_k collectively serve as the training set in order to obtain a first model, which is tested on D_1 ; the second iteration is trained on subsets D_1, D_3, \dots, D_k and tested on D_2 ; and so on. Unlike the holdout and random subsampling methods above, here, each sample is used the same number of times for training and once for testing. For classification, the accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of tuples in the initial data. For prediction, the error estimate can be computed as the total loss from the k iterations, divided by the total number of initial tuples.

Leave-one-out is a special case of k -fold cross-validation where k is set to the number of initial tuples. That is, only one sample is “left out” at a time for the test set. In **stratified cross-validation**, the folds are stratified so that the class distribution of the tuples in each fold is approximately the same as that in the initial data.

In general, stratified 10-fold cross-validation is recommended for estimating accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

6.13.3 Bootstrap

Unlike the accuracy estimation methods mentioned above, the **bootstrap method** samples the given training tuples uniformly *with replacement*. That is, each time a tuple is selected, it is equally likely to be selected again and readded to the training set. For instance, imagine a machine that randomly selects tuples for our training set. In *sampling with replacement*, the machine is allowed to select the same tuple more than once.

There are several bootstrap methods. A commonly used one is the **.632 bootstrap**, which works as follows. Suppose we are given a data set of d tuples. The data set is sampled d times, with replacement, resulting in a *bootstrap sample* or training set of d samples. It is very likely that some of the original data tuples will occur more than once in this sample. The data tuples that did not make it into the training set end up forming the test set. Suppose we were to try this out several times. As it turns out, on average, 63.2% of the original data tuples will end up in the bootstrap, and the remaining 36.8% will form the test set (hence, the name, .632 bootstrap.)

Ensemble Methods

Ensemble Methods—Increasing the Accuracy

In Section 6.3.3, we saw how pruning can be applied to decision tree induction to help improve the accuracy of the resulting decision trees. Are there *general* strategies for improving classifier and predictor accuracy?

The answer is yes. *Bagging* and *boosting* are two such techniques (Figure 6.30). They are examples of **ensemble methods**, or methods that use a *combination* of models. Each combines a series of k learned models (classifiers or predictors), M_1, M_2, \dots, M_k , with the aim of creating an improved composite model, M^* . Both bagging and boosting can be used for classification as well as prediction.



Bagging

We first take an intuitive look at how bagging works as a method of increasing accuracy. For ease of explanation, we will assume at first that our model is a classifier. Suppose that you are a patient and would like to have a diagnosis made based on your symptoms. Instead of asking one doctor, you may choose to ask several. If a certain diagnosis occurs more than any of the others, you may choose this as the final or best diagnosis. That is, the final diagnosis is made based on a majority vote, where each doctor gets an equal vote. Now replace each doctor by a classifier, and you have the basic idea behind bagging. Intuitively, a majority vote made by a large group of doctors may be more reliable than a majority vote made by a small group.

Boosting

- ✓ We now look at the ensemble method of boosting. As in the previous section, suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnoses they have made. The final diagnosis is then a combination of the weighted diagnoses. This is the essence behind boosting.
- ✓ In boosting, weights are assigned to each training tuple. A series of k classifiers is iteratively learned. After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1} , to “pay more attention” to the training tuples that were misclassified by M_i . The final boosted classifier, M^* , combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy. The boosting algorithm can be extended for the prediction of continuous values.
- ✓ Adaboost is a popular boosting algorithm. Suppose we would like to boost the accuracy of some learning method.