CHAPTER 4

Classification Models

This chapter gives an overall introduction of various classification models used in the data-mining process, their comparisons, and research challenges.

4.1 INTRODUCTION TO CLASSIFICATION MODELS

Data mining focuses on automatically searching large stores of data to discover patterns and trends that go beyond simple analysis as pointed out in Chapter 1. Classification is a data-mining procedure that assigns things in a collection to categories or classes. The objective of classification is to accurately predict the target class for every case within the data. For example, a classification model may be accustomed to identify loan candidates as low, medium, or high credit risks. In the coming sections, we will discuss some of the important classification models utilized in this discipline.

4.2 DECISION TREE

A promising approach to ease the data-acquisition bottleneck is to use some learning mechanism to extract the specified knowledge automatically from actual cases or examples that are antecedently handled by the domain consultants. This machine-learning approach enjoys many advantages:

1. In issues where data is expert-dependent, one will merely learn from examples handled by totally different experts, with the hope that this can average the variations among totally different experts.

2. The ability to construct data automatically makes the upgrading task easier. As a result, one will rerun the learning system as a lot of examples accumulate. Some learning ways are always "incremental" in nature.
3. Machine learning is often applied to issues where no experts exist. This is often the case in data processing and knowledge discovery in databases, for which machine-learning techniques are utilized to automatically find new data.

Considerable effort has been made by machine-learning research scientists to the task of gaining "classification knowledge," for which, from among a pre-declared set of accessible categories, the target is to settle on the foremost applicable category for a given case. The goal in such analysis is to develop ways that induce the specified classification data from a given set of pre-classified examples. Vital progress has been made within the last decade toward this goal, and varied ways for automatically inducing classifiers from knowledge area unit are currently being offered. Specifically, constructing classifiers in the form of decision trees has been popular, and varieties of proven real-world applications that use decision tree construction ways are reported.

Ease of production rule generation and human expert comprehensibility makes decision trees most appropriate for knowledge-based systems. Moreover, decision trees have the capability to handle a condition as well as generate solution for the same stating the reasons behind the choice of the case [1, 2]. These features are crucial in typical application domains where human experts use tools to aid in assisting their task by remaining in the driver's seat. An added merit of using decision trees is the ease and potency of their construction compared to alternative classifiers such as neural networks.

Decision trees follow divide-and-conquer approach for solving a problem. A decision tree is a tree structured classifier where each node represents either a leaf node, which stores the value of the target attribute of application, or a decision node, which specifies some test to be carried out on a single attribute value. It can have many children, where each child contains one branch and subtree for each possible outcome of the test.

Each internal node represents a decision or a test on an attribute. Each branch corresponds to attribute value. Each leaf node assigns a classification. A decision tree is built top-down from a root node and involves partitioning the data into subsets. Target attributes will be discrete valued. The logic used to build the tree is generated as rules from decision tree.

4.2.1 Examples of Constructing Decision Trees

A decision tree can act as a "classifier" for selecting the best path from a set of possible action paths. Consider, for example, the task of targeting good candidates to be sent an invitation to apply for a credit card. Given some details about an individual, our aim is to determine whether or not he or she can be a candidate worth inviting. In this example, the details of individual are given as a vector

of attributes that may include sex (male/female), age, status (student/employee/unemployed), college grade point average (GPA), annual income, social security number, etc. Outcomes are viewed as classes, which are in this case "to offer" or "not to offer" an invitation. A decision tree that performs this task is sketched in Fig. 4.1. As the figure shows, each internal node in the tree is labelled with a "test" defined in terms of the attributes and has a branch for each possible outcome for that test, and each leaf in the tree is labelled with a class.

Attributes used for specifying conditions can be of two types—nominal or continuous. Nominal attributes take one value out of a pre-specified set of values. "Sex" and "status" are nominal attributes whereas "age" and "GPA" are continuous ones in the given example. In general, a test condition performed on a nominal attribute yields a single output for each value of the attribute whereas a test condition performed on a continuous attribute is based on a fixed threshold and, thus, yields two possible results, one for each interval as specified by this threshold. The decision tree in Fig. 4.1 illustrates these tests.

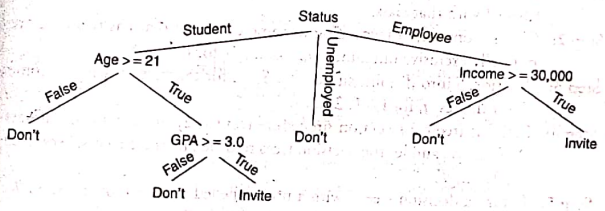

Fig. 4.1 Sample decision tree

To find the appropriate class for a given condition, we start with the test at the root ("Status" in our example) of the tree and keep following the branches as determined by the values of the attributes of the condition currently processed, until a leaf node is reached. For example, suppose the attribute values for a given condition are as follows:

Name = Andrew; Social Security No = 199199; Age = 22; Sex = Male;

Status = Student; Annual Income = 2000; College GPA = 3.39

To classify this case, we start at the root of the tree of Fig. 4.1, which is labelled "Status", and follow the branch labelled "Student" from there. Then at the test node "Age ≥ 21," we follow the true branch, and at the test node "GPA ≥ 3.0," we again follow the true branch. This finally leads to a leaf labelled "invite", indicating that this person is to be invited according to this decision tree.

Decision tree learning is the task of constructing a decision tree classifier, such as the one in Fig. 4.1, from a collection of historical cases. These are individuals who are already marked by experts as being good candidates or not. Each historical case is called a training example and the collection of such examples from which a decision tree is to be constructed is called a training sample. A training example is assumed to be represented as a pair (X, C), where X is a vector of attribute values describing some case and C is the appropriate class for that case. The following subsections describe how a decision tree can be constructed from such a collection of training examples.

4.2.2 A Basic Tree-Construction Procedure

Let $S = [(X_1, C_1), (X_2, C_2), ..., (X_k, C_k)]$ be a training sample. The construction of a decision tree from S can be done in a divide-and-conquer fashion as follows:

Step 1: If all the examples in S are labelled with the same class, return a leaf labelled with that class.

Step 2: Choose some test t (according to some criterion) that has two or more mutually exclusive outcomes $O_1, O_2, ..., O_r$ for the set S.

Step 3: Partition S into disjoint subsets $S_1, S_2, ...$ for examples having outcome O_i for the test t, for $i = 1, 2, ..., r$.

Step 4: Call this tree-construction procedure recursively on each of the subsets $S_1, S_2, ..., S_r$ and let the decision trees returned by these recursive calls be $T_1, T_2, ..., T_r$.

Step 5: Return a decision tree T with a node labelled t as the root and trees $T_1, T_2, ..., T_r$ as subtrees below that node.

For illustration, let us apply this procedure on the set of examples of Table 4.1. We will use Case IDs 1–15 (listed in the first column) to refer to each of these

Table 4.1 Examples for the credit card task

Case ID	Name	Social Security No.	Age	Sex	Status	Income	GPA	Class
1	John	123321	22	Male	Student	3,000	3.22	Invite
2	Mary	343422	38	Female	Employee	32,000	2.00	Invite
3	Ali	876345	46	Male	Employee	69,000	2.90	Invite
4	Lee	673245	23	Male	Student	3,500	3.1	Invite
5	Ted	451087	45	Male	Unemployed	5,000	3.1	Don't
6	Nick	239847	19	Male	Student	1,300	3.8	Don't

(*continued*)

Table 4.1 Continued

Case ID	Name	Social Security No.	Age	Sex	Status	Income	GPA	Class
7	Liz	229951	23	Female	Student	12,000	2.8	Don't
8	Debby	234819	33	Female	Unemployed	5,000	0.00	Don't
9	Pat	258199	32	Male	Student	1,000	2.1	Don't
10	Peter	813672	20	Male	Student	32,000	3.9	Don't
11	Dona	501184	23	Female	Student	6,600	3.3	Don't
12	Jim	619458	40	Male	Unemployed	35,000	3.3	Don't
13	Kim	654397	31	Female	Unemployed	14,000	3.0	Don't
14	Pan	350932	59	Male	Employee	29,000	2.8	Don't
15	Mike	357922	33	Male	Employee	19,000	2.6	Don't

examples. Now, $S = 1, 2, 3, ..., 15$ have a mixture of classes; hence, we jump back to step 2.

Suppose we use the attribute "Status" for our test. This test has three outcomes, "Student," "Unemployed," and "Employee." It partitions S into subsets $S_1 = 1, 4, 6, 7, 9, 10, 11$; $S_2 = 5, 8, 12, 13$; and $S_3 = 2, 3, 14, 15$, respectively, for these outcomes.

It is noticed that S_1 has a mixture of classes. Therefore, the selection of the test case "Age ≥ 21?" partitions S_1 into $S_{11} = 6, 10$ for false outcomes and into $S_{12} = 1, 4, 7, 9, 11$ for the true outcome.

Now, $S_{11} = 6, 10$ has just one class, i.e., "don't." So a leaf labelled with this class is returned for the call on S_{11}.

For set S_{12}, which has a mixture of classes, if we choose GPA ≥ 3.0, then the set will be partitioned into $S_{121} = 7, 9$ and $S_{122} = 1, 4, 1, 1$.

Calls on sets S_{121} and S_{122} will return leaves labelled "don't" and "invite," respectively, and thus, the call on set S_{12} will return the subtree of Fig. 4.2(a).

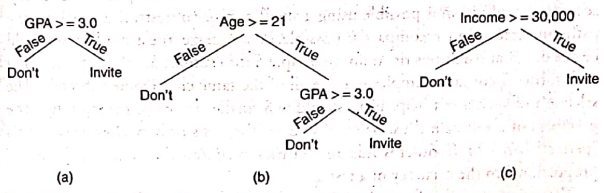

Fig. 4.2 Decision tree stages

Now that we are done with the recursive calls on S_{11} and S_{12}, the call on set S_1 will return the subtree of Fig. 4.2(b).

Call on set S_2 will return a leaf labelled "don't."

For S_3, which contains a mixture of classes, suppose we choose the test "Income ≥ 30,000?" This will partition S_3 into $S_{31} = 1, 4, 1, 5$ for the false outcome and into $S_{32} = 2, 3$ for the true outcome.

Recursive calls on S_{31} and S_{32} will return leaves labelled "don't" and "invite," respectively. Thus, the call on S_3 will return the subtree of Fig. 4.2(c).

Finally, the call on the entire training sample S will return the tree of Fig. 4.1.

Obviously, the accuracy of the tree constructed by this top-down procedure depends solely on how tests are chosen in step 2.

Also, the termination condition of step 1 may not always be the strategy to stop recursion and reduce the growth of the tree.

4.2.3 Selecting the Best Splitting Criterion

The procedure described in the previous subsection will result in a decision tree regardless of the condition-choosing criterion adopted in step 2. That is, for any training sample X subset of S, the decision tree yields C as the path for X. Here the tree-building process is not focused on training examples only but the goal is to build (among many possible consistent trees) a tree that reveals the underlying structure of the domain, so that it can be used to "predict" the class of new examples not included in the training sample and can also be used by human experts to gain useful insight about the application domain. Therefore, some careful criterion ought to be used for test selection in step 2 so that prominent tests (such as "Income" and "Status" in our credit card example) are preferred and irrelevant ones (such as "Name" and "Sex") are neglected. Ideally, one would like the ultimate tree to be as compact as possible because this is an indication that focus has been given to the most relevant tests.

As a matter of fact, since compact decision tree building is a subset of intractable problem, we need to rely on some heuristics to find a "reasonably small" or optimal tree. The core idea is to measure the importance of a test by estimating how much influence it has on the classification of the examples. In this way, accurate classification is possible using a small number of tests, thus, making the paths and tree as much compact as possible. At any stage, the accurate test would be the one that partitions the training sample S into subsets $S_1, S_2, \ldots, S_r$ such that each subset S_i contains samples that are all of the same class ("pure subsets"). The selection of such a test helps us to shut down further recursive partitioning. The goodness of a test can, thus, be estimated on the basis of how close it is to this "perfect" behavior. In other words, the purities of subsets $S_1, S_2, \ldots, S_r$ are directly proportional to the accuracy of a test T.

The principles of information theory can give an idea about the expected amount of information provided by a test [3, 4]. According to this theory, for a

sample S, the average amount of information needed to find the class of a case in S is estimated by the function

$$\text{info}(S) = -\sum_{i=1}^{k} \frac{|S_i|}{|S|} \times \log_2 \frac{|S_i|}{|S|} \text{ bits},$$

where $S_i \subseteq S$ is the set of examples S of class i and k is the number of classes. For example, when $k = 2$ and S has equal numbers of samples of each class, the above quantity evaluates to 1, indicating that knowing the category of a case in S is worth one bit. On the other hand, if all the samples of S are of the same class, then the test condition evaluates to 0, because knowing the category of a case in S provides little information. Typically, the entropy of the set S is at its peak value when all the sample classes are of equal frequency and is equal to 0 when S becomes pure, that is, when all the samples in S are of the same group.

Let T be a test that divides S into $S_1, S_2, \ldots, S_r$. Then the weighted average entropy over these subsets can be obtained using the formula

$$\sum_{i=1}^{r} \frac{|S_i|}{|S|} \times \text{info}(S_i).$$

Test T can be evaluated based on gain factor.

$$\text{gain}(t) = \text{info}(S) - \sum_{i=1}^{r} \frac{|S_i|}{|S|} \times \text{info}(S_i)$$

It measures the reduction in entropy, if test t is applied. This factor, called the information gain of t, is universally used as the basis for test selection during the construction of a decision tree.

$$-\frac{5}{15}\log_2\frac{5}{15} - \frac{10}{15}\log_2\frac{10}{15} = 0.918 \text{ bits.}$$

For illustration, let us compute the gain of the test on the nominal attribute "Status" in the set of training examples of Table 4.1. In the example, there exist five samples of case "invite" and ten samples of case "don't." Hence, the entropy of the set can be computed as

The test on "Status" partitions the set into three subsets: $S_1 = 1, 4, 6, 7, 9, 10, 11$; $S_2 = 5, 8, 12, 13$; and $S_3 = 2, 3, 14, 15$. In S_1, there are three examples of case "invite" and four examples of case "don't." Therefore, the entropy of S_1 is computed as

$$-\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.985 \text{ bits.}$$

All the examples of S_2 are of one class "don't." Thus, the entropy of this set is 0. Finally, in S_3, there are two examples of each of the classes "invite" and "don't." Therefore, the entropy of S_3 is

$$\frac{7}{15} \times 0.985 + \frac{4}{15} \times 0 + \frac{4}{15} \times 1 = 0.726 \text{ bits},$$

Thus, the weighted average entropy after applying the test "Status" becomes 0.726 and the gain of this test is

$$0.918 - 0.726 = 0.192 \text{ bits}$$

To choose the best informative test, these steps are repeated for all the available test conditions and the test with the highest information gain is chosen. Although the information gain test selection criterion has been experimentally shown to lead to compact decision trees in many cases, it was found to be biased in favor of tests that induce finer partitions. As an extreme example, consider the (meaningless) tests defined on attributes "Name" and "Social Security Number" in our credit card application. These tests divide the training sample into a large number of subsets, each containing just one example. Since these sample subsets do not contain a mixture of examples, their entropy will be zero. Hence, the information gain of using these trivial tests becomes maximal.

A bias exists in the gain criterion. It can be overcome by dividing the information gain of a test by the entropy of the test results, which calculates the extent of partitioning done by the test.

$$\text{split}(t) = -\sum_{i=1}^{r} \frac{|S_i|}{|S|} \log \frac{|S_i|}{|S|},$$

giving the *gain-ratio* measure

$$\text{gain-ratio}(t) = \frac{\text{gain}(t)}{\text{split}(t)}.$$

Note that split(t) increases as tests partition the examples into large number of small subsets. In the example of credit card application, tests on the "Name" and "Social Security Number" attributes have high gain. Therefore, dividing by split(t) in the above manner inflicts high penalty on their scores, not allowing them to be selected. Applying the basic tree-construction procedure using the gain-ratio to the set of examples of Table 4.1, we get the decision tree given in Fig. 4.3.

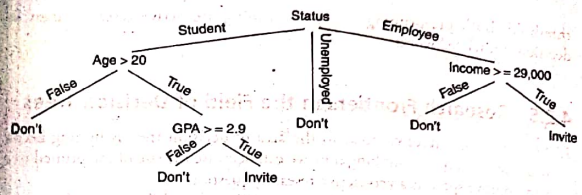

Fig. 4.3 Decision tree construction

4.2.4 Overfitting and Its Reasons

For constructing decision trees we should have training data. This is necessary to capture some underlying functions or trends in that data, usually to be used in prediction. We look for general trends, as we are not focused in identifying the exact nuances and irregularities of the training data. These normally occur due to errors or peculiarities that we are not likely to come across again. The decision tree model thus obtained can be used to predict or generalize over future instances that we may obtain. Overfitting happens when our decision tree considers too much information or noise in our training data.

Our aim is to construct a generalized decision tree; but unfortunately if we build a decision tree until all the training data has been classified correctly and all leaf nodes have been reached, then there is a chance that it will not be a generalized one and that we will have plenty of incorrect classifications once we try to use it. Overfitting can be rectified by methods such as pruning.

Pruning

Greedy heuristics such as "entropy reduction" followed in the construction of decision tree may lead to overfitting the training data and poor accuracy in future predictions. To rectify this problem, modern decision tree algorithms adopt a pruning strategy of some sort. For example, post-pruning or backward pruning involves growing the tree from a data set until all possible leaf nodes have been traversed and then removing particular subtrees. Studies have shown that post-pruning will result in compact and high accurate trees by up to 25%. Even though there exist several pruning techniques, there is no variation in terms of performance. In pre-pruning (another pruning technique), before traversing each path of the tree, a comparison is made to check whether there is a considerable advantage in exploring that path. Comparison is made based on a predefined

threshold. Both pre-pruning and post-pruning are active research areas in decision tree related works.

4.2.5 Research Frontiers in the Field of Decision Trees

1. One of the hot research areas in the area of decision trees is finding better pruning techniques. Thinking how we can decrease the size of the pruned tree and improve speed is a very active research question.
2. Ensembles of decision trees such as cascaded trees and forests are also under research to see whether these improve the accuracy of classical decision tree concepts.
3. Constructing cost-sensitive decision trees is also an active research area. In certain cases, the cost of testing an attribute is high. For example, consider classifying a patient as prone to cancer. For that suppose there is a tree node that conducts a test "A" to find if it is positive or negative. But suppose it costs $1000 to conduct that test in a laboratory. Similarly, many tests have to be conducted on many other nodes for deciding which branch to choose. Hence, the testing cost should be minimum. Especially in decision trees on patient data, the overall testing cost has to be minimum. Such trees are called cost-sensitive decision trees.

4.3 NEURAL NETWORKS

Neural networks provide a mathematical model that makes an attempt to mimic the human brain [5, 6]. They represent knowledge as a layered set of interconnected processors, named neurons. Each node possesses weighted association to other nodes in adjacent layers. Individual nodes accept input from connected nodes and apply the weights together to a function to compute output values. Learning in neural networks is made possible by analyzing the weight changes while a set of input instances is repeatedly passed through the network. Once training is complete, an unknown instance passing through the network is classified based on the values seen at the output layer. Many studies have analyzed the existing work on neural network construction, important issues, progresses made, and the current state of the art [7, 8]. Typically, a neural network model has the configuration shown in Fig. 4.4. Neurons fire when input is greater than some threshold. But it should also be noted that firing does not get larger as the stimulus increases; it is an all-or-nothing arrangement [9].

Consider a firing rate at every neuron. Also assume that a neuron connects with m other neurons and hence receives m inputs $x_1, x_2, \ldots, x_m$. This configuration is actually called a "perceptron." The perceptron model was one of the earliest neural network models. It was proposed by Rosenblatt in 1962. A perceptron model considers the weighted sum of inputs and fires the output 1 if the sum is greater than some adjustable threshold value; otherwise, it sends 0. Thus, it

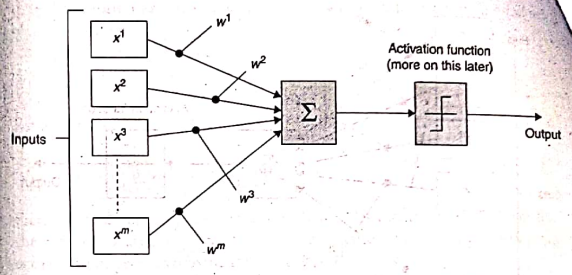

Fig. 4.4 A neural network configuration

models a neuron structure. This is the all-or-nothing firing described in the previous paragraph. The function based on which the output is determined is called the "activation function."

The inputs $(x_1, x_2, \ldots, x_m)$ and connection weights $(w_1, w_2, \ldots, w_m)$ in Fig. 4.4 are generally real values, both positive (+) and negative (–). If the feature of some x_i fires the perceptron, the weight w_i will be positive; else it will be negative. The perceptron as a whole includes weights, the summation processor, an activation function, and an adjustable threshold processor (called bias). For convenience, bias is treated as just another input. Figure 4.5 illustrates the revised configuration with bias.

The bias can be considered the propensity (a tendency toward a particular way of behaving) of the perceptron to fire without taking into consideration the inputs. The perceptron configuration network shown in Fig. 4.5 fires if the weighted sum is greater than 0. In mathematical terms, it can be represented as

$$\text{Weighted sum} = \sum_{i=1}^{m} \text{bias} + (w^i x^i)$$

Activation function

The activation functions can be of the following forms:

Sigmoid function: The stronger the input, the faster the neuron fires. The sigmoid is also very helpful in multilayer networks, as the sigmoid curve allows for differentiation (which is needed in Back Propagation training of multilayer networks). Mathematically stating,

$$F(x) = \frac{1}{(1 + e^{-x})}$$

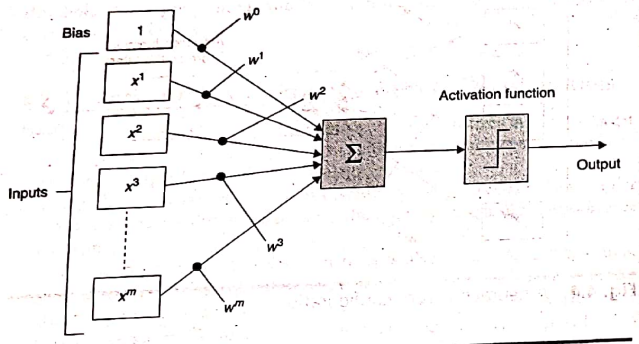

Fig. 4.5 Artificial neuron configurations, with bias as additional input

Step function: A step function is a basic on/off type of function; if $x < 0$, then 0, else if $x >= 0$, then 1. Thus, based on the type of input, output, and problem domain, suited functions can be adopted at respective layers.

Learning can be of two types: supervised and unsupervised. For supervised learning, consider a real-world example of a child learning to recognize a chair. He/she is taught by placing chairs with several other objects. After this training, the child will be able to correctly classify a new object as chair or not. This exactly illustrates the idea behind perceptron. For unsupervised learning, consider the case of a six-month old baby recognizing his mother. Here there is no supervisor. All classification algorithms belong to the class of supervised learning, and clustering algorithms belong to unsupervised learning.

Perceptron learning

A perceptron is a single-layer neural network whose weights and biases are trained to generate a correct target vector when fed with the corresponding input vector. The learning technique followed is called the perceptron learning rule. Due to its ability to generalize the training vectors and work with randomly distributed connections, perceptron is widely accepted by researchers. A perceptron is particularly fitted for simple problems in pattern classification. If the data could be separated into two groups using a hyperplane, then the data is said to be linearly separable [10]. For linearly separable data, the perceptron *learning rule* that can be applied is as follows:

The perceptron is trained to accept each input vector with a corresponding target output of either 0 or 1. The learning rule converges to a solution in finite

time if a solution exists. The learning rule can be summarized by the following two equations:

$$b = b + [T - A]$$

For all inputs i,

$$W(i) = W(i) + [T - A] * P(i)$$

Here W corresponds to the vector of weights, P is the input vector presented to the network, T is the expected result that the neuron should have shown, A is the actual output generated by neuron, and b is the bias.

Training

Vectors from a training set are fed to the network one after another. If the network generates correct output, any further action is not needed. Otherwise, the weights and biases are adjusted using the perceptron learning rule (as shown before). Training is completed when an epoch (an entire pass through all the input training vectors is called an epoch) of the training set occurs without any error.

After the training phase, for any further inputs, the perceptron will respond with the correct output vector. If a vector **P**, which was not present in the training set, is fed to the network, the network will try to exhibit generalization by generating an output similar to target vectors for input vectors close to the previously unseen input vector **P**. The transfer function used in the hidden layer is log-sigmoid while that in the output layer is pure linear.

Neural networks are very good in classification and regression tasks where the attributes have missing values and also when the attribute values are categorical in nature [9, 10]. The accuracy observed is very good, but the only bottleneck is the extra training time and complexity in the learning process when the number of training set examples seems very high. Krieger and Xu [6, 7] describe how neural networks can be applied in data mining. There are some algorithms for extracting comprehensible representations from neural networks. Browne *et al.* [5] describe research to generalize and extend the capabilities of these algorithms. The application of data-mining technology based on neural network is vast. One such area of application is in the design of mechanical structure. Wang and Sui [8] introduce one such application of data mining based on neural network to analyze the effects of structural technological parameters on stress in the weld region of the shield engine rotor in a submarine.

There are some studies that explain the application of neural networks in the study of proteins [11]. In that work, global adaptive techniques from multiple alignments are used for prediction of beta-turns. This also introduces global adap-

tive techniques such as Conjugate Gradient Method and Preconditioned Conjugate Gradient Method. An approach to discover symbolic classification rules using neural networks is discussed in some related works [10].

Here, the network is first trained to achieve the required accuracy rate, and then the activation values of the hidden units in the network are analyzed. Classification rules are generated using the result of this analysis.

Back propagation in multilayer perceptrons

Among the several neural network architectures for supervised classification, feed-forward multilayer network trained with back propagation algorithm is the most popular. Neural networks in which the signal flows from the input to the output (forward direction) are called feed-forward neural networks. A single-layer neural network can solve linearly separable problems only. But when the problem to be solved is more complex, a multilayer feed-forward neural network can be used. Here there can be hidden layers other than the input and output layers. There is a layer of weights between two adjacent levels of units (input, hidden, or output). Back propagation is the training method most often used with feed-forward multilayer networks, typically when the problem to be solved is a nonlinear classification problem. In this algorithm, the error at the output layer is propagated back to adjust the weights of network connections. It starts by making weight modifications at the output layer and then moving backward through the hidden layers. In many works, a multilayer feed-forward network is proposed to select the input attributes that are most useful for discriminating classes in a given set of input patterns [9]. This is particularly helpful in feature selection. There are some practical applications of neural networks in patient data analysis [12, 13] and in protein sequence classification [14].

A simple example of a back propagation network with calculations is as follows (see Fig. 4.6).

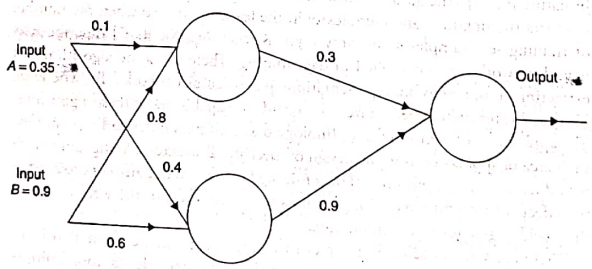

Fig. 4.6 Example of a back propagation network

Assume that the neurons have a sigmoid activation function.

1. Perform a forward pass on the network.
2. Perform a reverse pass (training) once (target = 0.5).
3. Perform a further forward pass and comment on the result.

Answer:

1. Input to top neuron = $(0.35 \times 0.1) + (0.9 \times 0.8) = 0.755$. Out = 0.68.
   Input to bottom neuron = $(0.9 \times 0.6) + (0.35 \times 0.4) = 0.68$. Out = 0.6637.
   Input to final neuron = $(0.3 \times 0.68) + (0.9 \times 0.6637) = 0.80133$. Out = 0.69.
2. Output error $\delta = (t - o)(1 - o)o = (0.5 - 0.69)(1 - 0.69)0.69 = -0.0406$.
   New weights for output layer

$$w_{1^+} = w_1 + (\delta \times \text{input}) = 0.3 + (-0.0406 \times 0.68) = 0.272392.$$

$$w_{2^+} = -w_2 + (\delta \times \text{input}) = 0.9 + (-0.0406 \times 0.6637) = 0.87305.$$

Errors for hidden layers:

$$\delta_1 = \delta \times w_1 = -0.0406 \times 0.272392 \times (1 - o)o = -2.406 \times 10^{-3}$$

$$\delta_2 = \delta \times w_2 = -0.0406 \times 0.87305 \times (1 - o)o = -7.916 \times 10^{-3}$$

New hidden layer weights:

$$w_{3^+} = 0.1 + (-2.406 \times 10^{-3} \times 0.35) = 0.09916.$$

$$w_{4^+} = 0.8 + (-2.406 \times 10^{-3} \times 0.9) = 0.7978.$$

$$w_{5^+} = 0.4 + (-7.916 \times 10^{-3} \times 0.35) = 0.3972.$$

$$w_{6^+} = 0.6 + (-7.916 \times 10^{-3} \times 0.9) = 0.5928.$$

3. Old error was −0.19. New error is −0.18205. Therefore, error has reduced.

4.3.1 Research Frontiers in Neural Networks

1. Improving the speed of the learning process of a network using faster converging functions is one of the main goals of researchers in the neural network domain. Functions such as gradient descent and its new variants are always emerging in this domain.
2. The problem of local minima is to be addressed, and finding an optimal structure and other parameters of a neural network, such as learning rate for a particular problem, is an active research area.

3. Solution-oriented hybrid systems that use other models along with a neural network to solve a particular problem are also very active among researchers.
4. Using genetic or evolutionary algorithms to train neural network and then trying to optimize the network is found in many papers in the literature. They can be used in pattern recognition and related fields.