Contents

Chapter 7

Classification and Prediction

Databases are rich with hidden information that can be used for making intelligent business decisions. Classification and prediction are two forms of data analysis which can be used to extract models describing important data classes or to predict future data trends. Whereas *classification* predicts categorical labels (or discrete values), *prediction* models continuous-valued functions. For example, a classification model may be built to categorize bank loan applications as either safe or risky, while a prediction model may be built to predict the expenditures of potential customers on computer equipment given their income and occupation. Many classification and prediction methods have been proposed by researchers in machine learning, expert systems, statistics, and neurobiology. Most algorithms are memory resident, typically assuming a small data size. Recent database mining research has built on such work, developing scalable classification and prediction techniques capable of handling large, disk resident data. These techniques often consider parallel and distributed processing.

In this chapter, you will learn basic techniques for data classification such as decision tree induction, Bayesian classification and Bayesian belief networks, and neural networks. The integration of data warehousing technology with classification is also discussed, as well as association-based classification. Other approaches to classification, such as k-nearest neighbor classifiers, case-based reasoning, genetic algorithms, rough sets, and fuzzy logic techniques are introduced. Methods for prediction, including linear, nonlinear, and generalized linear regression models are briefly discussed. Where applicable, you will learn of modifications, extensions and optimizations to these techniques for their application to data classification and prediction for large databases.

7.1 What is classification? What is prediction?

Data classification is a two step process (Figure 7.1). In the first step, a model is built describing a predetermined set of data classes or concepts. The model is constructed by analyzing database tuples described by attributes. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the **class label attribute**. In the context of classification, data tuples are also referred to as *samples, examples*, or *objects*. The data tuples analyzed to build the model collectively form the **training data set**. The individual tuples making up the training set are referred to as **training samples** and are randomly selected from the sample population. Since the class label of each training sample *is provided*, this step is also known as **supervised learning** (i.e., the learning of the model is 'supervised' in that it is told to which class each training sample belongs). It contrasts with **unsupervised learning** (or **clustering**), in which the class labels of the training samples are not known, and the number or set of classes to be learned may not be known in advance. Clustering is the topic of Chapter 8.

Typically, the learned model is represented in the form of classification rules, decision trees, or mathematical formulae. For example, given a database of customer credit information, classification rules can be learned to identify customers as having either excellent or fair credit ratings (Figure 7.1a). The rules can be used to categorize future data samples, as well as provide a better understanding of the database contents.

In the second step (Figure 7.1b), the model is used for classification. First, the predictive accuracy of the model (or classifier) is estimated. Section 7.9 of this chapter describes several methods for estimating classifier accuracy. The **holdout method** is a simple technique which uses a **test set** of class-labeled samples. These samples are

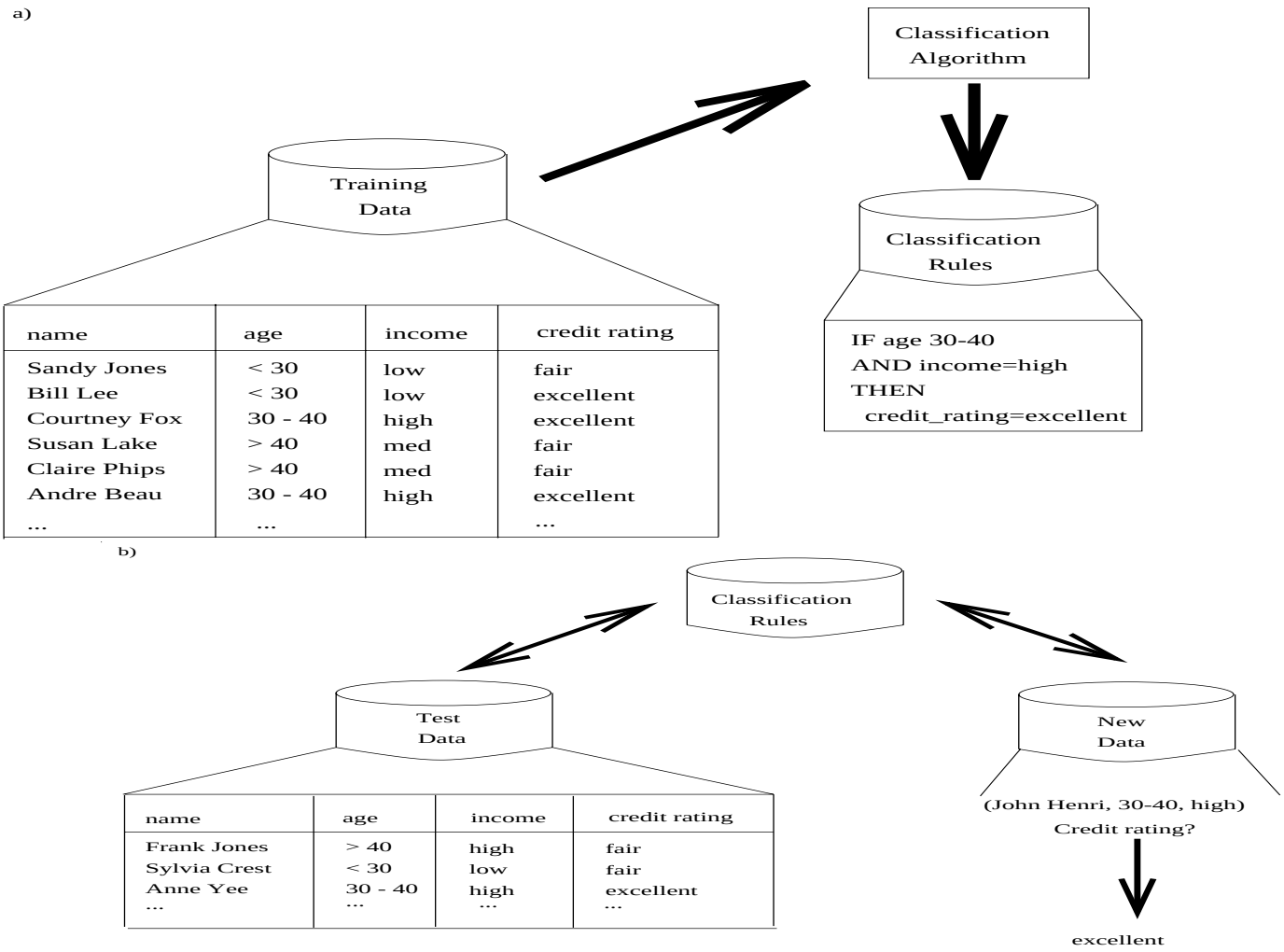


Figure 7.1: The data classification process: a) *Learning:* Training data are analyzed by a classification algorithm. Here, the class label attribute is *credit_rating*, and the learned model or classifier is represented in the form of classification rules. b) *Classification:* Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

randomly selected and are independent of the training samples. The **accuracy** of a model on a given test set is the percentage of test set samples that are correctly classified by the model. For each test sample, the known class label is compared with the learned model's class prediction for that sample. Note that if the accuracy of the model were estimated based on the training data set, this estimate could be optimistic since the learned model tends to **overfit** the data (that is, it may have incorporated some particular anomalies of the training data which are not present in the overall sample population). Therefore, a test set is used.

If the accuracy of the model is considered acceptable, the model can be used to classify future data tuples or objects for which the class label is not known. (Such data are also referred to in the machine learning literature as "unknown" or "previously unseen" data). For example, the classification rules learned in Figure 7.1a from the analysis of data from existing customers can be used to predict the credit rating of new or future (i.e., previously unseen) customers.

"How is prediction different from classification?" **Prediction** can be viewed as the construction and use of a model to assess the class of an unlabeled object, or to assess the value or value ranges of an attribute that a given object is likely to have. In this view, classification and regression are the two major types of prediction problems where classification is used to predict discrete or nominal values, while regression is used to predict continuous or

ordered values. In our view, however, we refer to the use of predication to predict class labels as *classification* and the use of predication to predict continuous values (e.g., using regression techniques) as *prediction*. This view is commonly accepted in data mining.

Classification and prediction have numerous applications including credit approval, medical diagnosis, performance prediction, and selective marketing.

Example 7.1 Suppose that we have a database of customers on the *AllElectronics* mailing list. The mailing list is used to send out promotional literature describing new products and upcoming price discounts. The database describes attributes of the customers, such as their name, age, income, occupation, and credit rating. The customers can be classified as to whether or not they have purchased a computer at *AllElectronics*. Suppose that new customers are added to the database and that you would like to notify these customers of an uncoming computer sale. To send out promotional literature to every new customer in the database can be quite costly. A more cost efficient method would be to only target those new customers who are likely to purchase a new computer. A classification model can be constructed and used for this purpose.

Suppose instead that you would like to predict the number of major purchases that a customer will make at *AllElectronics* during a fiscal year. Since the predicted value here is ordered, a prediction model can be constructed for this purpose. □

7.2 Issues regarding classification and prediction

Preparing the data for classification and prediction. The following preprocessing steps may be applied to the data in order to help improve the accuracy, efficiency, and scalability of the classification or prediction process.

- **Data cleaning.** This refers to the preprocessing of data in order to remove or reduce *noise* (by applying smoothing techniques, for example), and the treatment of *missing values* (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics). Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.

- **Relevance analysis.** Many of the attributes in the data may be *irrelevant* to the classification or prediction task. For example, data recording the day of the week on which a bank loan application was filed is unlikely to be relevant to the success of the application. Furthermore, other attributes may be *redundant*. Hence, relevance analysis may be performed on the data with the aim of removing any irrelevant or redundant attributes from the learning process. In machine learning, this step is known as *feature selection*. Including such attributes may otherwise slow down, and possibly mislead, the learning step.

  Ideally, the time spent on relevance analysis, when added to the time spent on learning from the resulting "reduced" feature subset, should be less than the time that would have been spent on learning from the original set of features. Hence, such analysis can help improve classification efficiency and scalability.

- **Data transformation.** The data can be *generalized* to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuous-valued attributes. For example, numeric values for the attribute *income* may be generalized to discrete ranges such as *low, medium*, and *high*. Similarly, nominal-valued attributes, like *street*, can be generalized to higher-level concepts, like *city*. Since generalization compresses the original training data, fewer input/output operations may be involved during learning.

  The data may also be normalized, particularly when neural networks or methods involving distance measurements are used in the learning step. **Normalization** involves scaling all values for a given attribute so that they fall within a small specified range, such as -1.0 to 1.0, or 0 to 1.0. In methods which use distance measurements, for example, this would prevent attributes with initially large ranges (like, say *income*) from outweighing attributes with initially smaller ranges (such as binary attributes).

Data cleaning, relevance analysis, and data transformation are described in greater detail in Chapter 3 of this book.

Comparing classification methods. Classification and prediction methods can be compared and evaluated according to the following criteria:

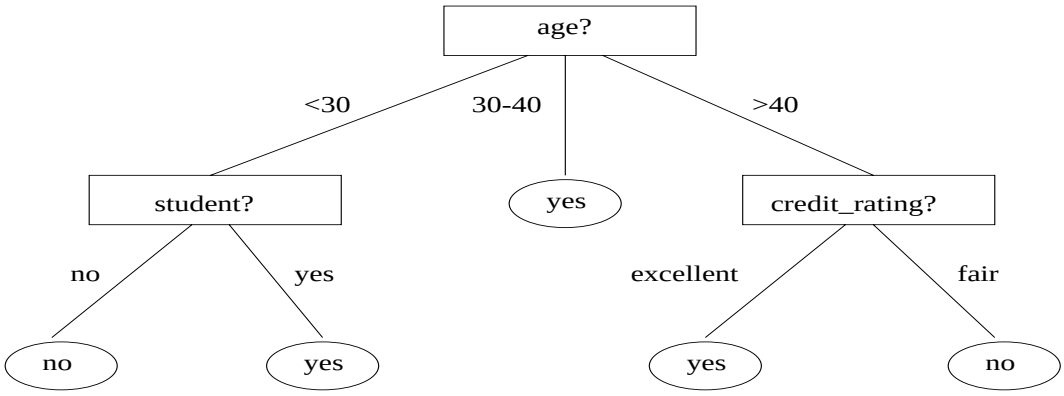


Figure 7.2: A decision tree for the concept *buys_computer*, indicating whether or not a customer at *AllElectronics* is likely to purchase a computer. Each internal (non-leaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer = yes* or *buys_computer = no*).

1. **Predictive accuracy.** This refers to the ability of the model to correctly predict the class label of new or previously unseen data.
2. **Speed.** This refers to the computation costs involved in generating and using the model.
3. **Robustness.** This is the ability of the model to make correct predictions given noisy data or data with missing values.
4. **Scalability.** This refers to the ability of the learned model to perform efficiently on large amounts of data.
5. **Interpretability.** This refers is the level of understanding and insight that is provided by the learned model.

These issues are discussed throughout the chapter. The database research community's contributions to classification and prediction for data mining have strongly emphasized the scalability aspect, particularly with respect to decision tree induction.

7.3 Classification by decision tree induction

"What is a decision tree?"

A **decision tree** is a flow-chart-like tree structure, where each *internal node* denotes a test on an attribute, each *branch* represents an outcome of the test, and *leaf nodes* represent classes or class distributions. The topmost node in a tree is the *root* node. A typical decision tree is shown in Figure 7.2. It represents the concept *buys_computer*, that is, it predicts whether or not a customer at *AllElectronics* is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.

In order to classify an unknown sample, the attribute values of the sample are tested against the decision tree. A path is traced from the root to a leaf node which holds the class prediction for that sample. Decision trees can easily be converted to classification rules.

In Section 7.3.1, we describe a basic algorithm for learning decision trees. 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. Tree pruning is described in Section 7.3.2. The extraction of classification rules from decision trees is discussed in Section 7.3.3. Enhancements of the basic decision tree algorithm are given in Section 7.3.4. Scalability issues for the induction of decision trees from large databases are discussed in Section 7.3.5. Section 7.3.6 describes the integration of decision tree induction with data warehousing facilities, such as data cubes, allowing the mining of decision trees at multiple levels of granularity. Decision trees have been used in many application areas ranging from medicine to game theory and business. Decision trees are the basis of several commercial rule induction systems.

Algorithm 7.3.1 (Generate_decision_tree) Generate a decision tree from the given training data.

Input: The training samples, *samples*, represented by discrete-valued attributes; the set of candidate attributes, *attribute-list*.

Output: A decision tree.

Method:

1) create a node N;
2) **if** *samples* are all of the same class, C **then**
3) return N as a leaf node labeled with the class C;
4) **if** *attribute-list* is empty **then**
5) return N as a leaf node labeled with the most common class in *samples*; // majority voting
6) select *test-attribute*, the attribute among *attribute-list* with the highest information gain;
7) label node N with *test-attribute*;
8) **for each** known value a_i **of** *test-attribute* // partition the samples
9) grow a branch from node N for the condition *test-attribute*=a_i;
10) let s_i be the set of samples in *samples* for which *test-attribute*=a_i; // a partition
11) **if** s_i is empty **then**
12) attach a leaf labeled with the most common class in *samples*;
13) **else** attach the node returned by Generate_decision_tree(s_i, *attribute-list* - *test-attribute*);

□

Figure 7.3: Basic algorithm for inducing a decision tree from training samples.

7.3.1 Decision tree induction

The basic algorithm for decision tree induction is a greedy algorithm which constructs decision trees in a top-down recursive divide-and-conquer manner. The algorithm, summarized in Figure 7.3, is a version of ID3, a well-known decision tree induction algorithm. Extensions to the algorithm are discussed in Sections 7.3.2 to 7.3.6.

The basic strategy is as follows:

- The tree starts as a single node representing the training samples (step 1).
- If the samples are all of the same class, then the node becomes a leaf and is labeled with that class (steps 2 and 3).
- Otherwise, the algorithm uses an entropy-based measure known as *information gain* as a heuristic for selecting the attribute that will best separate the samples into individual classes (step 6). This attribute becomes the "test" or "decision" attribute at the node (step 7). In this version of the algorithm, all attributes are categorical, i.e., discrete-valued. Continuous-valued attributes must be discretized.
- A branch is created for each known value of the test attribute, and the samples are partitioned accordingly (steps 8-10).
- The algorithm uses the same process recursively to form a decision tree for the samples at each partition. Once an attribute has occurred at a node, it need not be considered in any of the node's descendents (step 13).
- The recursive partitioning stops only when any one of the following conditions is true:
  1. All samples for a given node belong to the same class (step 2 and 3), or
  2. There are no remaining attributes on which the samples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting the given node into a leaf and labeling it with the class in majority among *samples*. Alternatively, the class distribution of the node samples may be stored; or
  3. There are no samples for the branch *test-attribute*=a_i (step 11). In this case, a leaf is created with the majority class in *samples* (step 12).

Attribute selection measure. The **information gain** measure is used to select the test attribute at each node in the tree. Such a measure is referred to as an *attribute selection measure* or a *measure of the goodness of split*. The attribute with the highest information gain (or greatest *entropy* reduction) is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an information-theoretic approach minimizes the expected number of tests needed to classify an object and guarantees that a simple (but not necessarily the simplest) tree is found.

Let S be a set consisting of s data samples. Suppose the class label attribute has m distinct values defining m distinct classes, C_i (for $i = 1, \ldots, m$). Let s_i be the number of samples of S in class C_i. The expected information needed to classify a given sample is given by:

$$I(s_1, s_2, \ldots, s_m) \quad = \quad -\sum_{i=1}^{m} p_i log_2(p_i) \tag{7.1}$$

where p_i is the probability than an arbitrary sample belongs to class C_i and is estimated by s_i/s. Note that a *log* function to the base 2 is used since the information is encoded in bits.

Let attribute A have v distinct values, $\{a_1, a_2, \cdots, a_v\}$. Attribute A can be used to partition S into v subsets, $\{S_1, S_2, \cdots, S_v\}$, where S_j contains those samples in S that have value a_j of A. If A were selected as the test attribute (i.e., best attribute for splitting), then these subsets would correspond to the branches grown from the node containing the set S. Let s_{ij} be the number of samples of class C_i in a subset S_j. The **entropy**, or expected information based on the partitioning into subsets by A is given by:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \cdots + s_{mj}}{s} I(s_{1j}, \ldots, s_{mj}). \tag{7.2}$$

The term $\sum_{j=1}^{v} \frac{s_{1j}+\cdots+s_{mj}}{s}$ acts as the weight of the j^{th} subset and is the number of samples in the subset (i.e., having value a_j of A) divided by the total number of samples in S. The smaller the entropy value is, the greater the purity of the subset partitions. The encoding information that would be gained by branching on A is

$$Gain(A) \quad = \quad I(s_1, s_2, \ldots, s_m) - E(A). \tag{7.3}$$

In other words, *Gain(A)* is the expected reduction in entropy caused by knowing the value of attribute A.

The algorithm computes the information gain of each attribute. The attribute with the highest information gain is chosen as the test attribute for the given set S. A node is created and labeled with the attribute, branches are created for each value of the attribute, and the samples are partitioned accordingly.

Example 7.2 Induction of a decision tree. Table 7.1 presents a training set of data tuples taken from the *AllElectronics* customer database. (The data are adapted from [Quinlan 1986b]). The class label attribute, *buys_computer*, has two distinct values (namely {*yes, no*}), therefore, there are two distinct classes ($m = 2$). Let C_1 correspond to the class *yes* and class C_2 correspond to *no*. There are 9 samples of class *yes* and 5 samples of class *no*. To compute the information gain of each attribute, we first use Equation (7.1) to compute the expected information needed to classify a given sample. This is:

$$I(s_1, s_2) = I(9, 5) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$

Next, we need to compute the entropy of each attribute. Let's start with the attribute *age*. We need to look at the distribution of *yes* and *no* samples for each value of *age*. We compute the expected information for each of these distributions.

for *age* = *"<30"*:	$s_{11} = 2$	$s_{21} = 3$	$I(s_{11}, s_{21}) = 0.971$
for *age* = *"30-40"*:	$s_{12} = 4$	$s_{22} = 0$	$I(s_{12}, s_{22}) = 0$
for *age* = *">40"*:	$s_{13} = 3$	$s_{23} = 2$	$I(s_{13}, s_{23}) = 0.971$

rid	age	income	student	credit_rating	Class: buys_computer
1	<30	high	no	fair	no
2	<30	high	no	excellent	no
3	30-40	high	no	fair	yes
4	>40	medium	no	fair	yes
5	>40	low	yes	fair	yes
6	>40	low	yes	excellent	no
7	30-40	low	yes	excellent	yes
8	<30	medium	no	fair	no
9	<30	low	yes	fair	yes
10	>40	medium	yes	fair	yes
11	<30	medium	yes	excellent	yes
12	30-40	medium	no	excellent	yes
13	30-40	high	yes	fair	yes
14	>40	medium	no	excellent	no

Table 7.1: Training data tuples from the *AllElectronics* customer database.

Using Equation (7.2), the expected information needed to classify a given sample if the samples are partitioned according to *age*, is:

$$E(age) = \frac{5}{14}I(s_{11}, s_{21}) + \frac{4}{14}I(s_{12}, s_{22}) + \frac{5}{14}I(s_{13}, s_{23}) = 0.694.$$

Hence, the gain in information from such a partitioning would be:

$$Gain(age) = I(s_1, s_2) - E(age) = 0.246$$

Similarly, we can compute $Gain(income) = 0.029$, $Gain(student) = 0.151$, and $Gain(credit\_rating) = 0.048$. Since *age* has the highest information gain among the attributes, it is selected as the test attribute. A node is created and labeled with *age*, and branches are grown for each of the attribute's values. The samples are then partitioned accordingly, as shown in Figure 7.4. Notice that the samples falling into the partition for *age = 30-40* all belong to the same class. Since they all belong to class *yes*, a leaf should therefore be created at the end of this branch and labeled with *yes*. The final decision tree returned by the algorithm is shown in Figure 7.2. □

In summary, decision tree induction algorithms have been used for classification in a wide range of application domains. Such systems do not use domain knowledge. The learning and classification steps of decision tree induction are generally fast. Classification accuracy is typically high for data where the mapping of classes consists of long and thin regions in concept space.

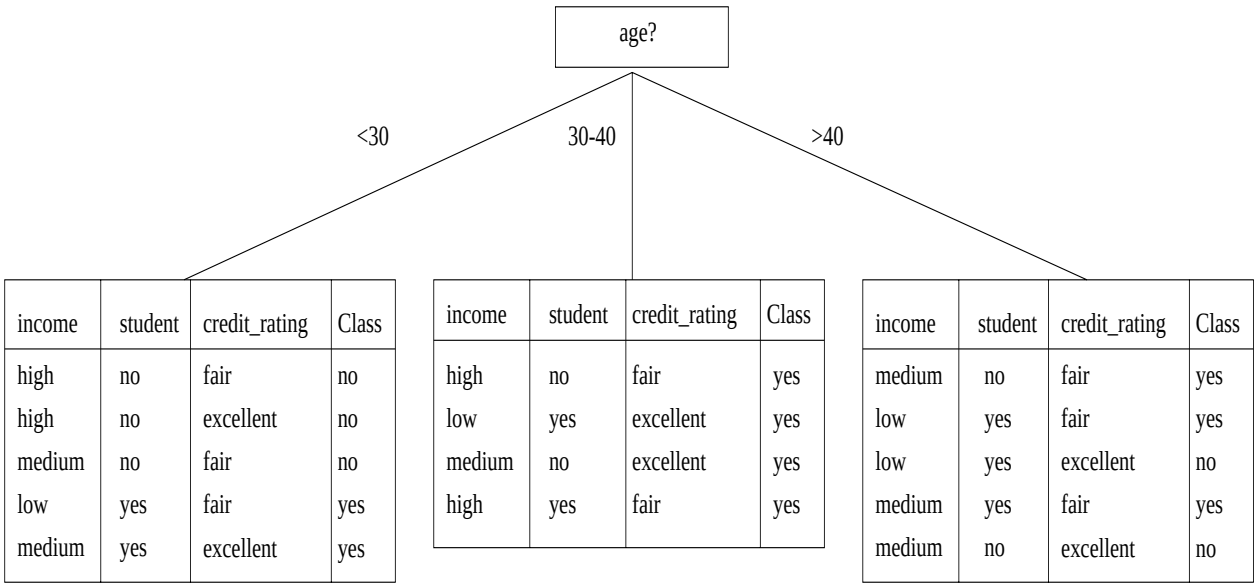
7.3.2 Tree pruning

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, generally resulting in faster classification and an improvement in the ability of the tree to correctly classify independent test data.

"How does tree pruning work?" There are two common approaches to tree pruning.

- 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 samples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset samples, or the probability distribution of those samples.

  When constructing a tree, measures such as statistical significance, χ^2, information gain, etc., can be used to assess the goodness of a split. If partitioning the samples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. There are difficulties, however,



income	student	credit_rating	Class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit_rating	Class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

income	student	credit_rating	Class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Figure 7.4: The attribute *age* has the highest information gain and therefore becomes a test attribute at the root node of the decision tree. Branches are grown for each value of *age*. The samples are shown partitioned according to each branch.

in choosing an appropriate threshold. High thresholds could result in oversimplified trees, while low thresholds could result in very little simplification.

- The **postpruning** approach removes branches from a "fully grown" tree. A tree node is pruned by removing its branches.

  The *cost complexity* pruning algorithm is an example of the postpruning approach. The pruned node becomes a leaf and is labeled by the most frequent class among its former branches. For each non-leaf node in the tree, the algorithm calculates the expected error rate that would occur if the subtree at that node were pruned. Next, the expected error rate occurring if the node were not pruned is calculated using the error rates for each branch, combined by weighting according to the proportion of observations along each branch. If pruning the node leads to a greater expected error rate, then the subtree is kept. Otherwise, it is pruned. After generating a set of progressively pruned trees, an independent test set is used to estimate the accuracy of each tree. The decision tree that minimizes the expected error rate is preferred.

  Rather than pruning trees based on expected error rates, we can prune trees based on the number of bits required to encode them. The "best pruned tree" is the one that minimizes the number of encoding bits. This method adopts the Minimum Description Length (MDL) principle which follows the notion that the simplest solution is preferred. Unlike cost complexity pruning, it does not require an independent set of samples.

Alternatively, prepruning and postpruning may be interleaved for a combined approach. Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree.

7.3.3 Extracting classification rules from decision trees

"Can I get classification rules out of my decision tree? If so, how?"

The knowledge represented in decision trees can be extracted and represented in the form of classification IF-THEN rules. One rule is created for each path from the root to a leaf node. Each attribute-value pair along a given path forms a conjunction in the rule antecedent ("IF" part). The leaf node holds the class prediction, forming the rule consequent ("THEN" part). The IF-THEN rules may be easier for humans to understand, particularly if the given tree is very large.

Example 7.3 Generating classification rules from a decision tree. The decision tree of Figure 7.2 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 7.2 are:

IF *age* = "<30" AND *student* = *no* THEN *buys_computer* = *no*
IF *age* = "<30" AND *student* = *yes* THEN *buys_computer* = *yes*
IF *age* = "30-40" THEN *buys_computer* = *yes*
IF *age* = ">40" AND *credit_rating* = *excellent* THEN *buys_computer* = *yes*
IF *age* = ">40" AND *credit_rating* = *fair* THEN *buys_computer* = *no*

□

C4.5, a later version of the ID3 algorithm, uses the training samples to estimate the accuracy of each rule. Since this would result in an optimistic estimate of rule accuracy, C4.5 employs a pessimistic estimate to compensate for the bias. Alternatively, a set of test samples independent from the training set can be used to estimate rule accuracy.

A rule can be "pruned" by removing any condition in its antecedent that does not improve the estimated accuracy of the rule. For each class, rules within a class may then be ranked according to their estimated accuracy. Since it is possible that a given test sample will not satisfy any rule antecedent, a default rule assigning the majority class is typically added to the resulting rule set.

7.3.4 Enhancements to basic decision tree induction

"What are some enhancements to basic decision tree induction?"

Many enhancements to the basic decision tree induction algorithm of Section 7.3.1 have been proposed. In this section, we discuss several major enhancements, many of which are incorporated into C4.5, a successor algorithm to ID3.

The basic decision tree induction algorithm of Section 7.3.1 requires all attributes to be categorical or discretized. The algorithm can be modified to allow for continuous-valued attributes. A test on a continuous-valued attribute A results in two branches, corresponding to the conditions $A \leq V$ and $A > V$ for some numeric value, V, of A. Given v values of A, then $v-1$ possible splits are considered in determining V. Typically, the midpoints between each pair of adjacent values are considered. If the values are sorted in advance, then this requires only one pass through the values.

The basic algorithm for decision tree induction creates one branch for each value of a test attribute, and then distributes the samples accordingly. This partitioning can result in numerous small subsets. As the subsets become smaller and smaller, the partitioning process may end up using sample sizes that are statistically insufficient. The detection of useful patterns in the subsets may become impossible due to insufficiency of the data. One alternative is to allow for the grouping of categorical attribute values. A tree node may test whether the value of an attribute belongs to a given set of values, such as $A_i \in \{a_1, a_2, \ldots, a_n\}$. Another alternative is to create binary decision trees, where each branch holds a boolean test on an attribute. Binary trees result in less fragmentation of the data. Some empirical studies have found that binary decision trees tend to be more accurate that traditional decision trees.

The information gain measure is biased in that it tends to prefer attributes with many values. Many alternatives have been proposed, such as gain ratio, which considers the probability of each attribute value. Various other selection measures exist, including the gini index, the χ^2 contingency table statistic, and the G-statistic.

Many methods have been proposed for handling missing attribute values. A missing or unknown value for an attribute A may be replaced by the most common value for A, for example. Alternatively, the apparent information gain of attribute A can be reduced by the proportion of samples with unknown values of A. In this way, "fractions" of a sample having a missing value can be partitioned into more than one branch at a test node. Other methods may look for the most probable value of A, or make use of known relationships between A and other attributes.

Incremental versions of decision tree induction have been proposed. When given new training data, these restructure the decision tree acquired from learning on previous training data, rather than relearning a new tree "from scratch".

Additional enhancements to basic decision tree induction which address scalability, and the integration of data warehousing techniques, are discussed in Sections 7.3.5 and 7.3.6, respectively.

7.3.5 Scalability and decision tree induction

"How scalable is decision tree induction?"

The efficiency of existing decision tree algorithms, such as ID3 and C4.5, has been well established for relatively small data sets. Efficiency and scalability become issues of concern when these algorithms are applied to the mining of very large, real-world databases. Most decision tree algorithms have the restriction that the training samples should reside in main memory. In data mining applications, very large training sets of millions of samples are common. Hence, this restriction limits the scalability of such algorithms, where the decision tree construction can become inefficient due to swapping of the training samples in and out of main and cache memories.

Early strategies for inducing decision trees from large databases include discretizing continuous attributes, and sampling data at each node. These, however, still assume that the training set can fit in memory. An alternative method first partitions the data into subsets which individually can fit into memory, and then builds a decision tree from each subset. The final output classifier combines each classifier obtained from the subsets. Although this method allows for the classification of large data sets, its classification accuracy is not as high as the single classifier that would have been built using all of the data at once.

rid	credit_rating	age	buys_computer
1	excellent	38	yes
2	excellent	26	yes
3	fair	35	no
4	excellent	49	no

Table 7.2: Sample data for the class *buys_computer*.

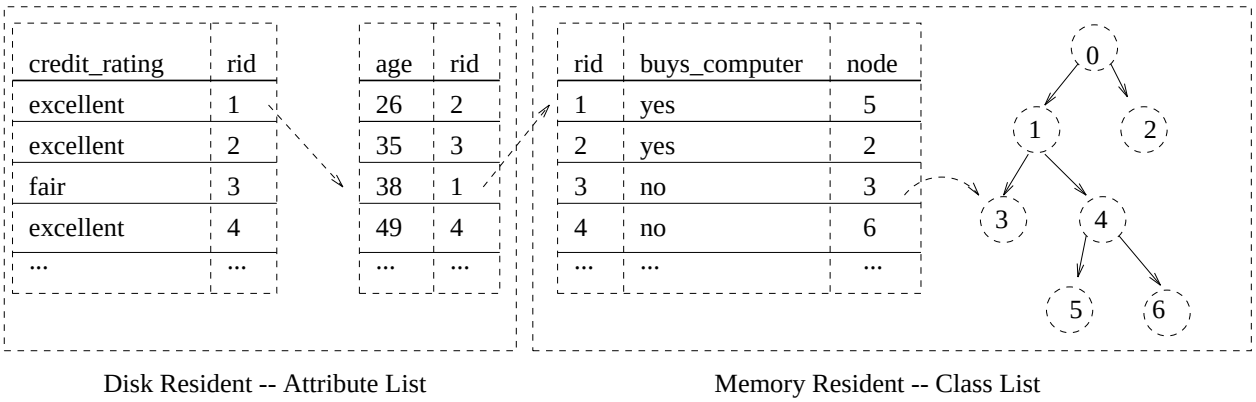


Figure 7.5: Attribute list and class list data structures used in SLIQ for the sample data of Table 7.2.

More recent decision tree algorithms which address the scalability issue have been proposed. Algorithms for the induction of decision trees from very large training sets include SLIQ and SPRINT, both of which can handle categorical and continuous-valued attributes. Both algorithms propose pre-sorting techniques on disk-resident data sets that are too large to fit in memory. Both define the use of new data structures to facilitate the tree construction. SLIQ employs disk resident *attribute lists* and a single memory resident *class list*. The attribute lists and class lists generated by SLIQ for the sample data of Table 7.2 are shown in Figure 7.5. Each attribute has an associated attribute list, indexed by *rid* (a record identifier). Each tuple is represented by a linkage of one entry from each attribute list to an entry in the class list (holding the class label of the given tuple), which in turn is linked to its corresponding leaf node in the decision tree. The class list remains in memory since it is often accessed and modified in the building and pruning phases. The size of the class list grows proportionally with the number of tuples in the training set. When a class list cannot fit into memory, the performance of SLIQ decreases.

SPRINT uses a different *attribute list* data structure which holds the class and *rid* information, as shown in Figure 7.6. When a node is split, the attribute lists are partitioned and distributed among the resulting child nodes

credit_rating	buys_computer	rid
excellent	yes	1
excellent	yes	2
fair	no	3
excellent	no	4
...	...	...

age	buys_computer	rid
26	y	2
35	n	3
38	y	1
49	n	4
...	...	...

Figure 7.6: Attribute list data structure used in SPRINT for the sample data of Table 7.2.

accordingly. When a list is partitioned, the order of the records in the list is maintained. Hence, partitioning lists does not require resorting. SPRINT was designed to be easily parallelized, further contributing to its scalability.

While both SLIQ and SPRINT handle disk-resident data sets that are too large to fit into memory, the scalability of SLIQ is limited by the use of its memory-resident data structure. SPRINT removes all memory restrictions, yet requires the use of a hash tree proportional in size to the training set. This may become expensive as the training set size grows.

RainForest is a framework for the scalable induction of decision trees. The method adapts to the amount of main memory available, and apply to any decision tree induction algorithm. It maintains an AVC-set (Attribute-Value, Class label) indicating the class distribution for each attribute. RainForest reports a speed-up over SPRINT.

7.3.6 Integrating data warehousing techniques and decision tree induction

Decision tree induction can be integrated with data warehousing techniques for data mining. In this section we discuss the method of attribute-oriented induction to generalize the given data, and the use of multidimensional data cubes to store the generalized data at multiple levels of granularity. We then discuss how these approaches can be integrated with decision tree induction in order to facilitate interactive multilevel mining. The use of a data mining query language to specify classification tasks is also discussed. In general, the techniques described here are applicable to other forms of learning as well.

Attribute-oriented induction (AOI) uses concept hierarchies to generalize the training data by replacing lower level data with higher level concepts (Chapter 5). For example, numerical values for the attribute *income* may be generalized to the ranges *"<30K", "30K-40K", ">40K"*, or the categories *low, medium*, or *high*. This allows the user to view the data at more meaningful levels. In addition, the generalized data are more compact than the original training set, which may result in fewer input/output operations. Hence, AOI also addresses the scalability issue by compressing the training data.

The generalized training data can be stored in a **multidimensional data cube**, such as the structure typically used in data warehousing (Chapter 2). The data cube is a multidimensional data structure, where each dimension represents an attribute or a set of attributes in the data schema, and each cell stores the value of some aggregate measure (such as *count*). Figure 7.7 shows a data cube for customer information data, with the dimensions *income*, *age*, and *occupation*. The original numeric values of *income and age* have been generalized to ranges. Similarly, original values for *occupation*, such as *accountant* and *banker*, or *nurse* and *X-ray technician*, have been generalized to *finance* and *medical*, respectively. The advantage of the multidimensional structure is that it allows fast indexing to cells (or *slices*) of the cube. For instance, one may easily and quickly access the total count of customers in occupations relating to finance who have an income greater than $40K, or the number of customers who work in the area of medicine and are less than 40 years old.

Data warehousing systems provide a number of operations that allow mining on the data cube at multiple levels of granularity. To review, the **roll-up** operation performs aggregation on the cube, either by climbing up a concept hierarchy (e.g., replacing the value *banker* for *occupation* by the more general, *finance*), or by removing a dimension in the cube. **Drill-down** performs the reverse of roll-up, by either stepping down a concept hierarchy or adding a dimension (e.g., time). A **slice** performs a selection on one dimension of the cube. For example, we may obtain a data slice for the generalized value *accountant* of *occupation*, showing the corresponding *income* and *age* data. A **dice** performs a selection on two or more dimensions. The **pivot** or *rotate* operation rotates the data axes in view

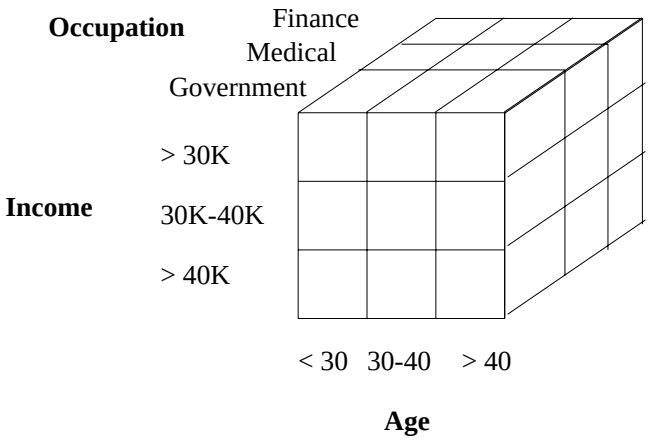


Figure 7.7: A multidimensional data cube.

in order to provide an alternative presentation of the data. For example, pivot may be used to transform a 3-D cube into a series of 2-D planes.

The above approaches can be integrated with decision tree induction to provide interactive multilevel mining of decision trees. The data cube and knowledge stored in the concept hierarchies can be used to induce decision trees at different levels of abstraction. Furthermore, once a decision tree has been derived, the concept hierarchies can be used to generalize or specialize individual nodes in the tree, allowing attribute roll-up or drill-down, and reclassification of the data for the newly specified abstraction level. This interactive feature will allow users to focus their attention on areas of the tree or data which they find interesting.

When integrating AOI with decision tree induction, generalization to a very low (specific) concept level can result in quite large and bushy trees. Generalization to a very high concept level can result in decision trees of little use, where interesting and important subconcepts are lost due to overgeneralization. Instead, generalization should be to some intermediate concept level, set by a domain expert or controlled by a user-specified threshold. Hence, the use of AOI may result in classification trees that are more understandable, smaller, and therefore easier to interpret than trees obtained from methods operating on ungeneralized (larger) sets of low-level data (such as SLIQ or SPRINT).

A criticism of typical decision tree generation is that, because of the recursive partitioning, some resulting data subsets may become so small that partitioning them further would have no statistically significant basis. The maximum size of such "insignificant" data subsets can be statistically determined. To deal with this problem, an **exception threshold** may be introduced. If the portion of samples in a given subset is less than the threshold, further partitioning of the subset is halted. Instead, a leaf node is created which stores the subset and class distribution of the subset samples.

Owing to the large amount and wide diversity of data in large databases, it may not be reasonable to assume that each leaf node will contain samples belonging to a common class. This problem may be addressed by employing a **precision** or **classification threshold**. Further partitioning of the data subset at a given node is terminated if the percentage of samples belonging to any given class at that node exceeds this threshold.

A data mining query language may be used to specify and facilitate the enhanced decision tree induction method. Suppose that the data mining task is to predict the credit risk of customers aged 30-40, based on their income and occupation. This may be specified as the following data mining query:

```
mine classification
analyze credit_risk
in relevance to income, occupation
from Customer_db
where (age >= 30) and (age < 40)
display as rules
```

The above query, expressed in DMQL[1], executes a relational query on *Customer_db* to retrieve the task-relevant data. Tuples not satisfying the where clause are ignored, and only the data concerning the attributes specified in the in relevance to clause, and the class label attribute (*credit_risk*) are collected. AOI is then performed on this data. Since the query has not specified which concept hierarchies to employ, default hierarchies are used. A graphical user interface may be designed to facilitate user specification of data mining tasks via such a data mining query language. In this way, the user can help guide the automated data mining process.

Hence, many ideas from data warehousing can be integrated with classification algorithms, such as decision tree induction, in order to facilitate data mining. Attribute-oriented induction employs concept hierarchies to generalize data to multiple abstraction levels, and can be integrated with classification methods in order to perform multilevel mining. Data can be stored in multidimensional data cubes to allow quick accessing to aggregate data values. Finally, a data mining query language can be used to assist users in interactive data mining.

7.4 Bayesian classification

"What are Bayesian classifiers"?

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given sample belongs to a particular class.

Bayesian classification is based on Bayes theorem, described below. Studies comparing classification algorithms have found a simple Bayesian classifier known as the *naive Bayesian classifier* to be comparable in performance with decision tree and neural network classifiers. 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". *Bayesian belief networks* are graphical models, which unlike naive Bayesian classifiers, allow the representation of dependencies among subsets of attributes. Bayesian belief networks can also be used for classification.

Section 7.4.1 reviews basic probability notation and Bayes theorem. You will then learn naive Bayesian classification in Section 7.4.2. Bayesian belief networks are described in Section 7.4.3.

7.4.1 Bayes theorem

Let X be a data sample whose class label is unknown. Let H be some hypothesis, such as that the data sample 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 observed data sample X.

$P(H|X)$ is the **posterior probability**, or *a posteriori probability*, of H conditioned on X. For example, suppose the world of data samples consists of fruits, described by their color and shape. Suppose that X is red and round, and that H is the hypothesis that X is an apple. Then $P(H|X)$ reflects our confidence that X is an apple given that we have seen that X is red and round. In contrast, $P(H)$ is the **prior probability**, or *a priori probability* of H. For our example, this is the probability that any given data sample is an apple, regardless of how the data sample looks. The posterior probability, $P(H|X)$ is based on more information (such as background knowledge) 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 X is red and round given that we know that it is true that X is an apple. $P(X)$ is the prior probability of X. Using our example, it is the probability that a data sample from our set of fruits is red and round.

"How are these probabilities estimated?" $P(X)$, $P(H)$, and $P(X|H)$ 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 $P(H)$, $P(X)$, and $P(X|H)$. Bayes theorem is:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{7.4}$$

In the next section, you will learn how Bayes theorem is used in the naive Bayesian classifier.

[1] The use of a data mining query language to specify data mining queries is discussed in Chapter 4, using the SQL-based DMQL language.

7.4.2 Naive Bayesian classification

The **naive Bayesian** classifier, or **simple Bayesian** classifier, works as follows:

1. Each data sample is represented by an n-dimensional feature vector, $X = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the sample from n attributes, respectively $A_1, A_2, .., A_n$.

2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given an unknown data sample, X (i.e., having no class label), the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naive Bayesian classifier assigns an unknown sample X to the class C_i if and only if :

$$P(C_i|X) > P(C_j|X) \text{ for } 1 \leq j \leq m,\ j \neq i.$$

   Thus we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posteriori hypothesis*. By Bayes theorem (Equation (7.4)),

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}. \tag{7.5}$$

3. As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, i.e. $P(C_1) = P(C_2) = \ldots = P(C_m)$, and we would therefore maximize $P(X|C_i)$. Otherwise, we maximize $P(X|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = \frac{s_i}{s}$, where s_i is the number of training samples of class C_i, and s is the total number of training samples.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$. In order to reduce computation in evaluating $P(X|C_i)$, the naive assumption of **class conditional independence** is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the sample, i.e., that there are no dependence relationships among the attributes. Thus,

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i). \tag{7.6}$$

   The probabilities $P(x_1|C_i), P(x_2|C_i), \ldots, P(x_n|C_i)$ can be estimated from the training samples, where:

   (a) If A_k is categorical, then $P(x_k|C_i) = \frac{s_{ik}}{s_i}$, where s_{ik} is the number of training samples of class C_i having the value x_k for A_k, and s_i is the number of training samples belonging to C_i.

   (b) If A_k is continuous-valued, then the attribute is assumed to have a Gaussian distribution. Therefore,

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) = \frac{1}{\sqrt{2\pi}\sigma_{C_i}} e^{-\frac{(x-\mu_{C_i})^2}{2\sigma_{C_i}^2}}, \tag{7.7}$$

   where $g(x_k, \mu_{C_i}, \sigma_{C_i})$ is the **Gaussian (normal) density function** for attribute A_k, while μ_{C_i} and σ_{C_i} are the mean and variance respectively given the values for attribute A_k for training samples of class C_i.

5. In order to classify an unknown sample X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. Sample X is then assigned to the class C_i if and only if :

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \text{ for } 1 \leq j \leq m,\ j \neq i.$$

   In other words, it is assigned to the class, C_i, for which $P(X|C_i)P(C_i)$ is the maximum.

"How effective are Bayesian classifiers?"

In theory, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. However, in practice this is not always the case owing to inaccuracies in the assumptions made for its use, such as class conditional independence, and the lack of available probability data. However, various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains.

Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers which do not explicitly use Bayes theorem. For example, under certain assumptions, it can be shown that many neural network and curve fitting algorithms output the *maximum posteriori* hypothesis, as does the naive Bayesian classifier.

Example 7.4 Predicting a class label using naive Bayesian classification. We wish to predict the class label of an unknown sample using naive Bayesian classification, given the same training data as in Example 7.2 for decision tree induction. The training data are in Table 7.1. The data samples are described by the attributes *age*, *income*, *student*, and *credit_rating*. The class label attribute, *buys_computer*, has two distinct values (namely {*yes*, *no*}). Let C_1 correspond to the class *buys_computer = yes* and C_2 correspond to *buys_computer = no*. The unknown sample we wish to classify is

$$X = (age = \text{"<30"},\ income = medium,\ student = yes,\ credit\_rating = fair).$$

We need to maximize $P(X|C_i)P(C_i)$, for $i = 1, 2$. $P(C_i)$, the prior probability of each class, can be computed based on the training samples:

$$P(buys\_computer = yes) = 9/14 = 0.643$$
$$P(buys\_computer = no) = 5/14 = 0.357$$

To compute $P(X|C_i)$, for $i = 1, 2$, we compute the following conditional probabilities:

$$\begin{aligned}
&P(age = \text{"<30"} \mid buys_computer = yes) &&= 2/9 = 0.222\\
&P(age = \text{"<30"} \mid buys_computer = no) &&= 3/5 = 0.600\\
&P(income = medium \mid buys_computer = yes) &&= 4/9 = 0.444\\
&P(income = medium \mid buys_computer = no) &&= 2/5 = 0.400\\
&P(student = yes \mid buys_computer = yes) &&= 6/9 = 0.667\\
&P(student = yes \mid buys_computer = no) &&= 1/5 = 0.200\\
&P(credit_rating = fair \mid buys_computer = yes) &&= 6/9 = 0.667\\
&P(credit_rating = fair \mid buys_computer = no) &&= 2/5 = 0.400
\end{aligned}$$

Using the above probabilities, we obtain

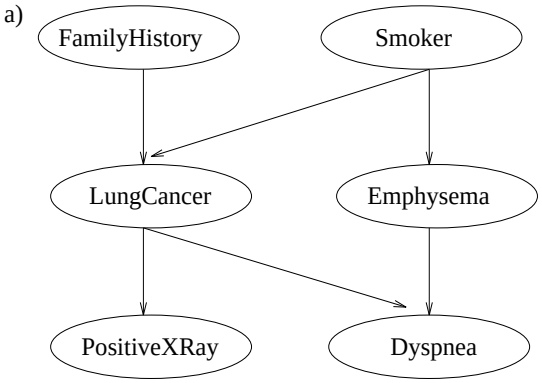
$$P(X|buys\_computer = yes) = 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044$$
$$P(X|buys\_computer = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019$$
$$P(X|buys\_computer = yes)P(buys\_computer = yes) = 0.044 \times 0.643 = 0.028$$
$$P(X|buys\_computer = no)P(buys\_computer = no) = 0.019 \times 0.357 = 0.007$$

Therefore, the naive Bayesian classifier predicts *"buys_computer = yes"* for sample X. □

7.4.3 Bayesian belief networks

The naive Bayesian classifier makes the assumption of class conditional independence, i.e., that given the class label of a sample, the values of the attributes are conditionally independent of one another. This assumption simplifies computation. When the assumption holds true, then the naive Bayesian classifier is the most accurate in comparison with all other classifiers. In practice, however, dependencies can exist between variables. **Bayesian belief networks** specify joint conditional probability distributions. They allow class conditional independencies to be defined between subsets of variables. They provide a graphical model of causal relationships, on which learning can be performed. These networks are also known as **belief networks**, **Bayesian networks**, and **probabilistic networks**. For brevity, we will refer to them as belief networks.

a)



b)

	FH, S	FH, ~S	~FH, S	~FH, ~S
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

Figure 7.8: a) A simple Bayesian belief network; b) The conditional probability table for the values of the variable *LungCancer (LC)* showing each possible combination of the values of its parent nodes, *Family History (FH)* and *Smoker (S)*.

A belief network is defined by two components. The first is a *directed acyclic graph*, where each node represents a random variable, and each arc represents a probabilistic dependence. If an arc is drawn from a node Y to a node Z, then Y is a **parent** or **immediate predecessor** of Z, and Z is a **descendent** of Y. Each variable is conditionally independent of its nondescendents in the graph, given its parents. The variables may be discrete or continuous-valued. They may correspond to actual attributes given in the data, or to "hidden variables" believed to form a relationship (such as medical syndromes in the case of medical data).

Figure 7.8a) shows a simple belief network, adapted from [Russell et al. 1995a] for six Boolean variables. The arcs allow a representation of causal knowledge. For example, having lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. Furthermore, the arcs also show that the variable *LungCancer* is conditionally independent of *Emphysema*, given its parents, *FamilyHistory* and *Smoker*. This means that once the values of *FamilyHistory* and *Smoker* are known, then the variable *Emphysema* does not provide any additional information regarding *LungCancer*.

The second component defining a belief network consists of one *conditional probability table (CPT)* for each variable. The CPT for a variable Z specifies the conditional distribution $P(Z|Parents(Z))$, where $Parents(Z)$ are the parents of Z. Figure 7.8b) showns a CPT for *LungCancer*. The conditional probability for each value of *LungCancer* is given for each possible combination of values of its parents. For instance, from the upper leftmost and bottom rightmost entries, respectively, we see that

$$P(LungCancer = Yes \mid FamilyHistory = Yes,\ Smoker = Yes) = 0.8, \text{ and}$$
$$P(LungCancer = No \mid FamilyHistory = No,\ Smoker = No) = 0.9.$$

The joint probability of any tuple $(z_1, ..., z_n)$ corresponding to the variables or attributes $Z_1, ..., Z_n$ is computed by

$$P(z_1, ..., z_n) = \prod_{i=1}^{n} P(z_i|Parents(Z_i)), \tag{7.8}$$

where the values for $P(z_i|Parents(Z_i))$ correspond to the entries in the CPT for Z_i.

A node within the network can be selected as an "output" node, representing a class label attribute. There may be more than one output node. Inference algorithms for learning can be applied on the network. The classification process, rather than returning a single class label, can return a probability distribution for the class label attribute, i.e., predicting the probability of each class.

7.4.4 Training Bayesian belief networks

"How does a Bayesian belief network learn?"

In learning or training a belief network, a number of scenarios are possible. The network structure may be given in advance, or inferred from the data. The network variables may be *observable* or *hidden* in all or some of the training samples. The case of hidden data is also referred to as *missing values* or *incomplete data*.

If the network structure is known and the variables are observable, then learning the network is straightforward. It consists of computing the CPT entries, as is similarly done when computing the probabilities involved in naive Bayesian classification.

When the network structure is given and some of the variables are hidden, then a method of gradient descent can be used to train the belief network. The object is to learn the values for the CPT entries. Let S be a set of s training samples, $X_1, X_2, .., X_s$. Let w_{ijk} be a CPT entry for the variable $Y_i = y_{ij}$ having the parents $U_i = u_{ik}$. For example, if w_{ijk} is the upper leftmost CPT entry of Figure 7.8b), then Y_i is *LungCancer*; y_{ij} is its value, *Yes*; U_i lists the parent nodes of Y_i, namely $\{$*FamilyHistory, Smoker*$\}$; and u_{ik} lists the values of the parent nodes, namely $\{$*Yes, Yes*$\}$. The w_{ijk} are viewed as weights, analogous to the weights in hidden units of neural networks (Section 7.5). The weights, w_{ijk}, are initialized to random probability values. The gradient descent strategy performs greedy hill-climbing. At each iteration, the weights are updated, and will eventually converge to a local optimum solution.

The method aims to maximize $P(S|H)$. This is done by following the gradient of $lnP(S|H)$, which makes the problem simpler. Given the network structure and initialized w_{ijk}, the algorithm proceeds as follows.

1. **Compute the gradients:** For each i, j, k, compute

$$\frac{\partial lnP(S|H)}{\partial w_{ijk}} = \sum_{d=1}^{s} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|X_d)}{w_{ijk}} \tag{7.9}$$

   The probability in the right-hand side of Equation (7.9) is to be calculated for each training sample X_d in S. For brevity, let's refer to this probability simply as p. When the variables represented by Y_i and U_i are hidden for some X_d, then the corresponding probability p can be computed from the observed variables of the sample using standard algorithms for Bayesian network inference (such as those available by the commercial software package, Hugin).

2. **Take a small step in the direction of the gradient:** The weights are updated by

$$w_{ijk} \leftarrow w_{ijk} + (l)\frac{\partial lnP(S|H)}{\partial w_{ijk}}, \tag{7.10}$$

   where l is the **learning rate** representing the step size, and $\frac{\partial lnP(S|H)}{\partial w_{ijk}}$ is computed from Equation (7.9). The learning rate is set to a small constant.

3. **Renormlize the weights:** Because the weights w_{ijk} are probability values, they must be between 0 and 1.0, and $\sum_j w_{ijk}$ must equal 1 for all i, k. These criteria are achieved by renormalizing the weights after they have been updated by Equation (7.10).

Several algorithms exist for learning the network structure from the training data given observable variables. The problem is one of discrete optimization. For solutions, please see the bibliographic notes at the end of this chapter.

7.5 Classification by backpropagation

"What is backpropagation?"

Backpropagation is a neural network learning algorithm. The field of neural networks was originally kindled by psychologists and neurobiologists who sought to develop and test computational analogues of neurons. Roughly

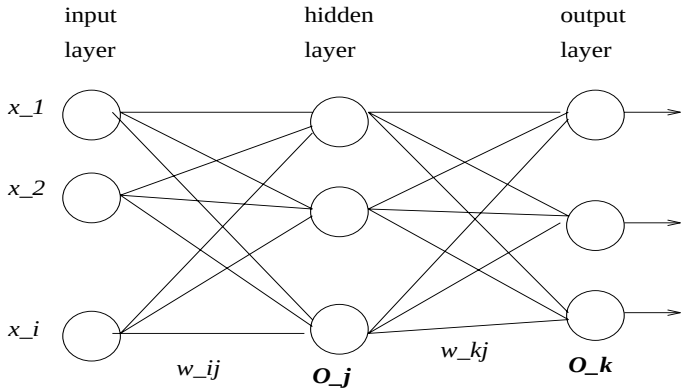


Figure 7.9: A multilayer feed-forward neural network: A training sample, $X = (x_1, x_2, .., x_i)$, is fed to the input layer. Weighted connections exist between each layer, where w_{ij} denotes the weight from a unit j in one layer to a unit i in the previous layer.

speaking, a **neural network** is a set of connected input/output units where 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 samples. Neural network learning is also referred to as *connectionist learning* due to the connections between units.

Neural networks involve long training times, and are therefore more suitable for applications where this is feasible. They require a number of parameters which are typically best determined empirically, such as the network topology or "structure". Neural networks have been criticized for their poor interpretability, since it is difficult for humans to interpret the symbolic meaning behind the learned weights. These features initially made neural networks less desirable for data mining.

Advantages of neural networks, however, include their high tolerance to noisy data as well as their ability to classify patterns on which they have not been trained. In addition, several algorithms have recently been developed for the extraction of rules from trained neural networks. These factors contribute towards the usefulness of neural networks for classification in data mining.

The most popular neural network algorithm is the backpropagation algorithm, proposed in the 1980's. In Section 7.5.1 you will learn about multilayer feed-forward networks, the type of neural network on which the backpropagation algorithm performs. Section 7.5.2 discusses defining a network topology. The backpropagation algorithm is described in Section 7.5.3. Rule extraction from trained neural networks is discussed in Section 7.5.4.

7.5.1 A multilayer feed-forward neural network

The backpropagation algorithm performs learning on a **multilayer feed-forward** neural network. An example of such a network is shown in Figure 7.9. The inputs correspond to the attributes measured for each training sample. The inputs are fed simultaneously into a layer of units making up the **input layer**. The weighted outputs of these units are, in turn, fed simultaneously to a second layer of "neuron-like" units, known as a **hidden layer**. The hidden layer's weighted outputs 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 samples.

The units in the hidden layers and output layer are sometimes referred to as **neurodes**, due to their symbolic biological basis, or as **output units**. The multilayer neural network shown in Figure 7.9 has two layers of output units. Therefore, we say that it is a **two-layer** neural network. Similarly, a network containing two hidden layers is called a *three-layer* neural network, and so on. The network is **feed-forward** in that none of the weights cycle back to an input unit or to an output unit of a previous layer. It is **fully connected** in that each unit provides input to each unit in the next forward layer.

Multilayer feed-forward networks of linear threshold functions, given enough hidden units, can closely approximate any function.

7.5.2 Defining a network topology

"How can I design the topology of the neural network?"

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 samples will help speed up the learning phase. Typically, input values are normalized so as to fall between 0 and 1.0. Discrete-valued attributes may be encoded such that there is one input unit per domain value. For example, if the domain of an attribute A is $\{a_0, a_1, a_2\}$, 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. If $A = a_1$, I_1 is set to 1, and so on. 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 1 output unit per class is used.

There are no clear rules as to the "best" number of hidden layer units. Network design is a trial by 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, then it is common to repeat the training process with a different network topology or a different set of initial weights.

7.5.3 Backpropagation

"How does backpropagation work?"

Backpropagation learns by iteratively processing a set of training samples, comparing the network's prediction for each sample with the actual known class label. For each training sample, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual class. These modifications are made in the "backwards" direction, i.e., from the output layer, through each hidden layer down to the first hidden layer (hence the name *backpropagation*). Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops. The algorithm is summarized in Figure 7.10. Each step is described below.

Initialize the weights. The weights in the network are initialized to small random numbers (e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5). Each unit has a *bias* associated with it, as explained below. The biases are similarly initialized to small random numbers.

Each training sample, X, is processed by the following steps.

Propagate the inputs forward. In this step, the net input and output of each unit in the hidden and output layers are computed. First, the training sample is fed to the input layer of the network. The net input to each unit in the hidden and output layers is then computed as a linear combination of its inputs. To help illustrate this, a hidden layer or output layer unit is shown in Figure 7.11. The inputs to the unit are, in fact, the outputs of the units connected to it in the previous layer. To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed. Given a unit j in a hidden or output layer, the net input, I_j, to unit j is:

$$I_j = \sum_i w_{ij} O_i + \theta_j \tag{7.11}$$

where w_{ij} is the weight of the connection from unit i in the previous layer to unit j; O_i is the output of unit i from the previous layer; and θ_j is the **bias** of the unit. The bias acts as a threshold in that it serves to vary the activity of the unit.

Each unit in the hidden and output layers takes its net input, and then applies an **activation** function to it, as illustrated in Figure 7.11. The function symbolizes the activation of the neuron represented by the unit. The **logistic**, or **simoid** function is used. Given the net input I_j to unit j, then O_j, the output of unit j, is computed as:

$$O_j = \frac{1}{1 + e^{-I_j}} \tag{7.12}$$

This function is also referred to as a *squashing function*, since it maps a large input domain onto the smaller range

Algorithm 7.5.1 (Backpropagation) Neural network learning for classification, using the backpropagation algorithm.

Input: The training samples, *samples*; the learning rate, *l*; a multilayer feed-forward network, *network*.

Output: A neural network trained to classify the samples.

Method:

1) Initialize all weights and biases in *network*;
2) **while** terminating condition is not satisfied {
3) **for** each training sample X in *samples* {
4) // Propagate the inputs forward:
5) **for** each hidden or output layer unit j
6) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j
7) **for** each hidden or output layer unit j
8) $O_j = \frac{1}{1+e^{-I_j}}$; // compute the output of each unit j
9) // Backpropagate the errors:
10) **for** each unit j in the output layer
11) $Err_j = O_j(1-O_j)(T_j - O_j)$; // compute the error
12) **for** each unit j in the hidden layers
13) $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$; // compute the error
14) **for** each weight w_{ij} in *network* {
15) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
16) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
17) **for** each bias θ_j in *network* {
18) $\Delta \theta_j = (l) Err_j$; // bias increment
19) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
20) } }

□

Figure 7.10: Backpropagation algorithm.

of 0 to 1. The logistic function is nonlinear and differentiable, allowing the backpropagation algorithm to model classification problems that are linearly inseparable.

Backpropagate the error. The error is propagated backwards by updating the weights and biases to reflect the error of the network's prediction. For a unit j in the output layer, the error Err_j is computed by:

$$Err_j = O_j(1-O_j)(T_j - O_j) \tag{7.13}$$

where O_j is the actual output of unit j, and T_j is the *true* output, based on the known class label of the given training sample. Note that $O_j(1-O_j)$ is the derivative of the logistic function.

To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered. The error of a hidden layer unit j is:

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk} \tag{7.14}$$

where w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and Err_k is the error of unit k.

The weights and biases are updated to reflect the propagated errors. Weights are updated by Equations (7.15) and (7.16) below, where Δw_{ij} is the change in weight w_{ij}.

$$\Delta w_{ij} = (l) Err_j O_i \tag{7.15}$$

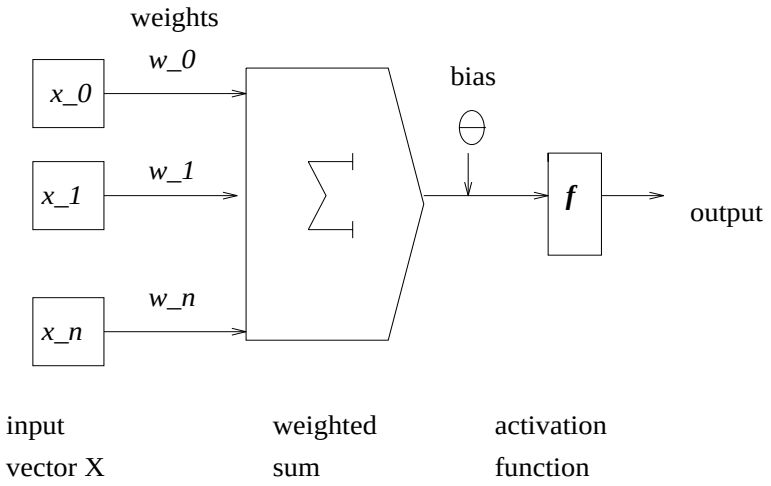


Figure 7.11: A hidden or output layer unit: The inputs are multiplied by their corresponding weights in order to form a weighted sum, which is added to the bias associated with the unit. A nonlinear activation function is applied to the net input.

$$w_{ij} = w_{ij} + \Delta w_{ij} \tag{7.16}$$

"What is the 'l' in Equation 7.15?" The variable l is the **learning rate**, a constant typically having a value between 0 and 1.0. Backpropagation learns using a method of gradient descent to search for a set of weights which can model the given classification problem so as to minimize the mean squared distance between the network's class predictions and the actual class label of the samples. The learning rate helps to avoid getting stuck at a local minimum in decision space (i.e., where the weights appear to converge, but are not the optimum solution), and encourages finding the global minimum. If the learning rate is too small, then learning will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur. A rule of thumb is to set the learning rate to $1/t$, where t is the number of iterations through the training set so far.

Biases are updated by Equations (7.17) and (7.18) below, where $\Delta\theta_j$ is the change in bias θ_j.

$$\Delta\theta_j = (l)Err_j \tag{7.17}$$

$$\theta_j = \theta_j + \Delta\theta_j \tag{7.18}$$

Note that here we are updating the weights and biases after the presentation of each sample. This is referred to as **case updating**. Alternatively, the weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all of the samples in the training set have been presented. This latter strategy is called **epoch updating**, where one iteration through the training set is an **epoch**. In theory, the mathematical derivation of backpropagation employs epoch updating, yet in practice, case updating is more common since it tends to yield more accurate results.

Terminating condition. Training stops when either

1. all Δw_{ij} in the previous epoch were so small as to be below some specified threshold, or
2. the percentage of samples misclassified in the previous epoch is below some threshold, or
3. a prespecified number of epochs has expired.

In practice, several hundreds of thousands of epochs may be required before the weights will converge.

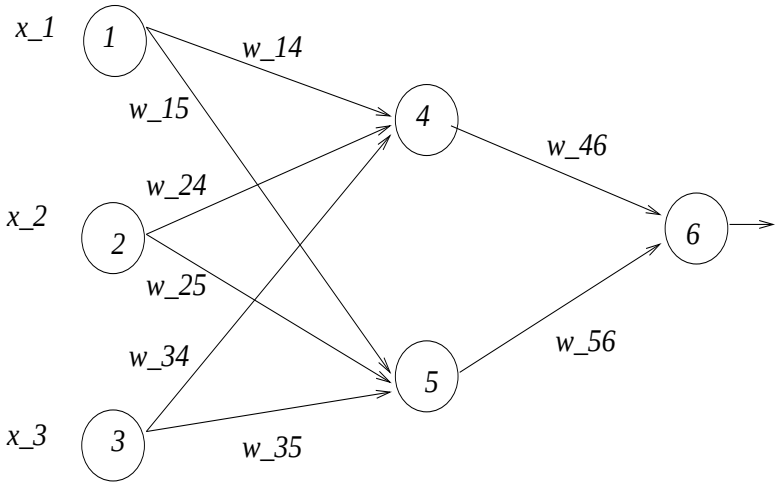


Figure 7.12: An example of a multilayer feed-forward neural network.

Example 7.5 Sample calculations for learning by the backpropagation algorithm. Figure 7.12 shows a multilayer feed-forward neural network. The initial weight and bias values of the network are given in Table 7.3, along with the first training sample, $X = (1, 0, 1)$.

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 7.3: Initial input, weight, and bias values.

This example shows the calculations for backpropagation, given the first training sample, X. The sample is fed into the network, and the net input and output of each unit are computed. These values are shown in Table 7.4.

Unit j	Net Input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.33$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.52$
6	$(0.3)(0.33) - (0.2)(0.52) + 0.1 - 0.19$	$1/(1 + e^{-0.19}) = 0.55$

Table 7.4: The net input and output calculations.

The error of each unit is computed and propagated backwards. The error values are shown in Table 7.5. The weight and bias updates are shown in Table 7.6.

□

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.

7.5.4 Backpropagation and interpretability

"How can I 'understand' what the backpropgation network has learned?"

A major disadvantage of neural networks lies in their knowledge representation. Acquired knowledge in the form of a network of units connected by weighted links is difficult for humans to interpret. This factor has motivated research

Unit j	Err_j
6	$(0.55)(1-0.55)(1-0.55) = 0.495$
5	$(0.52)(1-0.52)(0.495)(-0.3) = 0.037$
4	$(0.33)(1-0.33)(0.495)(-0.2) = -0.022$

Table 7.5: Calculation of the error at each node.

Weight or Bias	New Value
w_{46}	$-0.3 = (0.9)(0.495)(0.33) = -0.153$
w_{56}	$-0.2 = (0.9)(0.495)(0.52) = -0.032$
w_{14}	$0.2 = (0.9)(-0.022)(1) = 0.180$
w_{15}	$-0.3 = (0.9)(0.037)(1) = -0.267$
w_{24}	$0.4 = (0.9)(-0.022)(0) = 0.4$
w_{25}	$0.1 = (0.9)(0.037)(0) = 0.1$
w_{34}	$-0.5 = (0.9)(-0.022)(1) = -0.520$
w_{35}	$0.2 = (0.9)(0.037)(1) = 0.233$
θ_6	$0.1 + (0.9)(0.495) = 0.546$
θ_5	$0.2 + (0.9)(0.037) = 0.233$
θ_4	$-0.4 + (0.9)(-0.022) = -0.420$

Table 7.6: Calculations for weight and bias updating.

in extracting the knowledge embedded in trained neural networks and in representing that knowledge symbolically. Methods include extracting rules from networks and sensitivity analysis.

Various algorithms for the extraction of rules have been proposed. The methods typically impose restrictions regarding procedures used in training the given neural network, the network topology, and the discretization of input values.

Fully connected networks are difficult to articulate. Hence, often, the first step towards extracting rules from neural networks is **network pruning**. This consists of removing weighted links that do not result in a decrease in the classification accuracy of the given network.

Once the trained network has been pruned, some approaches will then perform link, unit, or activation value clustering. In one method, for example, clustering is used to find the set of common activation values for each hidden unit in a given trained two-layer neural network (Figure 7.13). The combinations of these activation values for each hidden unit are analyzed. Rules are derived relating combinations of activation values with corresponding output unit values. Similarly, the sets of input values and activation values are studied to derive rules describing the relationship between the input and hidden unit layers. Finally, the two sets of rules may be combined to form IF-THEN rules. Other algorithms may derive rules of other forms, including M-of-N rules (where M out of a given N conditions in the rule antecedent must be true in order for the rule consequent to be applied), decision trees with M-of-N tests, fuzzy rules, and finite automata.

Sensitivity analysis is used to assess the impact that a given input variable has on a network output. The input to the variable is varied while the remaining input variables are fixed at some value. Meanwhile, changes in the network output are monitored. The knowledge gained from this form of analysis can be represented in rules such as "*IF X decreases* 5% *THEN Y increases* 8%".

7.6 Association-based classification

"Can association rule mining be used for classification?"

Association rule mining is an important and highly active area of data mining research. Chapter 6 of this book described many algorithms for association rule mining. Recently, data mining techniques have been developed which apply association rule mining to the problem of classification. In this section, we study such association-based

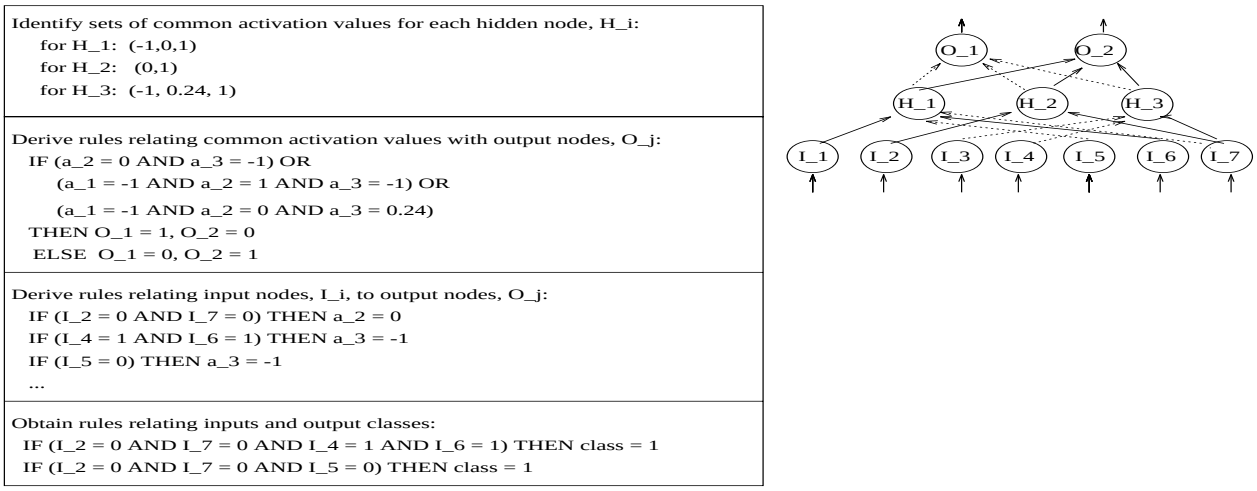


Figure 7.13: Rules can be extracted from training neural networks.

classification.

One method of association-based classification, called **associative classification**, consists of two steps. In the first step, association rules are generated using a modified version of the standard association rule mining algorithm known as Apriori. The second step constructs a classifier based on the association rules discovered.

Let D be the training data, and Y be the set of all classes in D. The algorithm maps categorical attributes to consecutive positive integers. Continuous attributes are discretized and mapped accordingly. Each data sample d in D then is represented by a set of (attribute, integer-value) pairs called **items**, and a class label y. Let I be the set of all items in D. A **class association rule (CAR)** is of the form $condset \Rightarrow y$, where $condset$ is a set of items ($condset \subseteq I$) and $y \in Y$. Such rules can be represented by **ruleitems** of the form <**condset, y**>.

A CAR has confidence c if c% of the samples in D that contain $condset$ belong to class y. A CAR has support s if s% of the samples in D contain $condset$ and belong to class y. The support count of a condset (**condsupCount**) is the number of samples in D that contain the condset. The rule count of a ruleitem (**rulesupCount**) is the number of samples in D that contain the condset and are labeled with class y. Ruleitems that satisfy minimum support are **frequent ruleitems**. If a set of ruleitems has the same condset, then the rule with the highest confidence is selected as the **possible rule (PR)** to represent the set. A rule satisfying minimum confidence is called **accurate**.

"How does associative classification work?"

The first step of the associative classification method finds the set of all PRs that are both frequent and accurate. These are the class association rules (CARs). A ruleitem whose condset contains k items is a **k-ruleitem**. The algorithm employs an iterative approach, similar to that described for Apriori in Section 5.2.1, where ruleitems are processed rather than itemsets. The algorithm scans the database, searching for the frequent k-ruleitems, for $k = 1, 2, ..$, until all frequent k-ruleitems have been found. One scan is made for each value of k. The k-ruleitems are used to explore (k+1)-ruleitems. In the first scan of the database, the count support of 1-ruleitems is determined, and the frequent 1-ruleitems are retained. The frequent 1-ruleitems, referred to as the set F_1, are used to generate candidate 2-ruleitems, C_2. Knowledge of frequent ruleitem properties is used to prune candidate ruleitems that cannot be frequent. This knowledge states that *all non-empty subsets of a frequent ruleitem must also be frequent.* The database is scanned a second time to compute the support counts of each candidate, so that the frequent 2-ruleitems (F_2) can be determined. This process repeats, where F_k is used to generate C_{k+1}, until no more frequent ruleitems are found. The frequent ruleitems that satisfy minimum confidence form the set of CARs. Pruning may be applied to this rule set.

The second step of the associative classification method processes the generated CARs in order to construct the classifier. Since the total number of rule subsets that would be examined in order to determine the most accurate set of rules can be huge, a heuristic method is employed. A precedence ordering among rules is defined where a rule r_i has greater precedence over a rule r_j (i.e., $r_i \succ r_j$) if (1) the confidence of r_i is greater than that of r_j, or (2) the confidences are the same, but r_i has greater support, or (3) the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j. In general, the algorithm selects a set of high precedence CARs to cover the samples in D. The algorithm requires slightly more than one pass over D in order to determine the final classifier. The classifier maintains the selected rules from high to low precedence order. When classifying a new sample, the first rule satisfying the sample is used to classify it. The classifier also contains a default rule, having lowest precedence, which specifies a default class for any new sample that is not satisfied by any other rule in the classifier.

In general, the above associative classification method was empirically found to be more accurate than C4.5 on several data sets. Each of the above two steps was shown to have linear scale-up.

Association rule mining based on clustering has also been applied to classification. The ARCS, or Association Rule Clustering System (Section 6.4.3) mines association rules of the form $A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$, where A_{quan1} and A_{quan2} are tests on quantitative attribute ranges (where the ranges are dynamically determined), and A_{cat} assigns a class label for a categorical attribute from the given training data. Association rules are plotted on a 2-D grid. The algorithm scans the grid, searching for rectangular clusters of rules. In this way, adjacent ranges of the quantitative attributes occurring within a rule cluster may be combined. The clustered association rules generated by ARCS were applied to classification, and their accuracy was compared to C4.5. In general, ARCS is slightly more accurate when there are outliers in the data. The accuracy of ARCS is related to the degree of discretization used. In terms of scalability, ARCS requires a constant amount of memory, regardless of the database size. C4.5 has exponentially higher execution times than ARCS, requiring the entire database, multiplied by some factor, to fit entirely in main memory. Hence, association rule mining is an important strategy for generating accurate and scalable classifiers.

7.7 Other classification methods

In this section, we give a brief description of a number of other classification methods. These methods include k-nearest neighbor classification, case-based reasoning, genetic algorithms, rough set and fuzzy set approaches. In general, these methods are less commonly used for classification in commercial data mining systems than the methods described earlier in this chapter. Nearest-neighbor classification, for example, stores all training samples, which may present difficulties when learning from very large data sets. Furthermore, many applications of case-based reasoning, genetic algorithms, and rough sets for classification are still in the prototype phase. These methods, however, are enjoying increasing popularity, and hence we include them here.

7.7.1 k-nearest neighbor classifiers

Nearest neighbor classifiers are based on learning by analogy. The training samples are described by n-dimensional numeric attributes. Each sample represents a point in an n-dimensional space. In this way, all of the training samples are stored in an n-dimensional pattern space. When given an unknown sample, a **k-nearest neighbor classifier** searches the pattern space for the k training samples that are closest to the unknown sample. These k training samples are the k "nearest neighbors" of the unknown sample. "Closeness" is defined in terms of Euclidean distance, where the Euclidean distance between two points, $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ is:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{7.19}$$

The unknown sample is assigned the most common class among its k nearest neighbors. When $k = 1$, the unknown sample is assigned the class of the training sample that is closest to it in pattern space.

Nearest neighbor classifiers are **instance-based** since they store all of the training samples. They can incur expensive computational costs when the number of potential neighbors (i.e., stored training samples) with which to compare a given unlabeled sample is great. Therefore, efficient indexing techniques are required. Unlike decision tree

induction and backpropagation, nearest neighbor classifiers assign equal weight to each attribute. This may cause confusion when there are many irrelevant attributes in the data.

Nearest neighbor classifiers can also be used for prediction, i.e., to return a real-valued prediction for a given unknown sample. In this case, the classifier returns the average value of the real-valued labels associated with the k nearest neighbors of the unknown sample.

7.7.2 Case-based reasoning

Case-based reasoning (CBR) classifiers are instanced-based. Unlike nearest neighbor classifiers, which store training samples as points in Euclidean space, the samples or "cases" stored by CBR are complex symbolic descriptions. Business applications of CBR include problem resolution for customer service help desks, for example, 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.

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 which 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.

Challenges in case-based reasoning include finding a good similarity metric (e.g., for matching subgraphs), developing efficient techniques for indexing training cases, and methods for combining solutions.

7.7.3 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.

7.7.4 Rough set theory

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 prior to its use.

Rough set theory is based on the establishment of **equivalence classes** within the given training data. All of the data samples 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

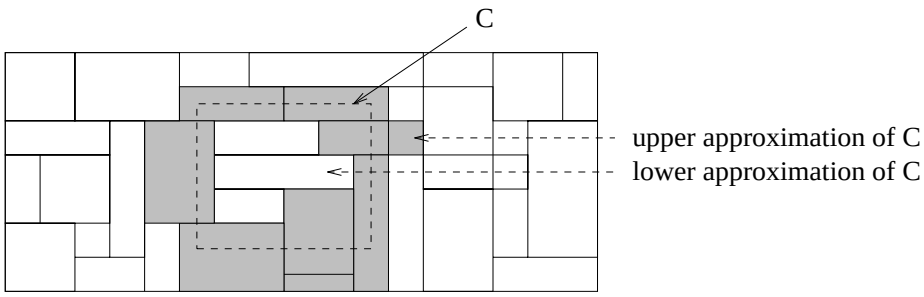


Figure 7.14: A rough set approximation of the set of samples of the class C using lower and upper approximation sets of C. The rectangular regions represent equivalence classes.

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 samples which, 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 samples which, 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 7.14, 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.

Rough sets can also be used for feature reduction (where attributes that do not contribute towards the classification of the given training data can be identified and removed), and relevance analysis (where the contribution or significance of each attribute is assessed with respect to the classification task). The problem of finding the minimal subsets (**reducts**) of attributes that can describe all of the concepts in the given data set is NP-hard. However, algorithms to reduce the computation intensity have been proposed. In one method, for example, a **discernibility matrix** is used which stores the differences between attribute values for each pair of data samples. Rather than searching on the entire training set, the matrix is instead searched to detect redundant attributes.

7.7.5 Fuzzy set approaches

Rule-based systems for classification have the disadvantage that they involve sharp cut-offs for continuous attributes. For example, consider Rule (7.20) below for customer credit application approval. The rule essentially says that applications for customers who have had a job for two or more years, and who have a high income (i.e., of more than \$50K) are approved.

$$IF\ (years\_employed >= 2) \wedge (income \geq 50K)\ THEN\ credit = approved. \tag{7.20}$$

By Rule (7.20), a customer who has had a job for at least 2 years will receive credit if her income is, say, \$51K, but not if it is \$50K. Such harsh thresholding may seem unfair. Instead, fuzzy logic can be introduced into the system to allow "fuzzy" thresholds or boundaries to be defined. Rather than having a precise cutoff between categories or sets, 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. Hence, with fuzzy logic, we can capture the notion that an income of \$50K is, to some degree, high, although not as high as an income of \$51K.

Fuzzy logic is useful for data mining systems performing classification. It provides the advantage of working at a high level of abstraction. In general, the use of fuzzy logic in rule-based systems involves the following:

- Attribute values are converted to fuzzy values. Figure 7.15 shows how values for the continuous attribute *income* are mapped into the discrete categories {*low, medium, high*}, as well as how the fuzzy membership or truth values are calculated. Fuzzy logic systems typically provide graphical tools to assist users in this step.
- For a given new sample, more than one fuzzy rule may apply. Each applicable rule contributes a vote for membership in the categories. Typically, the truth values for each predicted category are summed.

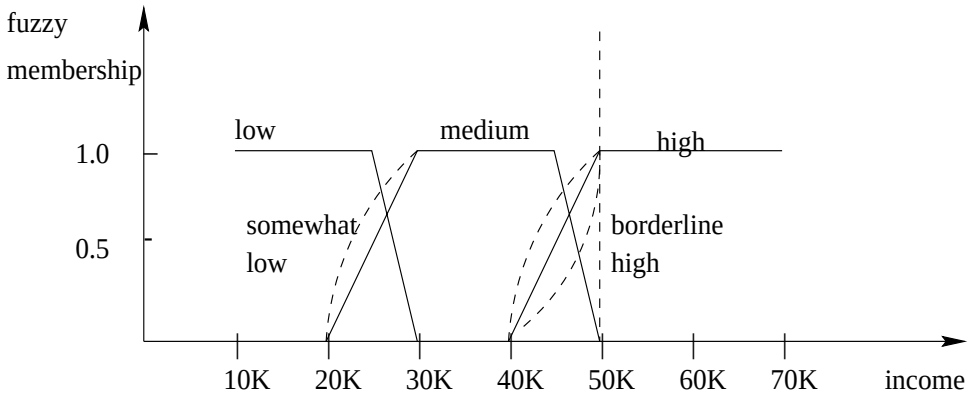


Figure 7.15: Fuzzy values for *income*.

- The sums obtained above are combined into a value that is returned by the system. This process may be done by weighting each category by its truth sum and multiplying by the mean truth value of each category. The calculations involved may be more complex, depending on the complexity of the fuzzy membership graphs.

Fuzzy logic systems have been used in numerous areas for classification, including health care and finance.

7.8 Prediction

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

The prediction of continuous values can be modeled by statistical techniques of *regression*. For example, we may like to develop a model to predict the salary of college graduates with 10 years of work experience, or the potential sales of a new product given its price. Many problems can be solved by *linear regression*, and even more can be tackled by applying transformations to the variables so that a nonlinear problem can be converted to a linear one. For reasons of space, we cannot give a fully detailed treatment of regression. Instead, this section provides an intuitive introduction to the topic. By the end of this section, you will be familiar with the ideas of linear, multiple, and nonlinear regression, as well as generalized linear models.

Several software packages exist to solve regression problems. Examples include SAS (*http://www.sas.com*), SPSS (*http://www.spss.com*), and S-Plus (*http://www.mathsoft.com*).

7.8.1 Linear and multiple regression

"What is linear regression?"

In **linear regression**, data are modeled using a straight line. Linear regression is the simplest form of regression. Bivariate linear regression models a random variable, Y (called a **response variable**), as a linear function of another random variable, X (called a **predictor variable**), i.e.,

$$Y = \alpha + \beta X, \tag{7.21}$$

where the variance of Y is assumed to be constant, and α and β are **regression coefficients** specifying the Y-intercept and slope of the line, respectively. These coefficients can be solved for by the **method of least squares**, which minimizes the error between the actual line separating the data and the estimate of the line. Given s samples or data points of the form (x_1, y_1), (x_2, y_2), .., (x_s, y_s), then the regression coefficients can be estimated using this method with Equations (7.22) and (7.23),

$$\beta = \frac{\sum_{i=1}^{s}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{s}(x_i - \bar{x})^2}, \tag{7.22}$$

$$\alpha = \bar{y} - \beta\bar{x}, \tag{7.23}$$

where $\bar{x}$ is the average of $x_1, x_2, .., x_s$, and $\bar{y}$ is the average of $y_1, y_2, .., y_s$. The coefficients α and β often provide good approximations to otherwise complicated regression equations.

X *years experience*	Y *salary (in $1000)*
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83

Table 7.7: Salary data.

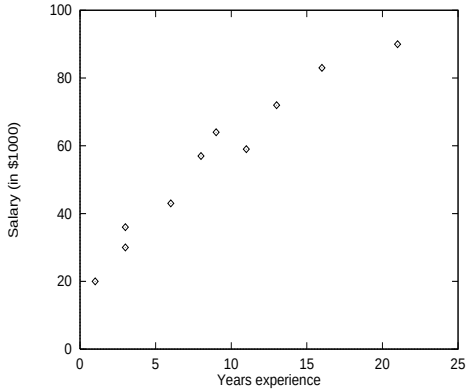


Figure 7.16: Plot of the data in Table 7.7 for Example 7.6. Although the points do not fall on a straight line, the overall pattern suggests a linear relationship between X (*years experience*) and Y (*salary*).

Example 7.6 Linear regression using the method of least squares. Table 7.7 shows a set of paired data where X is the number of years of work experience of a college graduate and Y is the corresponding salary of the graduate. A plot of the data is shown in Figure 7.16, suggesting 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 $Y = \alpha + \beta X$.

Given the above data, we compute $\bar{x} = 9.1$ and $\bar{y} = 55.4$. Substituting these values into Equation (7.22), we get

$\beta = \frac{(3-9.1)(30-55.4)+(8-9.1)(57-55.4)+...+(16-9.1)(83-55.4)}{(3-9.1)^2+(8-9.1)^2+...+(16-9.1)^2} = 3.7$

$\alpha = 55.4 - (3.7)(9.1) = 21.7$

Thus, the equation of the least squares line is estimated by $Y = 21.7 + 3.7X$. Using this equation, we can predict that the salary of a college graduate with, say, 10 years of experience is $58.7K. □

Multiple regression is an extension of linear regression involving more than one predictor variable. It allows response variable Y to be modeled as a linear function of a multidimensional feature vector. An example of a multiple regression model based on two predictor attributes or variables, X_1 and X_2, is shown in Equation (7.24).

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 \tag{7.24}$$

The method of least squares can also be applied here to solve for α, β_1, and β_2.

7.8.2 Nonlinear regression

"How can we model data that does not show a linear dependence? For example, what if a given response variable and predictor variables have a relationship that may be modeled by a polynomial function?"

Polynomial regression 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 7.7 Transformation of a polynomial regression model to a linear regression model. Consider a cubic polynomial relationship given by Equation (7.25).

$$Y = \alpha + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 \tag{7.25}$$

To convert this equation to linear form, we define new variables as shown in Equation (7.26).

$$X_1 = X \qquad X_2 = X^2 \qquad X_3 = X^3 \tag{7.26}$$

Equation (7.25) can then be converted to linear form by applying the above assignments, resulting in the equation $Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$, which is solvable by the method of least squares.

□

In Exercise 7, you are asked to find the transformations required to convert a nonlinear model involving a power function into a linear regression model.

Some models are intractably nonlinear (such as the sum of exponential terms, for example) and cannot be converted to a linear model. For such cases, it may be possible to obtain least-square estimates through extensive calculations on more complex formulae.

7.8.3 Other regression models

Linear regression is used to model continuous-valued functions. It is widely used, owing largely to its simplicity. *"Can it also be used to predict categorical labels?"* **Generalized linear models** represent the theoretical foundation on which linear regression can be applied to the modeling of categorical response variables. In generalized linear models, the variance of the response variable Y is a function of the mean value of Y, unlike in linear regression, where the variance of Y is constant. Common types of generalized linear models include **logistic regression** and **Poisson regression**. Logistic regression models the probability of some event occurring as a linear function of a set of predictor variables. Count data frequently exhibit a Poisson distribution and are commonly modeled using Poisson regression.

Log-linear models approximate *discrete* multidimensional probability distributions. They may be used to estimate the probability value associated with data cube cells. For example, suppose we are given data for the attributes *city, item, year*, and *sales*. In the log-linear method, all attributes must be categorical, hence continuous-valued attributes (like *sales*) must first be discretized. The method can then be used to estimate the probability of each cell in the 4-D base cuboid for the given attributes, based on the 2-D cuboids for *city* and *item*, *city* and *year*, *city* and *sales*, and the 3-D cuboid for *item, year*, and *sales*. In this way, an iterative technique can be used to build higher order data cubes from lower order ones. The technique scales up well to allow for many dimensions. Aside from prediction, the log-linear model is useful for data compression (since the smaller order cuboids together typically occupy less space than the base cuboid) and data smoothing (since cell estimates in the smaller order cuboids are less subject to sampling variations than cell estimates in the base cuboid).

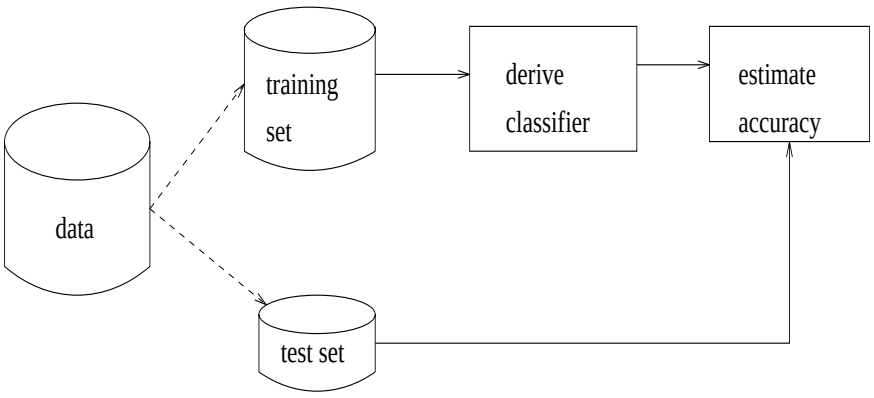


Figure 7.17: Estimating classifier accuracy with the holdout method.

7.9 Classifier accuracy

Estimating classifier accuracy is important in that it allows one to evaluate how accurately a given classifier will correctly label future data, i.e., data on which the classifier has not been trained. For example, if data from previous sales are used to train a classifier to predict customer purchasing behavior, we would like some estimate of how accurately the classifier can predict the purchasing behavior of future customers. Accuracy estimates also help in the comparison of different classifiers. In Section 7.9.1, we discuss techniques for estimating classifier accuracy, such as the *holdout* and *k-fold cross-validation* methods. Section 7.9.2 describes *bagging* and *boosting*, two strategies for increasing classifier accuracy. Section 7.9.3 discusses additional issues relating to classifier selection.

7.9.1 Estimating classifier accuracy

Using training data to derive a classifier and then to estimate the accuracy of the classifier can result in misleading over-optimistic estimates due to overspecialization of the learning algorithm (or model) to the data. Holdout and cross-validation are two common techniques for assessing classifier accuracy, based on randomly-sampled partitions of the given data.

In the **holdout** 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 classifier, whose accuracy is estimated with the test set (Figure 7.17). The estimate is pessimistic since only a portion of the initial data is used to derive the classifier. **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.

In **k-fold cross validation**, the initial data are randomly partitioned into k mutually exclusive subsets or "folds", $S_1, S_2, ..., S_k$, each of approximately equal size. Training and testing is performed k times. In iteration i, the subset S_i is reserved as the test set, and the remaining subsets are collectively used to train the classifier. That is, the classifier of the first iteration is trained on subsets $S_2, .., S_k$, and tested on S_1; the classifier of the section iteration is trained on subsets $S_1, S_3, .., S_k$, and tested on S_2; and so on. The accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of samples in the initial data. In **stratified cross-validation**, the folds are stratified so that the class distribution of the samples in each fold is approximately the same as that in the initial data.

Other methods of estimating classifier accuracy include **bootstrapping**, which samples the given training instances uniformly *with replacement*, and **leave-one-out**, which is k-fold cross validation with k set to s, the number of initial samples. In general, stratified 10-fold cross-validation is recommended for estimating classifier accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

The use of such techniques to estimate classifier accuracy increases the overall computation time, yet is useful for selecting among several classifiers.

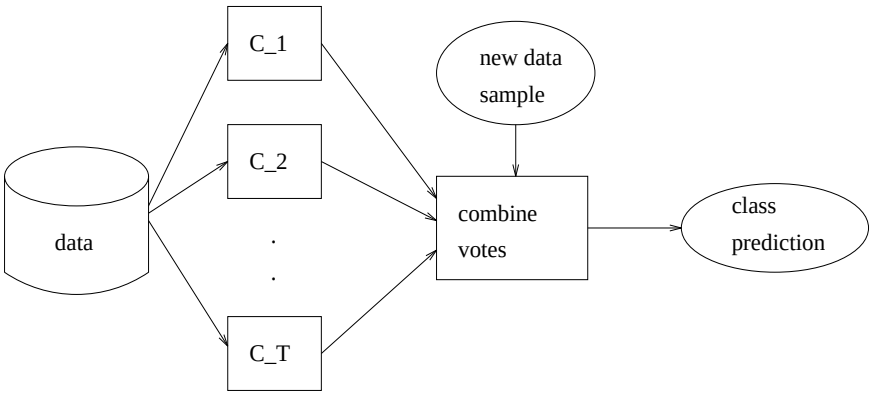


Figure 7.18: Increasing classifier accuracy: Bagging and boosting each generate a set of classifiers, $C_1, C_2, .., C_T$. Voting strategies are used to combine the class predictions for a given unknown sample.

7.9.2 Increasing classifier accuracy

In the previous section, we studied methods of estimating classifier accuracy. In Section 7.3.2, we saw how pruning can be applied to decision tree induction to help improve the accuracy of the resulting decision trees. Are there *general* techniques for improving classifier accuracy?

The answer is yes. **Bagging** (or *boostrap aggregation*) and **boosting** are two such techniques (Figure 7.18). Each combines a series of T learned classifiers, $C_1, C_2, .., C_T$, with the aim of creating an improved composite classifier, $C*$.

"How do these methods work?" 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 the others, you may choose this as the final or best diagnosis. Now replace each doctor by a classifier, and you have the intuition behind bagging. Suppose instead, that 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. Let us have a closer look at these two techniques.

Given a set S of s samples, bagging works as follows. For iteration t ($t = 1, 2, .., T$), a training set S_t is sampled with replacement from the original set of samples, S. Since sampling with replacement is used, some of the original samples of S may not be included in S_t, while others may occur more than once. A classifier C_t is learned for each training set, S_t. To classify an unknown sample, X, each classifier C_t returns its class prediction, which counts as one vote. The bagged classifier, $C*$, counts the votes and assigns the class with the most votes to X. Bagging can be applied to the prediction of continuous values by taking the average value of each vote, rather than the majority.

In boosting, weights are assigned to each training sample. A series of classifiers is learned. After a classifier C_t is learned, the weights are updated to allow the subsequent classifier, C_{t+1}, to "pay more attention" to the misclassification errors made by C_t. The final boosted classifier, $C*$, 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.

7.9.3 Is accuracy enough to judge a classifier?

In addition to accuracy, classifiers can be compared with respect to their speed, robustness (e.g., accuracy on noisy data), scalability, and interpretability. Scalability can be evaluated by assessing the number of I/O operations involved for a given classification algorithm on data sets of increasingly large size. Interpretability is subjective, although we may use objective measurements such as the complexity of the resulting classifier (e.g., number of tree nodes for decision trees, or number of hidden units for neural networks, etc.) in assessing it.

"Is it always possible to assess accuracy?" In classification problems, it is commonly assumed that all objects are uniquely classifiable, i.e., that each training sample can belong to only one class. As we have discussed above, classification algorithms can then be compared according to their accuracy. However, owing to the wide diversity

of data in large databases, it is not always reasonable to assume that all objects are uniquely classifiable. Rather, it is more probable to assume that each object may belong to more than one class. How then, can the accuracy of classifiers on large databases be measured? The accuracy measure is not appropriate, since it does not take into account the possibility of samples belonging to more than one class.

Rather than returning a class label, it is useful to return a probability class distribution. Accuracy measures may then use a **second guess** heuristic whereby a class prediction is judged as correct if it agrees with the first or second most probable class. Although this does take into consideration, in some degree, the non-unique classification of objects, it is not a complete solution.

7.10 Summary

- Classification and prediction are two forms of data analysis which can be used to extract models describing important data classes or to predict future data trends. While **classification** predicts categorical labels (classes), **prediction** models continuous-valued functions.
- Preprocessing of the data in preparation for classification and prediction can involve **data cleaning** to reduce noise or handle missing values, **relevance analysis** to remove irrelevant or redundant attributes, and **data transformation**, such as generalizing the data to higher level concepts, or normalizing the data.
- Predictive accuracy, computational speed, robustness, scalability, and interpretability are five **criteria** for the evaluation of classification and prediction methods.
- **ID3** and **C4.5** are greedy algorithms for the induction of decision trees. Each algorithm uses an information theoretic measure to select the attribute tested for each non-leaf node in the tree. **Pruning** algorithms attempt to improve accuracy by removing tree branches reflecting noise in the data. Early decision tree algorithms typically assume that the data are memory resident - a limitation to data mining on large databases. Since then, several scalable algorithms have been proposed to address this issue, such as SLIQ, SPRINT, and RainForest. Decision trees can easily be converted to classification IF-THEN rules.
- **Naive Bayesian classification** and **Bayesian belief networks** are based on Bayes theorem of posterior probability. Unlike naive Bayesian classification (which assumes class conditional independence), Bayesian belief networks allow class conditional independencies to be defined between subsets of variables.
- **Backpropagation** is a neural network algorithm for classification which employs a method of gradient descent. It searches for a set of weights which can model the data so as to minimize the mean squared distance between the network's class prediction and the actual class label of data samples. Rules may be extracted from trained neural networks in order to help improve the interpretability of the learned network.
- **Association mining** techniques, which search for frequently occurring patterns in large databases, can be applied to and used for classification.
- Nearest neighbor classifiers and cased-based reasoning classifiers are **instance-based** methods of classification in that they store all of the training samples in pattern space. Hence, both require efficient indexing techniques. In **genetic algorithms**, populations of rules "evolve" via operations of crossover and mutation until all rules within a population satisfy a specified threshold. **Rough set theory** can be used to approximately define classes that are not distinguishable based on the available attributes. **Fuzzy set** approaches replace "brittle" threshold cutoffs for continuous-valued attributes with degree of membership functions.
- Linear, nonlinear, and generalized linear models of **regression** can be used for prediction. Many nonlinear problems can be converted to linear problems by performing transformations on the predictor variables.
- Data warehousing techniques, such as attribute-oriented induction and the use of multidimensional data cubes, can be integrated with classification methods in order to allow fast **multilevel mining**. Classification tasks may be specified using a data mining query language, promoting interactive data mining.
- **Stratified k-fold cross validation** is a recommended method for estimating classifier accuracy. **Bagging** and **boosting** methods can be used to increase overall classification accuracy by learning and combining a series of individual classifiers.

Exercises

1. Table 7.8 consists of training data from an employee database. The data have been generalized. For a given row entry, *count* represents the number of data tuples having the values for *department, status, age*, and *salary* given in that row.

department	*status*	*age*	*salary*	*count*
sales	senior	31-35	45-50K	30
sales	junior	26-30	25-30K	40
sales	junior	31-35	30-35K	40
systems	junior	21-25	45-50K	20
systems	senior	31-35	65-70K	5
systems	junior	26-30	45-50K	3
systems	senior	41-45	65-70K	3
marketing	senior	36-40	45-50K	10
marketing	junior	31-35	40-45K	4
secretary	senior	46-50	35-40K	4
secretary	junior	26-30	25-30K	6

Table 7.8: Generalized relation from an employee database.

Let *salary* be the class label attribute.

   (a) How would you modify the ID3 algorithm to take into consideration the *count* of each data tuple (i.e., of each row entry)?

   (b) Use your modified version of ID3 to construct a decision tree from the given data.

   (c) Given a data sample with the values *"systems"*, *"junior"*, and *"20-24"* for the attributes *department, status*, and *age*, respectively, what would a naive Bayesian classification of the *salary* for the sample be?

   (d) Design a multilayer feed-forward neural network for the given data. Label the nodes in the input and output layers.

   (e) Using the multilayer feed-forward neural network obtained above, show the weight values after one iteration of the backpropagation algorithm given the training instance *"(sales, senior, 31-35, 45-50K)"*. Indicate your initial weight values and the learning rate used.

2. Write an algorithm for k-nearest neighbor classification given k, and n, the number of attributes describing each sample.

3. What is a drawback of using a separate set of samples to evaluate pruning?

4. Given a decision tree, you have the option of (a) converting the decision tree to rules and then pruning the resulting rules, or (b) pruning the decision tree and then converting the pruned tree to rules? What advantage does (a) have over (b)?

5. ADD QUESTIONS ON OTHER CLASSIFICATION METHODS.

6. Table 7.9 shows the mid-term and final exam grades obtained for students in a database course.

   (a) Plot the data. Do X and Y seem to have a linear relationship?

   (b) Use the method of least squares to find an equation for the prediction of a student's final exam grade based on the student's mid-term grade in the course.

   (c) Predict the final exam grade of a student who received an 86 on the mid-term exam.

7. Some nonlinear regression models can be converted to linear models by applying transformations to the predictor variables. Show how the nonlinear regression equation $Y = \alpha X^{\beta}$ can be converted to a linear regression equation solvable by the method of least squares.

X *mid-term exam*	Y *final exam*
72	84
50	63
81	77
74	78
94	90
86	75
59	49
83	79
65	77
33	52
88	74
81	90

Table 7.9: Mid-term and final exam grades.

8. It is difficult to assess classification accuracy when individual data objects may belong to more than one class at a time. In such cases, comment on what criteria you would use to compare different classifiers modeled after the same data.

Bibliographic Notes

Classification from a machine learning perspective is described in several books, such as Weiss and Kulikowski [136], Michie, Spiegelhalter, and Taylor [88], Langley [67], and Mitchell [91]. Weiss and Kulikowski [136] compare classification and prediction methods from many different fields, in addition to describing practical techniques for the evaluation of classifier performance. Many of these books describe each of the basic methods of classification discussed in this chapter. Edited collections containing seminal articles on machine learning can be found in Michalksi, Carbonell, and Mitchell [85, 86], Kodratoff and Michalski [63], Shavlik and Dietterich [123], and Michalski and Tecuci [87]. For a presentation of machine learning with respect to data mining applications, see Michalski, Bratko, and Kubat [84].

The C4.5 algorithm is described in a book by J. R. Quinlan [108]. The book gives an excellent presentation of many of the issues regarding decision tree induction, as does a comprehensive survey on decision tree induction by Murthy [94]. Other algorithms for decision tree induction include the predecessor of C4.5, ID3 (Quinlan [104]), CART (Breiman et al. [11]), FACT (Loh and Vanichsetakul [76]), QUEST (Loh and Shih [75]), and PUBLIC (Rastogi and Shim [111]). Incremental versions of ID3 include ID4 (Schlimmer and Fisher [120]) and ID5 (Utgoff [132]). In addition, INFERULE (Uthurusamy, Fayyad, and Spangler [133]) learns decision trees from inconclusive data. KATE (Manago and Kodratoff [80]) learns decision trees from complex structured data. Decision tree algorithms that address the scalability issue in data mining include SLIQ (Mehta, Agrawal, and Rissanen [81]), SPRINT (Shafer, Agrawal, and Mehta [121]), RainForest (Gehrke, Ramakrishnan, and Ganti [43]), and Kamber et al. [61]. Earlier approaches described include [16, 17, 18]. For a comparison of attribute selection measures for decision tree induction, see Buntine and Niblett [15], and Murthy [94]. For a detailed discussion on such measures, see Kononenko and Hong [65].

There are numerous algorithms for decision tree pruning, including cost complexity pruning (Breiman et al. [11]), reduced error pruning (Quinlan [105]), and pessimistic pruning (Quinlan [104]). PUBLIC (Rastogi and Shim [111]) integrates decision tree construction with tree pruning. MDL-based pruning methods can be found in Quinlan and Rivest [110], Mehta, Agrawal, and Rissanen [82], and Rastogi and Shim [111]. Others methods include Niblett and Bratko [96], and Hosking, Pednault, and Sudan [55]. For an empirical comparison of pruning methods, see Mingers [89], and Malerba, Floriana, and Semeraro [79].

For the extraction of rules from decision trees, see Quinlan [105, 108]. Rather than generating rules by extracting them from decision trees, it is also possible to induce rules directly from the training data. Rule induction algorithms

include CN2 (Clark and Niblett [21]), AQ15 (Hong, Mozetic, and Michalski [54]), ITRULE (Smyth and Goodman [126]), FOIL (Quinlan [107]), and Swap-1 (Weiss and Indurkhya [134]). Decision trees, however, tend to be superior in terms of computation time and predictive accuracy. Rule refinement strategies which identify the most interesting rules among a given rule set can be found in Major and Mangano [78].

For descriptions of data warehousing and multidimensional data cubes, see Harinarayan, Rajaraman, and Ullman [48], and Berson and Smith [8], as well as Chapter 2 of this book. Attribution-oriented induction (AOI) is presented in Han and Fu [45], and summarized in Chapter 5. The integration of AOI with decision tree induction is proposed in Kamber et al. [61]. The precision or classification threshold described in Section 7.3.6 is used in Agrawal et al. [2] and Kamber et al. [61].

Thorough presentations of Bayesian classification can be found in Duda and Hart [32], a classic textbook on pattern recognition, as well as machine learning textbooks such as Weiss and Kulikowski [136] and Mitchell [91]. For an analysis of the predictive power of naive Bayesian classifiers when the class conditional independence assumption is violated, see Domingos and Pazzani [31]. Experiments with kernel density estimation for continuous-valued attributes, rather than Gaussian estimation have been reported for naive Bayesian classifiers in John [59]. Algorithms for inference on belief networks can be found in Russell and Norvig [118] and Jensen [58]. The method of gradient descent, described in Section 7.4.4 for learning Bayesian belief networks, is given in Russell et al. [117]. The example given in Figure 7.8 is adapted from Russell et al. [117]. Alternative strategies for learning belief networks with hidden variables include the EM algorithm (Lauritzen [68]), and Gibbs sampling (York and Madigan [139]). Solutions for learning the belief network structure from training data given observable variables are proposed in [22, 14, 50].

The backpropagation algorithm was presented in Rumelhart, Hinton, and Williams [115]. Since then, many variations have been proposed involving, for example, alternative error functions (Hanson and Burr [47]), dynamic adjustment of the network topology (Fahlman and Lebiere [35], Le Cun, Denker, and Solla [70]), and dynamic adjustment of the learning rate and momentum parameters (Jacobs [56]). Other variations are discussed in Chauvin and Rumelhart [19]. Books on neural networks include [116, 49, 51, 40, 19, 9, 113]. Many books on machine learning, such as [136, 91], also contain good explanations of the backpropagation algorithm. There are several techniques for extracting rules from neural networks, such as [119, 42, 131, 40, 7, 77, 25, 69]. The method of rule extraction described in Section 7.5.4 is based on Lu, Setiono, and Liu [77]. Critiques of techniques for rule extraction from neural networks can be found in Andrews, Diederich, and Tickle [5], and Craven and Shavlik [26]. An extensive survey of applications of neural networks in industry, business, and science is provided in Widrow, Rumelhart, and Lehr [137].

The method of associative classification described in Section 7.6 was proposed in Liu, Hsu, and Ma [74]. ARCS was proposed in Lent, Swami, and Widom [73], and is also described in Chapter 6.

Nearest neighbor methods are discussed in many statistical texts on classification, such as Duda and Hart [32], and James [57]. Additional information can be found in Cover and Hart [24] and Fukunaga and Hummels [41]. References on case-based reasoning (CBR) include the texts [112, 64, 71], as well as [1]. For a survey of business applications of CBR, see Allen [4]. Examples of other applications include [6, 129, 138]. For texts on genetic algorithms, see [44, 83, 90]. Rough sets were introduced in Pawlak [97, 99]. Concise summaries of rough set theory in data mining include [141, 20]. Rough sets have been used for feature reduction and expert system design in many applications, including [98, 72, 128]. Algorithms to reduce the computation intensity in finding reducts have been proposed in [114, 125]. General descriptions of fuzzy logic can be found in [140, 8, 20].

There are many good textbooks which cover the techniques of regression. Example include [57, 30, 60, 28, 52, 95, 3]. The book by Press et al. [101] and accompanying source code contain many statistical procedures, such as the method of least squares for both linear and multiple regression. Recent nonlinear regression models include projection pursuit and MARS (Friedman [39]). Log-linear models are also known in the computer science literature as *multiplicative models*. For log-linear models from a computer science perspective, see Pearl [100]. Regression trees (Breiman et al. [11]) are often comparable in performance with other regression methods, particularly when there exist many higher order dependencies among the predictor variables.

Methods for data cleaning and data transformation are discussed in Pyle [102], Kennedy et al. [62], Weiss and Indurkhya [134], and Chapter 3 of this book. Issues involved in estimating classifier accuracy are described in Weiss and Kulikowski [136]. The use of stratified 10-fold cross-validation for estimating classifier accuracy is recommended over the holdout, cross-validation, leave-one-out (Stone [127]), and bootstrapping (Efron and Tibshirani [33]) methods, based on a theoretical and empirical study by Kohavi [66]. Bagging is proposed in Breiman [10]. The boosting technique of Freund and Schapire [38] has been applied to several different classifiers, including decision tree induction (Quinlan [109]), and naive Bayesian classification (Elkan [34]).

The University of California at Irvine (UCI) maintains a Machine Learning Repository of data sets for the development and testing of classification algorithms. For information on this repository, see *http://www.ics.uci.edu/~mlearn/MLRepository.html*.

No classification method is superior over all others for all data types and domains. Empirical comparisons on classification methods include [106, 37, 135, 122, 130, 12, 23, 27, 92, 29].

Bibliography

[1] A. Aamodt and E. Plazas. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Comm.*, 7:39–52, 1994.

[2] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 560–573, Vancouver, Canada, August 1992.

[3] A. Agresti. *An Introduction to Categorical Data Analysis.* John Wiley & Sons, 1996.

[4] B. P. Allen. Case-based reasoning: Business applications. *Comm. ACM*, 37:40–42, 1994.

[5] R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8, 1995.

[6] K. D. Ashley. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals.* Cambridge, MA: MIT Press, 1990.

[7] S. Avner. Discovery of comprehensible symbolic rules in a neural network. In *Intl. Symposium on Intelligence in Neural and Bilogical Systems*, pages 64–67, 1995.

[8] A. Berson and S. J. Smith. *Data Warehousing, Data Mining, and OLAP.* New York: McGraw-Hill, 1997.

[9] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford, UK: Oxford University Press, 1995.

[10] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[11] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[12] C. E. Brodley and P. E. Utgoff. Multivariate versus univariate decision trees. In *Technical Report 8*, Department of Computer Science, Univ. of Massachusetts, 1992.

[13] W. Buntine. Graphical models for discovering knowledge. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 59–82. AAAI/MIT Press, 1996.

[14] W. L. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[15] W. L. Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[16] J. Catlett. *Megainduction: Machine Learning on Very large Databases.* PHD Thesis, University of Sydney, 1991.

[17] P. K. Chan and S. J. Stolfo. Experiments on multistrategy learning by metalearning. In *Proc. 2nd. Int. Conf. Information and Knowledge Management*, pages 314–323, 1993.

[18] P. K. Chan and S. J. Stolfo. Metalearning for multistrategy and parallel learning. In *Proc. 2nd. Int. Workshop on Multistrategy Learning*, pages 150–165, 1993.

[19] Y. Chauvin and D. Rumelhart. *Backpropagation: Theory, Architectures, and Applications.* Hillsdale, NJ: Lawrence Erlbaum Assoc., 1995.

[20] K. Cios, W. Pedrycz, and R. Swiniarski. *Data Mining Methods for Knowledge Discovery.* Boston: Kluwer Academic Publishers, 1998.

[21] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[22] G. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[23] V. Corruble, D. E. Brown, and C. L. Pittard. A comparison of decision classifiers with backpropagation neural networks for multimodal classification problems. *Patern Recognition*, 26:953–961, 1993.

[24] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Information Theory*, 13:21–27, 1967.

[25] M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In D. Touretzky and M. Mozer M. Hasselmo, editors, *Advances in Neural Information Processing Systems.* Cambridge, MA: MIT Press, 1996.

[26] M. W. Craven and J. W. Shavlik. Using neural networks in data mining. *Future Generation Computer Systems*, 13:211–229, 1997.

[27] S. P. Curram and J. Mingers. Neural networks, decision tree induction and discriminant analysis: An empirical comparison. *J. Operational Research Society*, 45:440–450, 1994.

[28] J. L. Devore. *Probability and Statistics for Engineering and the Science, 4th ed.* Duxbury Press, 1995.

[29] T. G. Dietterich, H. Hild, and G. Bakiri. A comparison of ID3 and backpropagation for english text-to-speech mapping. *Machine Learning*, 18:51–80, 1995.

[30] A. J. Dobson. *An Introduction to Generalized Linear Models.* Chapman and Hall, 1990.

[31] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *Proc. 13th Intl. Conf. Machine Learning*, pages 105–112, 1996.

[32] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley: New York, 1973.

[33] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* New York: Chapman & Hall, 1993.

[34] C. Elkan. Boosting and naive bayesian learning. In *Technical Report CS97-557*, Dept. of Computer Science and Engineering, Univ. Calif. at San Diego, Sept. 1997.

[35] S. Fahlman and C. Lebiere. The cascade-correlation learning algorithm. In *Technical Report CMU-CS-90-100*, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1990.

[36] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[37] D. H. Fisher and K. B. McKusick. An empirical comparison of ID3 and back-propagation. In *Proc. 11th Intl. Joint Conf. AI*, pages 788–793, San Mateo, CA: Morgan Kaufmann, 1989.

[38] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[39] J. Friedman. Multivariate adaptive regression. *Annalsof Statistics*, 19:1–141, 1991.

[40] L. Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill, 1994.

[41] K. Fukunaga and D. Hummels. Bayes error estimation using parzen and k-nn procedure. In *IEEE Trans. Pattern Analysis and Machine Learning*, pages 634–643, 1987.

[42] S. I. Gallant. *Neural Network Learning and Expert Systems.* Cambridge, MA: MIT Press, 1993.

[43] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest: A framework for fast decision tree construction of large datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 416–427, New York, NY, August 1998.

[44] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley: Reading, MA, 1989.

[45] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399–421. AAAI/MIT Press, 1996.

[46] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaïane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, August 1996.

[47] S. J. Hanson and D. J. Burr. Minkowski back-propagation: Learning in connectionist models with non-euclidean error signals. In *Neural Information Processing Systems*, American Institute of Physics, 1988.

[48] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[49] R. Hecht-Nielsen. *Neurocomputing.* Reading, MA: Addison Wesley, 1990.

[50] D. Heckerman, D. Geiger, and D. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197, 1995.

[51] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley: Reading, MA., 1991.

[52] R. V. Hogg and A. T. Craig. *Introduction to Mathematical Statistics, 5th ed.* Prentice Hall, 1995.

[53] L. B. Holder. Intermediate decision trees. In *Proc. 14th Int. Joint Conf. Artificial Intelligence (IJCAI95)*, pages 1056–1062, Montreal, Canada, Aug. 1995.

[54] J. Hong, I. Mozetic, and R. S. Michalski. AQ15: Incremental learning of attribute-based descriptions from examples, the method and user's guide. In *Report ISG 85-5, UIUCDCS-F-86-949,*, Department of Computer Science, University of Illinois at Urbana-Champagin, 1986.

[55] J. Hosking, E. Pednault, and M. Sudan. A statistical perspective on data mining. In *Future Generation Computer Systems*, pages 117–134, ???, 1997.

[56] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[57] M. James. *Classification Algorithms.* John Wiley, 1985.

[58] F. V. Jensen. *An Introduction to Bayesian Networks.* Springer Verlag, 1996.

[59] G. H. John. *Enhancements to the Data Mining Process.* Ph.D. Thesis, Computer Science Dept., Stanford Univeristy, 1997.

[60] R. A. Johnson and D. W. Wickern. *Applied Multivariate Statistical Analysis, 3rd ed.* Prentice Hall, 1992.

[61] M. Kamber, L. Winstone, W. Gong, S. Cheng, and J. Han. Generalization and decision tree induction: Efficient classification in data mining. In *Proc. 1997 Int. Workshop Research Issues on Data Engineering (RIDE'97)*, pages 111–120, Birmingham, England, April 1997.

[62] R. L Kennedy, Y. Lee, B. Van Roy, C. D. Reed, and R. P. Lippman. *Solving Data Mining Problems Through Pattern Recognition.* Upper Saddle River, NJ: Prentice Hall, 1998.

[63] Y. Kodratoff and R. S. Michalski. *Machine Learning, An Artificial Intelligence Approach, Vol. 3.* Morgan Kaufmann, 1990.

[64] J. L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[65] I. Kononenko and S. J. Hong. Attribute selection for modeling. In *Future Generation Computer Systems*, pages 181–195, ???, 1997.

[66] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proc. 4th Int. Symp. Large Spatial Databases (SSD'95)*, pages 47–66, Portland, Maine, Aug. 1995.

[67] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.

[68] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[69] S. Lawrence, C. L Giles, and A. C. Tsoi. Symbolic conversion, grammatical inference and rule extraction for foreign exchange rate prediction. In Y. Abu-Mostafa and A. S. Weigend P. N Refenes, editors, *Neural Networks in the Captial Markets*. Singapore: World Scientific, 1997.

[70] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in neural Information Processing Systems, 2*, pages San Mateo, CA: Morgan Kaufmann. Cambridge, MA: MIT Press, 1990.

[71] D. B. Leake. CBR in context: The present and future. In D. B. Leake, editor, *Cased-Based Reasoning: Experience, Lessons, and Future Directions*, pages 3–30. Menlo Park, CA: AAAI Press, 1996.

[72] A. Lenarcik and Z. Piasta. Probabilistic rough classifiers with mixture of discrete and continuous variables. In T. Y. Lin N. Cercone, editor, *Rough Sets and Data Mining: Analysis for Imprecise Data*, pages 373–383. Boston: Kluwer, 1997.

[73] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, pages 220–231, Birmingham, England, April 1997.

[74] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, New York, NY, August 1998.

[75] W. Y. Loh and Y.S. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

[76] W. Y. Loh and N. Vanichsetakul. Tree-structured classificaiton via generalized discriminant analysis. *Journal of the American Statistical Association*, 83:715–728, 1988.

[77] H. Lu, R. Setiono, and H. Liu. Neurorule: A connectionist approach to data mining. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 478–489, Zurich, Switzerland, Sept. 1995.

[78] J. Major and J. Mangano. Selecting among rules induced from a hurricane database. *Journal of Intelligent Information Systems*, 4:39–52, 1995.

[79] D. Malerba, E. Floriana, and G. Semeraro. A further comparison of simplification methods for decision tree induction. In D.Fisher H. Lenz, editor, *Learning from Data: AI and Statistics*. Springer-Verlag, 1995.

[80] M. Manago and Y. Kodratoff. Induction of decision trees from complex structured data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 289–306. AAAI/MIT Press, 1991.

[81] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. 1996 Int. Conf. Extending Database Technology (EDBT'96)*, Avignon, France, March 1996.

[82] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *Proc. 1st Intl. Conf. Knowledge Discovery and Data Mining (KDD95)*, Montreal, Canada, Aug. 1995.

[83] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1992.

[84] R. S. Michalski, I. Bratko, and M. Kubat. *Machine Learning and Data Mining: Methods and Applications*. John Wiley & Sons, 1998.

[85] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 1.* Morgan Kaufmann, 1983.

[86] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 2.* Morgan Kaufmann, 1986.

[87] R. S. Michalski and G. Tecuci. *Machine Learning, A Multistrategy Approach, Vol. 4.* Morgan Kaufmann, 1994.

[88] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[89] J. Mingers. An empirical comparison of pruning methods for decision-tree induction. *Machine Learning*, 4:227–243, 1989.

[90] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[91] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[92] D. Mitchie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* New York: Ellis Horwood, 1994.

[93] R. Mooney, J. Shavlik, G. Towell, and A. Grove. An experimental comparison of symbolic and connectionist learning algorithms. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI'89)*, pages 775–787, Detroit, MI, Aug. 1989.

[94] S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1998.

[95] J. Neter, M. H. Kutner, C. J. Nachtsheim, and L. Wasserman. *Applied Linear Statistical Models, 4th ed.* Irwin: Chicago, 1996.

[96] T. Niblett and I. Bratko. Learning decision rules in noisy domains. In M. A. Bramer, editor, *Expert Systems '86: Research and Development in Expert Systems III*, pages 25–34. British Computer Society Specialist Group on Expert Systems, Dec. 1986.

[97] Z. Pawlak. Rough sets. *Intl. J. Computer and Information Sciences*, 11:341–356, 1982.

[98] Z. Pawlak. On learning - rough set approach. In *Lecture Notes 208*, pages 197–227, New York: Springer-Verlag, 1986.

[99] Z. Pawlak. *Rough Sets, Theoretical Aspects of Reasonign about Data.* Boston: Kluwer, 1991.

[100] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Palo Alto, CA: Morgan Kauffman, 1988.

[101] W. H. Press, S. A. Teukolsky, V. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing.* Cambridge, MA: Cambridge University Press, 1996.

[102] D. Pyle. *Data Preparation for Data Mining.* Morgan Kaufmann, 1999.

[103] J. R. Quinlan. The effect of noise on concept learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 2*, pages 149–166. Morgan Kaufmann, 1986.

[104] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[105] J. R. Quinlan. Simplifying decision trees. *Internation Journal of Man-Machine Studies*, 27:221–234, 1987.

[106] J. R. Quinlan. An empirical comparison of genetic and decision-tree classifiers. In *Proc. 5th Intl. Conf. Machine Learning*, pages 135–141, San Mateo, CA: Morgan Kaufmann, 1988.

[107] J. R. Quinlan. Learning logic definitions from relations. *Machine Learning*, 5:139–166, 1990.

[108] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[109] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proc. 13th Natl. Conf. on Artificial Intelligence (AAAI'96)*, volume 1, pages 725–730, Portland, OR, Aug. 1996.

[110] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, March 1989.

[111] R. Rastogi and K. Shim. Public: A decision tree classifer that integrates building and pruning. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 404–415, New York, NY, August 1998.

[112] C. Riesbeck and R. Schank. *Inside Case-Based Reasoning.* Hillsdale, NJ: Lawrence Erlbaum, 1989.

[113] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge: Cambridge University Press, 1996.

[114] S. Romanski. Operations on families of sets for exhaustive search, given a monotonic function. In *Proc. 3rd Intl. Conf on Data and Knowledge Bases, C. Beeri et. al. (eds.),*, pages 310–322, Jerusalem, Israel, 1988.

[115] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart J. L. McClelland, editor, *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.

[116] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.

[117] S. Russell, J. Binder, D. Koller, and K. Kanazawa. Local learning in probabilistic networks with hidden variables. In *Proc. 14th Joint Int. Conf. on Artificial Intelligence (IJCAI'95)*, volume 2, pages 1146–1152, Montreal, Canada, Aug. 1995.

[118] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[119] K. Saito and R. Nakano. Medical diagnostic expert system based on PDP model. In *Proc. IEEE International Conf. on Neural Networks*, volume 1, pages 225–262, San Mateo, CA, 1988.

[120] J. C. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proc. 5th Natl. Conf. Artificial Intelligence*, pages 496–501, Phildelphia, PA: Morgan Kaufmann, 1986.

[121] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 544–555, Bombay, India, Sept. 1996.

[122] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–144, 1991.

[123] J.W. Shavlik and T.G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, 1990.

[124] A. Skowron, L. Polowski, and J. Komorowski. Learning tolerance relations by boolean descriptors: Automatic feature extraction from data tables. In *Proc. 4th Intl. Workshop on Rough Sets, Fuzzy Sets, and Machine Discovery, S. Tsumoto et. al. (eds.)*, pages 11–17, University of Tokyo, 1996.

[125] A. Skowron and C. Rauszer. The discernibility matrices and functions in information systems. In R. Slowinski, editor, *Intelligent Decision Support, Handbook of Applications and Advances of the Rough Set Theory*, pages 331–362. Boston: Kluwer, 1992.

[126] P. Smyth and R.M. Goodman. Rule induction using information theory. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 159–176. AAAI/MIT Press, 1991.

[127] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

[128] R. Swiniarski. Rough sets and principal component analysis and their applications in feature extraction and seletion, data model building and classification. In S. Pal A. Skowron, editor, *Fuzzy Sets, Rough Sets and Decision Making Processes.* New York: Springer-Verlag, 1998.

[129] K. Sycara, R. Guttal, J. Koning, S. Narasimhan, and D. Navinchandra. CADET: A case-based synthesis tool for engineering design. *Int. Journal of Expert Systems*, 4:157–188, 1992.

[130] S. B. Thrun et al. The monk's problems: A performance comparison of different learning algorithms. In *Technical Report CMU-CS-91-197*, Department of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, 1991.

[131] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, Oct. 1993.

[132] P. E. Utgoff. An incremental ID3. In *Proc. Fifth Int. Conf. Machine Learning*, pages 107–120, San Mateo, California, 1988.

[133] R. Uthurusamy, U. M. Fayyad, and S. Spangler. Learning useful rules from inconclusive data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 141–157. AAAI/MIT Press, 1991.

[134] S. M. Weiss and N. Indurkhya. *Predictive Data Mining.* Morgan Kaufmann, 1998.

[135] S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proc. 11th Int. Joint Conf. Artificial Intelligence*, pages 781–787, Detroit, MI, Aug. 1989.

[136] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufman, 1991.

[137] B. Widrow, D. E. Rumelhart, and M. A. Lehr. Neural networks: Applications in industry, business and science. *Comm. ACM*, 37:93–105, 1994.

[138] K. D. Wilson. Chemreg: Using case-based reasoning to support health and safety compliance in the chemical industry. *AI Magazine*, 19:47–57, 1998.

[139] J. York and D. Madigan. Markov chaine monte carlo methods for hierarchical bayesian expert systems. In *Cheesman and Oldford*, pages 445–452, 1994.

[140] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[141] W. Ziarko. The discovery, analysis, and representation of data dependencies in databases. In G. Piatetsky-Shapiro W. J. Frawley, editor, *Knowledge Discovery in Databases*, pages 195–209. AAAI Press, 1991.