Chapter 19
Machine Learning

Abstract Traditionally there have been two paradigms of statistical analysis – classical and Bayesian. Machine learning is essentially a third paradigm, based on algorithms that rely heavily on the speed of modern computing to derive "decision rules" that predict customer behavior. We discuss several machine learning techniques, including covering algorithms, instance-based learning, genetic algorithms, Bayesian networks, support vector machines, and committee machine methods such as bagging and boosting.

19.1 Introduction

Machine learning encompasses a repertoire of data mining techniques that have mainly been developed in the field of computer science. Hence, its goal is no different from other data mining techniques: discover interesting patterns or information from data.

Historically, machine learning researchers have focused more on comprehensible patterns than on prediction *per se*. They have emphasized on understanding the structure of the data even though their techniques can be used for prediction and classification. They focused on explicitly representing knowledge so that decision makers know why the models work. In addition, researchers in machine learning have been more interested in the exploratory aspect of data analysis. Statistics has been more concerned with testing hypotheses whereas machine learning has been more concerned with formulating interesting hypotheses (Witten and Frank 2000). However, many techniques in machine learning have been significantly influenced by statistical concepts. Sometimes it is difficult to say whether a specific method is a machine learning technique or a statistical method.

Researchers in machine learning have developed a number of data analysis tools – they prefer the term "generalizations" – that can be used in prediction, classification, clustering, and uncovering associations. This chapter will focus on machine learning tools that are not covered in other chapters.

For example, constructing decision trees is one of the most important tools in machine learning, but it is extensively discussed in Chapter 17. Similarly, market basket analysis, a machine learning tool that discovers association rules, is covered in Chapter 13.

We first discuss three approaches to machine learning that may be unfamiliar to database marketing researchers: the 1-rule, rule-induction by covering algorithms, and instance-based learning. The 1-rule is a simple rule-induction algorithm that is often used as a baseline model for other machine learning algorithms. The 1-rule also provides insight for understanding other rule-induction algorithms. In the next section, we study two well-known covering algorithms for rule induction, PRISM and INDUCT. The PRISM algorithm constructs a set of "perfect" rules, assuming no "noise" in the data. The INDUCT algorithm overcomes this problem by introducing a probabilistic concept. In instance-based learning we describe the nearest-neighbor and k-nearest-neighbor methods, along with their extensions. We then discuss more recent machine learning techniques including genetic algorithms, Bayesian networks, support vector machines, and committee machines.

19.2 1-Rule

The 1-rule or 1R may be the simplest algorithm to generate a set of classification rules from training examples.[1] It is easy to understand and cheap to implement. The algorithm is conceptually similar to univariate profiling frequently used by database marketers. Each attribute (or independent variable) is considered one at a time, and a corresponding set of rules (i.e., relationships between the selected attribute and the dependent variable) is generated. The attribute with the smallest classification error is chosen and the corresponding set of rules becomes the final set of rules for the 1R.

First introduced by Holte (1993), the 1-rule is often used as the baseline model to compare with a more sophisticated model. Because of its simplicity, the 1-rule has a critical problem in that the joint effect of multiple attributes on the response cannot be modeled because each attribute is considered one at a time. However, its classification performance in validation samples has been comparable to highly complex machine learning algorithms in various applications (Holte 1993).

To see how the 1-rule algorithm works, consider the promotional response data in Table 19.1. It is a typical classification task in which we would like to find target customers for a promotional offer. The data consist of a dependent variable indicating the response status to the promotional offer and

[1] Researchers in machine learning prefer the terms training example/instance (or training sample) to estimation sample in traditional statistics. Estimation samples are used to determine the values of parameters in statistical models. Similarly, we derive a set of rules (or patterns) from training instances or examples.

Table 19.1 Promotional response data

Response	Sex	Age	Contact channel
Yes	Male	Teen	E-mail
Yes	Male	Teen	DM
No	Male	Teen	TM
Yes	Male	20s	E-mail
No	Male	20s	DM
No	Male	20s	TM
No	Male	20s	E-mail
No	Male	30 plus	DM
Yes	Male	30 plus	TM
No	Male	30 plus	E-mail
Yes	Female	Teen	DM
No	Female	Teen	TM
Yes	Female	Teen	E-mail
Yes	Female	Teen	DM
No	Female	20s	TM
Yes	Female	20s	E-mail
Yes	Female	20s	DM
No	Female	30 plus	TM
Yes	Female	30 plus	E-mail
No	Female	30 plus	DM

three independent variables characterizing each respondent. For expositional purposes, all three independent variables are categorical in nature. We will extend the algorithm to numerical independent variables later.

We take each attribute in turn and develop a set of rules. For example, there are ten observations for males and ten for females. Among ten males, four respond and six do not. Hence for males, the most accurate rule we could generate would be, 'if male then do not respond' or 'male → no' in short. This rule accurately forecasts six out of ten cases and leads to the error rate of 40%. Similarly, we make a rule for female, 'female → no' that will lead to the same error rate. Therefore, if we apply the rules made by the attribute sex, we would erroneously forecast 8 out of 20 observations.

Applying the same procedure into the other two attributes, we summarize the result in Table 19.2. The number of errors for each rule is given, along with the total number of errors for the set of rules for each attribute. We

Table 19.2 $1R$ applied to promotional response data

Attributes	Rules	Errors	Total errors
Sex	Male → No	4/10	8/20
	Female → Yes	4/10	
Age	Teen → Yes	2/7	7/20
	20s → No	3/7	
	30 plus → No	2/6	
Contact channel	E-mail → Yes	2/7	6/20
	DM → Yes	3/7	
	TM → No	1/6	

now select the attribute that generates the set of rules with the smallest percentage of errors. In this case, that attribute is contact channel, and the set of rules is: If contact channel is through e-mail or DM, then the customer responds. If it is through TM, then she does not respond.

The 1-rule algorithm treats numeric attributes as continuous and uses a straightforward method to divide the range of values into several disjoint intervals. The method is similar to the partitioning technique employed in decision trees (see Chapter 17). Suppose that ages in Table 19.1 are provided in numeric values instead of three categories. We rearrange observations in ascending order according to age.

16	17	17	18	18	19	19	21	23	24	24	25	27	29	32	34	38	40	41	44
Y	Y	Y	Y	Y	N	N	Y	N	N	N	N	Y	Y	Y	N	N	N	N	Y

To partition the sequence wherever the response class changes, we need 6 breakpoints. That is, with 6 breakpoints of 18.5, 20, 22, 26, 33 and 42.5, we have a zero error rate on the training example. The corresponding rule is 'age $< 18.5 \rightarrow$ Yes,' '$18.5 \leq$ age $< 20 \rightarrow$ No,' '$20 \leq$ age $< 22 \rightarrow$ Yes,' '$22 \leq$ age $< 26 \rightarrow$ No,' '$26 \leq$ age $< 33 \rightarrow$ Yes,' '$33 \leq$ age $< 42.5 \rightarrow$ No,' and '$42.5 \leq$ age $\rightarrow$ Yes.'

There is a risk of over-fitting in generating rules for continuously valued attributes. As we can see in the age example, the $1R$ method will naturally create a large number of breakpoints and many of them may simply be due to chance. To avoid this problem, the $1R$ method requires each partition (except the rightmost) to contain a minimum number of examples for the majority response. Suppose that the minimum is set to be three. Then the second, the third, and the fourth partition in the above age example should be merged, and the result would be:

Age	16	17	17	18	18	19	19	21	23	24	24	25	27	29	32	34	38	40	41	44
Actual	Y	Y	Y	Y	Y	N	N	Y	N	N	N	N	Y	Y	Y	N	N	N	N	Y
Prediction	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	Y	Y	Y	N	N	N	N	Y

The corresponding set of rules is 'age $< 18.5 \rightarrow$ Yes,' '$18.5 \leq$ age $< 26 \rightarrow$ No,' '$26 \leq$ age $< 33 \rightarrow$ Yes,' '$33 \leq$ age $< 42.5 \rightarrow$ No,' and 'age $\leq 42.5 \rightarrow$ Yes.' The error rate on the training sample would be 1 out of 20.

19.3 Rule Induction by Covering Algorithms

Rules are one of the most popular ways of representing knowledge that machine learning techniques use. There are hundreds of heuristics available to construct rules from a set of training examples. This section studies a class of rule-generating algorithms called covering algorithms that have been proven to be effective in a number of applications (Witten and Frank 2000)

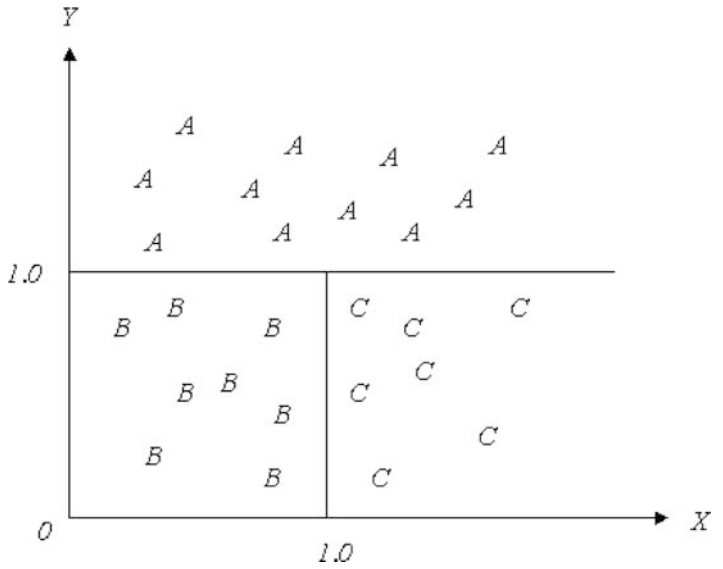

Fig. 19.1 The concept of a covering algorithm.

19.3.1 Covering Algorithms and Decision Trees

Trees and rules have some common features in representing knowledge. A set of rules directly can be constructed from a decision tree (even though the reverse is not always true). However, an important distinction is that decision trees are typically constructed using the principle of "divide and conquer." The decision tree algorithm works top-down, recursively. At each stage the algorithm selects the best attribute to separate examples into the known classes. The classification is refined as the process of branching is repeated.

On the other hand, the covering algorithm considers each class one at a time and attempts to construct a set of rules covering all examples (instances) of each class. That is, we select the rule such that it includes as many instances of a given class as possible and at the same time excludes as many instances of the other classes as possible (Witten and Frank 2000). The covering algorithm refines the selected rule by further restricting the number of instances. The process is repeated until the pre-specified condition for classification accuracy is met.

Figure 19.1 illustrates how the covering algorithm works. There are 26 training instances. Characterized in two attributes or dimensions, each instance is classified into one of three groups: A, B and C. First, let us construct a rule for class A. The horizontal line of $Y = 1$ will divide As from the other two classes. The rule for the class A should be: 'If $Y > 1$ then class $= A$.' Since all instances in the A class are covered by the rule, we construct a set of rules for class B. Restricting the attribute space by $Y \leq 1$

covers all instances in class B and C excluding all observations in class A. Adding $X \leq 1$ to the antecedent $Y \leq 1$, we can refine the rule to cover all instances in class B without including instances in class C. Hence, the rule for the class B should be: 'If $X \leq 1$ and $Y \leq 1$ then class $= B$.' Similarly, the rule for the class C is 'If $X > 1$ and $Y \leq 1$ then class $= C$.'

We could have applied a decision tree algorithm to the same example in Fig. 19.1. The tree will have the first split at $Y = 1$ and the second split at $X = 1$. Its result is similar to the covering algorithm. However, decision trees consider all classes in constructing trees while covering algorithms focus on one class (of response) at a time. Because of this difference in constructing rules, there are situations when the covering algorithm is much more effective in representing knowledge than decision trees.

Witten and Frank (2000) provide such case in which there are four attributes, w, x, y, and z and each attribute can take the value of 1, 2, or 3. Suppose that responses are 'Yes' if $x = 1$ and $y = 1$ (whatever the value of w and z), or if $w = 1$ and $z = 1$ (whatever the value of x and y). Otherwise, responses are 'No.' Rules derived from the covering algorithm are simple: 'If $x = 1$ and $y = 1$ then response = Yes,' 'if $w = 1$ and $z = 1$ then response = Yes,' and otherwise response = No. On the other hand, Fig. 19.2 shows the derived decision tree applied to the same training examples.[2] The solution provided by the decision trees is much more complicated than the rules by the covering algorithm. This is called the replicated sub-tree problem. The reason why rules from the covering algorithm are simple is that each rule (e.g., if $x = 1$ and $y = 1$ then response = Yes) represents an independent piece of knowledge. Hence, a new rule (e.g., if $w = 1$ and $z = 1$ then response = Yes) can be added to an existing rule without disturbing the previous set of rules. However, the whole tree needs to be reshaped if we want to add a new rule to an existing tree structure.

19.3.2 PRISM

A simple covering algorithm called PRISM has been proposed by Cendrowska (1987).[3] It is simple to implement and easy to understand. However, the method is known to overfit the training samples because it assumes no noise in the data.

To see how the PRISM algorithm works, we again consider the promotional response data in Table 19.1 introduced in the previous section. The promotional response consists of two classes: 'Yes' or 'No'. Applied to each class in turn, PRISM generates a set of rules for each of the two classes.

---

[2] Note that some sub-trees in the upper right of Fig. 19.2 are not drawn for simplification.

[3] Note: The PRISM algorithm is not to be confused with the PRISM market segmentation scheme derived from cluster analysis and discussed in Chapter 16.

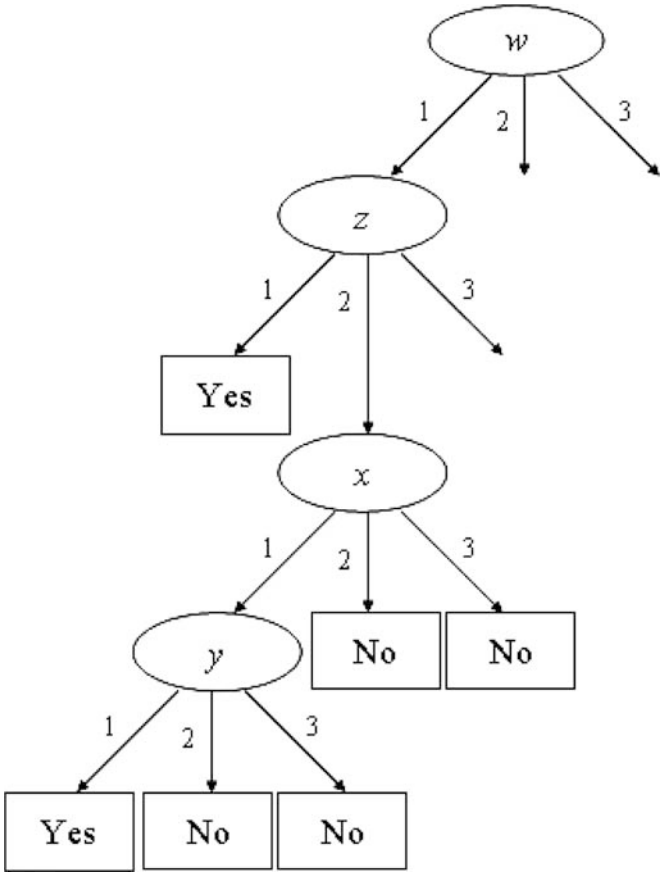

Fig. 19.2 The replicated sub-tree problem of decision trees (Modified from Witten and Frank 2000).

Let us first find a set of rules to identify the 'Yes' class. That is, we are seeking a rule taking the form like 'if X then response = Yes' where X is the characteristic of an attribute. For example, consider a rule 'if sex = male then response = Yes.' There are ten males among the training examples, and four out of the ten respond 'Yes.' Hence, this rule can be said to have a classification accuracy of 40%. We generate potential rules considering all possible Xs in Table 19.1 and evaluate their classification accuracy.

If sex = male then response = Yes	4/10
If sex = female then response = Yes	6/10
If age = teen then response = Yes	5/7
If age = 20s then response = Yes	3/7
If age = 30 plus then response = Yes	2/6
If contact channel = e-mail then response = Yes	5/7
If contact channel = DM then response = Yes	4/7
If contact channel = TM then response = Yes	1/6

Among the above lists, the most accurate rules are when X is 'age = teen' or 'contact channel = e-mail.' Because they are tied with the accuracy of 5/7, we arbitrarily choose 'age = teen' and create a temporary rule:

'If age = teen then response = Yes'

Because this rule is not perfect (i.e., its predictive accuracy is less than 100%), we refine it by adding more terms in the antecedent. That is, we are now seeking a rule of the form 'if age = teen and Y then response = Yes' where Y is the characteristic of an attribute. Note that we restrict our search space into the observations with age = teen. There are seven respondents. We generate the refined rules considering all possible Ys and evaluate the corresponding classification accuracy.

If age = teen and sex = male then response = Yes	2/3
If age = teen and sex = female then response = Yes	3/4
If age = teen and contact channel = e-mail then response = Yes	2/2
If age = teen and contact channel = DM then response = Yes	3/3
If age = teen and contact channel = TM then response = Yes	0/2

The third and the fourth rules above both have perfect predictive accuracy. To select one of them, we apply the coverage heuristic where the rule with the greater coverage should be selected. We select the fourth since the third rule only covers two instances and the fourth covers three. We now have the final rule:

'If age = teen and contact channel = DM then response = Yes'

This is just one rule covering only three 'Yes' instances in the data. There are still seven other 'Yes' instances to be covered. Continuing the PRISM algorithm, we delete the three instances that are already covered. The resulting data are shown in Table 19.3, consisting of 17 observations.

The same process is applied to the reduced data shown in Table 19.3. That is, consider a rule taking the form of 'if X then response = Yes.' Enumerating potential rules for all possible Xs with the corresponding classification accuracy:

If sex = male then response = Yes	3/9
If sex = female then response = Yes	4/8
If age = teen then response = Yes	2/4
If age = 20s then response = Yes	3/7
If age = 30 plus then response = Yes	2/6
If contact channel = e-mail then response = Yes	5/7
If contact channel = DM then response = Yes	1/4
If contact channel = TM then response = Yes	1/6

The most accurate rule is the sixth when X is 'contact channel = e-mail.' Hence, we create a temporary rule:

'If contact channel = e-mail then response = Yes'

Table 19.3 Promotional response data after the 1st iteration of the PRISM algorithm

Response	Sex	Age	Contact channel
Yes	Male	Teen	E-mail
No	Male	Teen	TM
Yes	Male	20s	E-mail
No	Male	20s	DM
No	Male	20s	TM
No	Male	20s	E-mail
No	Male	30 plus	DM
Yes	Male	30 plus	TM
No	Male	30 plus	E-mail
No	Female	Teen	TM
Yes	Female	Teen	E-mail
No	Female	20s	TM
Yes	Female	20s	E-mail
Yes	Female	20s	DM
No	Female	30 plus	TM
Yes	Female	30 plus	E-mail
No	Female	30 plus	DM

Again we refine this rule because it is not perfect. We consider a rule taking the form of 'if contact channel = e-mail and Y then response = Yes' where Y is the characteristic of an attribute. Enumerating potential rules for all possible Ys with the corresponding classification accuracy:

If contact channel = e-mail and sex = male then response = Yes 2/4
If contact channel = e-mail and sex = female then response = Yes 3/3
If contact channel = e-mail and age = teen then response = Yes 2/2
If contact channel = e-mail and age = 20s then response = Yes 2/3
If contact channel = e-mail and age = 30 plus then response = Yes 1/2

We find two perfect rules: the second and the third. Applying the principle of the greatest coverage, we select the second covering three instances and write another final rule:

'If contact channel = e-mail and sex = female then response = Yes'

We now cover 6 'Yes' instances out of 10 in the data. By the same process, we continue applying the PRISM algorithm in an effort to cover all 10 'Yes' instances. Summarizing all the final rules to identify the 'Yes' class:

'If age = teen and contact channel = DM then response = Yes'
'If contact channel = e-mail and sex = female then response = Yes'
'If contact channel = e-mail and age = teen then response = Yes'
'If age = 20s and sex = female and contact channel = DM then response = Yes'
'If sex = male and age = 30 plus and contact channel = TM then response = Yes'

As shown above, the first two rules cover three 'Yes' instances each. The remaining three rules cover one 'Yes' instance each. That is, the above five

rules cover 9 out of 10 'Yes' instances – we could not cover all 10 'Yeses'. It is interesting to look at the instance that could not be covered by the PRISM algorithm. It is the fourth observation in Table 19.1. The trouble is that the seventh observation has the identical attributes with different response. We cannot cover both these observations using the three attributes in Table 19.1.

Once all the final rules are generated for the 'Yes' class, the PRISM algorithm is applied to the 'No' class. The same process is applied and a set of the final rules is constructed for the 'No' class.

19.3.3 A Probability Measure for Rule Evaluation and the INDUCT Algorithm

The accuracy ratio is the critical metric to construct a set of perfect rules in the PRISM rule-generating algorithm. It is defined as the ratio c/n where n is the total number of instances covered by a rule and c is the number of instances classified (or predicted) correctly by the rule. For example, the first rule of the five final rules for class Yes in the previous section, 'If age = teen and contact channel = DM then response = Yes,' covers three instances in Table 19.1 and correctly classifies all three. So the accuracy ratio of this rule is perfect.

However, the PRISM algorithm tends to overfit the training instances because it uses the accuracy ratio as a metric to select the best rules. A method of overcoming this problem is proposed by Gaines and Compton (1995) who introduced the concept of the quality of a rule. Their algorithm, called INDUCT, exploits the idea of statistical significance based on the binomial distribution.

The quality of a proposed rule can be evaluated by calculating the probability of the baseline rule providing equal or better classification accuracy to the proposed rule. Let N be the total number of instances covered by the baseline rule and C be the number of instances correctly classified by this rule. The baseline rule for the given class typically is the rule that can be made without knowing any attribute information. For the promotional response data in Table 19.1, the baseline rule of the response class 'Yes' (or 'No') is to say 'Yes' (or 'No') whatever the characteristics of an instance are. This baseline rule covers 20 instances ($N = 20$) of which 10 instances ($C = 10$) are correctly classified. Hence, the accuracy ratio of a proposed rule should be at least greater than $0.5 (= 10/20)$ if it has any value.

Given the baseline rule with N and C, assume the proposed rule covers n instances with c correct classifications. For example, one of the best rules for Yes class, 'If age = 20s and sex = female and contact channel = DM then recommend = Yes' is a perfect rule with $n = c = 1$. Then what is the probability that the baseline rule would correctly classify c instances

when it is applied to n instances? This probability is known to follow a hypergeometric distribution. The problem is analogous to the well-known experiment of sampling without replacement in probability theory. That is, an urn contains N balls among which C balls are marked 'Yes' and the rest $N - C$ balls are marked 'No.' The experiment is to randomly draw n balls from the urn without replacement. The probability that the sample of n balls contains exactly c balls marked 'Yes' and the rest marked 'No' is given as a hypergeometric distribution.

$$P(c|N, C, n) = \frac{\binom{C}{c}\binom{N-C}{n-c}}{\binom{N}{n}} \tag{19.1}$$

We measure the quality of a proposed rule by the probability of the baseline rule providing equal or better classification accuracy to the proposed rule. Hence, going back to the urn example, it is the probability that the sample of n balls contains more than or equal to c balls marked 'Yes.' The quality of a proposed rule $q(R)$ is given by

$$q(R) = \sum_{i=c}^{n} P(i|N, C, n) \tag{19.2}$$

where $P(i|N, C, n)$ is the density of a hypergeometric distribution in Equation 19.1. Higher values of $q(R)$ mean that the proposed rule is not much better than the baseline rule, since the baseline rule has a high probability of achieving at least the same accuracy. For example, let us evaluate the quality of the proposed rule for class 'Yes': 'If age = 20s and sex = female and contact channel = DM then response = Yes'. It is a perfect rule with $n = c = 1$ and the corresponding baseline rule has $N = 20$ and $C = 10$. Employing Equation 19.2, the quality of the proposed rule $q(R)$ is calculated to be 0.5. That is, a random prediction baseline would have a 50% chance of predicting at least as well as the proposed rule. Now let us evaluate another perfect rule in the list. PRISM suggests 'If contact channel = e-mail and sex = female then recommend = Yes.' It is a first rule in the set of rules for class 'Yes.' It is a perfect rule with $n = c = 3$ and the corresponding baseline rule has $N = 20$ and $C = 10$. Its value of the quality is calculated to be 0.11. Hence, this proposed rule is better than the previously proposed rule because it is less likely the baseline rule could have generated the same or better accuracy as the new proposed rule.

We are ready to describe the covering algorithm of INDUCT (Gaines and Compton 1995). First, the PRISM algorithm is applied to construct the best perfect rules for each class. The perfect rules for class 'Yes' and 'No' are:

'If age = teen and contact channel = DM then response = Yes'
'If contact channel = e-mail and sex = female then response = Yes'
'If contact channel = e-mail and age = teen then response = Yes'
'If age = 20s and sex = female and contact channel = DM then response = Yes'
'If sex = male and age = 30 plus and contact channel = TM then response = Yes'
'If contact channel = TM and sex = female then response = No'
'If contact channel = TM and age = teen then response = No'
'If age = 30 plus and contact channel = DM then response = No'
'If sex = male and age = 20s and contact channel = TM then response = No'
'If sex = male and age = 20s and contact channel = DM then response = No'
'If sex = male and age = 30 plus and contact channel = e-mail then response = No'

Second, we use $q(R)$ to help us decide whether a given rule should be "pruned". Specifically, we compare each of the rules generated from the previous step with the corresponding rule before a further restriction is added. For example, consider one of the perfect rules in the above line, 'if contact channel = TM and sex = female then response = No.' This rule leads to 3 perfect classifications ($n = c = 3$) and the corresponding baseline rule has $N = 20$ and $C = 10$. The value of the rule ($q(R)$) is 0.11. In order to see whether pruning is appropriate, the preceding rule before adding the condition sex = female that is 'if contact channel = TM then response = No is considered.' This preceding rule is not perfect but $n = 6$ and $c = 5$. Its value is calculated to be 0.07, which is lower than the value of 0.11 calculated for the perfect rule. That is, cutting off the last term (sex = female) changes the accuracy ratio from 3/3 to 5/6 and improves the quality of the rule from 0.11 to 0.07. We continue this pruning process by removing the last term until the quality measure no longer improves. We apply this pruning process to each of 11 perfect rules above and summarize the results in Table 19.4.

Third, we select the rule with the smallest $q(R)$ among the set of pruned rules. There are 11 pruned rules in Table 19.4 and the rule 'if contact channel = TM' has the smallest $q(R)$ of 0.07.

Fourth, we delete the instances covered by the rule with the smallest $q(R)$ from the training examples and repeat the process of growing and pruning rules from step 1 to 3 for the remaining instances. We continue this procedure until no training examples remain.

The INDUCT covering algorithm elegantly avoids over-fitting by introducing a probabilistic metric of measuring the quality of a rule. However, the INDUCT does not guarantee the best rule generation partially because the evaluation of $q(R)$ is only made over the temporary rules generated by PRISM (Witten and Frank 2000). The ideal method of finding the best rule

Table 19.4 Pruning evaluation to PRISM rules

Consequent	Antecedent	$q(R)$	Status
Yes	Age = teen & contact channel = DM	0.11	Final
	Age = teen	0.21	
	Contact channel = e-mail & sex = female	0.11	Final
	Contact channel = e-mail	0.21	
	Contact channel = e-mail & age = teen	0.50	Prune
	Contact channel = e-mail	0.21	Final
	Age = 20s & sex = female & contact channel = DM	0.50	Prune
	Age = 20s & sex = female	0.50	Final
	Age = 20s	0.77	
	Sex = male & age = 30 plus & contact channel = TM	0.50	Final
	Sex = male & age = 30 plus	0.89	
No	Contact channel = TM & sex = female	0.11	Prune
	Contact channel = TM	0.07	Final
	Contact channel = TM & age = teen	0.50	Prune
	Contact channel = TM	0.07	Final
	Age = 30 plus & contact channel = DM	0.24	Final
	Age = 30 plus	0.34	
	Sex = male & age = 20s & contact channel = TM	0.50	Prune
	Sex = male & age = 20s	0.29	Final
	Sex = male	0.38	
	Sex = male & age = 20s & contact channel = DM	0.50	Prune
	Sex = male & age = 20s	0.29	Final
	Sex = male	0.38	
	Sex = male & age = 30 plus & contact channel = e-mail	0.50	Prune
	Sex = male & age = 30 plus	0.50	Prune
	Sex = male	0.38	Final

is to evaluate all possible rules exhaustively (e.g., all possible combinations of attributes), but this is too computationally burdensome. Recently a number of researchers have proposed alternative algorithms to search for the best rules without incurring too much computational time (Cohen 1995; Frank and Witten 1998).

19.4 Instance-Based Learning

Representing knowledge in the form of historical examples may be the simplest way of learning. People intentionally or unintentionally memorize a set of training instances. When a new instance comes up, memorized instances are searched to find the most similar one to the new instance. For example, when we first look at a person, we search our memory to find the person who has the most similar appearance. We judge the new person's personality based on this person whose personality is already known to us.

This section discusses a machine learning algorithm called instance-based learning or memory-based reasoning. Memory-based collaborative filtering

discussed in Chapter 14 employs an instance-based learning algorithm to predict the preference of the target user. We here discuss instance-based learning in a broader context. Its original idea was developed by statisticians who employed the nearest-neighbor methods for prediction and classification tasks (Fix and Hodges 1951; Johns 1961). Aha (1992) provided techniques to handle noisy data in instance-based learning algorithms and popularized these methods among machine learning researchers.

19.4.1 Strengths and Limitations

Instance-based learning techniques have successfully been applied to various classification and prediction problems. Their broad appeal comes from several sources. Instance-based learning is simple to implement and its results are easy to understand. More importantly, it is very flexible in the required format of the input records (Berry and Linoff 1997). The algorithm only requires basic arithmetic operations in calculating similarities between instances. Hence, it can easily be applied to diverse data including images and text.

However, instance-based learning tends to be resource-intensive both in computation and storage space. We need to process all the historical instances to find the most similar instance and classify the new instance. Other data mining techniques such as neural networks spend much time analyzing the training examples and constructing models. Once a generalized model is constructed and saved, the classification of new instances is fast and easy. In contrast, instance-based learning takes a completely opposite approach. It is fast in the analysis of the training examples because it does not need to construct models or rules, but simply stores training examples for future use. Instance-based learning postpones major computations for later. It requires a great deal of computer memory for saving training examples that will be fetched when classifying new instances.

Results of instance-based learning are sensitive to the choice of distance function and the attribute-weighting scheme, the subset selection of the training samples, and the presence of outliers. As discussed later, a number of techniques have been developed to make the algorithm robust even though a single best algorithm does not exist.

19.4.2 A Brief Description of an Instance-Based Learning Algorithm

Suppose that we have N training instances in which we observe the classification membership of each instance with its attribute values. Let the category identification of instance i be y_i and the corresponding vector of attributes

be $\mathbf{x}_i = (x_{i1}, \ldots, x_{im})$ where m represents the number of attributes. Our goal is to classify a new instance that has attributes $\mathbf{z} = (z_1, \ldots, z_m)$.

The nearest-neighbor method searches over the training instances and finds the most similar instance to the new instance. There are a number of ways to measure the similarity or the distance between two instances. Since the issue of similarity is extensively discussed in the chapter of collaborative filtering, here we simply use the Euclidean distance. That is, the distance between a new observation and instance i in the training set is

$$d(\mathbf{z}, \mathbf{x}_i) = \sqrt{(z_1 - x_{i1})^2 + \cdots + (z_m - x_{im})^2} \tag{19.3}$$

All attributes are transformed to have a range from zero to one so that each attribute is treated identically in distance contribution. Once the distances are calculated over all training instances, we find the training instance with the minimum distance. The class of the new instance is predicted to be the same as the category membership y of this closest training instance.

The nearest-neighbor method tends to be sensitive to few noisy instances. In order to reduce the effect of noisy instances on the results, we adopt the k-nearest-neighbor method where we select the k nearest training instances and use them together to predict the class of the new instance. One possibility is to take the majority class from the k nearest training instances as the predicted class of the new instance. Alternatively, each of the k nearest training instances can get differential weights in predicting the class of the new instance (Berry and Linoff 1997). The more similar to the new instance, the larger weight it is assigned. The inverse of the distance to the new instance is used for the weight assigned to each of the k nearest training instances.[4]

The suitable value of k should be determined empirically. That is, the k-nearest-neighbors method is applied with different values of k and performance in a validation sample is compared. In general, the optimal value of k increases with the amount of noise.

19.4.3 Selection of Exemplars

Instance-based learning easily becomes computationally intensive because all the training examples must be scanned to predict the class of a new instance. Hence, we want to limit the number of training examples used for classification to be as small as possible. We do not want to store a lot of redundant training examples. At the same time we do not want to exclude meaningful examples. Witten and Frank (2000) use the term "exemplars" to refer to the already-seen instances that are used for classification. We start with an exemplar randomly chosen from the calibration sample and classify each

[4] In order to avoid the problem of inversing zero distance, it is common to add one before taking the inverse.

new instance from the calibration sample with respect to the exemplar. If the new instance is misclassified, it is added to the exemplar database. Correctly classified instances (e.g., redundant instances) are discarded. Ideally, each exemplar in the exemplar database represents an important region of the instance space. However, this method of only storing misclassified instances will not work well when the data have a lot of noise. Noisy instances by definition cannot be explained by the model and should not be incorporated into the exemplar database. However, noisy instances are very likely to be misclassified and so saved into the exemplar database leading to low predictive accuracy. Our goal is to find interesting patterns (e.g., exemplars) without recording redundant patterns. We do not want to interpret statistical noise as meaningful patterns.

Aha (1992) proposed an instance-based learning algorithm to overcome the above problem of noisy instances, while minimizing the size of the exemplar database. It applies an accuracy filter that monitors the predictive performance of each instance and stores only those instances showing high classification accuracy. That is, it keeps track of the classification performance of each training instance (e.g., the ratio of the correctly classified to the total number of classifications). A test based on confidence intervals for proportions is employed to decide whether instances are acceptable, noisy or uncertain. The idea is to determine whether the instance predicts better than random. Therefore, a confidence interval is computed for both the instance's (cumulative) classification performance and its class's observed frequency. If the lower bound of the confidence interval for the instance's classification performance exceeds above the upper bound of the confidence interval for its class's observed frequency, the instance is acceptable for classification decision, and hence becomes the member of the exemplar set. In contrast, if the upper bound of the confidence interval for the instance's classification performance drops below the lower bound of the confidence interval for its class's observed frequency, the instance is indefinitely removed from the exemplar set. If the instance's performance is somewhere between acceptable and unacceptable, it is considered to be a member of the potential set. The instances in the potential set are not allowed to be used for prediction but their predictive performances are continually updated and eventually may be added to the exemplar set or indefinitely removed from the potential set.

For example, suppose that an exemplar has been used n_1 times to classify instances, and x_1 of these predictions have been correct. We can estimate the confidence interval for the true success rate of this exemplar.[5] Now, suppose the exemplar's class has occurred x_2 times out of a total number n_2 of training instances. From this, we can compute the confidence interval for the default (random) success rate (x_2/n_2), the probability of successfully classifying an

[5] This would be calculated using the confidence interval for proportions, $p \pm z_{\alpha/2}\sqrt{\frac{p(1-p)}{n}}$, where $p = x_1/n_1$ and $z_{\alpha/2}$ is the normal deviate from the standard normal distribution corresponding to a confidence level of $1 - \alpha$.

instance of this class without relying on the exemplar's information. The following table illustrates:

Method	Prediction	95% Conf. Int.	Comparison
Using exemplar	$x_1 = 250$; $n_1 = 500$	0.500 ± 0.044	Lower Bound = 0.456
Random prediction	$x_2 = 400$; $n_2 = 1,000$	0.400 ± 0.030	Upper Bound = 0.430

Since the lower bound of the 95% confidence interval for predictive accuracy using the exemplar is higher than the upper bound for predictive accuracy using random prediction, the exemplar is acceptable and would be added to the exemplar set.

Aha (1992) uses a high confidence level (5%) for acceptance and a lower confidence level (12.5%) for rejection. This makes it statistically easier for an instance to be dropped than to be accepted, and reduces the number of exemplars.

19.4.4 Attribute Weights

The accuracy of instance-based learning is sensitive to the type of distance function employed. The Euclidean distance in the Equation 19.3 performs reasonably well. However, it is intuitively unappealing to assume that all attributes should contribute equally in defining the distance or similarity between two instances. Hence, we generalize the Euclidean distance to accommodate unequal weightings,

$$d(\mathbf{z}, \mathbf{x}_i) = \sqrt{w_1(z_1 - x_{i1})^2 + \cdots + w_m(z_m - x_{im})^2} \tag{19.4}$$

which becomes the Euclidean distance when all the weights are the same.

Aha (1992) suggests an algorithm to update attribute-specific weights dynamically. All weights are updated after we find the most similar exemplar to a new training instance. Given the new training instance $\mathbf{z}$ and the most similar exemplar $\mathbf{x}_i$, $|z_k - x_{ik}|$ is calculated for each attribute k. This is a measure of the contribution of attribute k to the classification decision. Attribute weight w_k is updated based on the size of $|z_k - x_{ik}|$ and whether the classification was indeed correct or not. If the new instance is classified correctly, the weight will be increased. If it is incorrect, the weight will decrease. And the difference $|z_k - x_{ik}|$ will determine the magnitude of this increase or decrease.

19.5 Genetic Algorithms

A genetic algorithm is a class of robust and efficient search techniques based on the concept of adaptation in natural organisms. The theory of evolution and survival of the fittest tells us that species (or individuals) well fitted to

their environment survive over millions of years and their chromosomes or genome would be propagated one generation to the next. In a genetic algorithm, an individual (or candidate solution) is selected for reproduction based on how its fitness (predictive ability) compares to that of other individuals. Individuals are traditionally represented in binary as strings of 0s and 1s. The evolution begins with randomly generated individuals (solutions). In each generation, once the fitness of every individual is evaluated, less fit individuals do not survive and genomes of more fit individuals are propagated.

Genetic algorithms became popular after John Holland developed its solid theoretical foundation in the early 1970s (Holland 1975). They have been successfully applied to complex problems in diverse fields. Within marketing, Hurley et al. (1995) outlined 11 areas in which genetic algorithms have been applied. Gatarski (2002) employed a genetic algorithm to automatically design banner advertising. The resulting system created innovative banner designs that performed increasingly better than reference banners, improving the click-through rate from 0.68% for the standard banner to 3.1% without human intervention from the advertiser. Kim et al. (2005) used artificial neural networks guided by genetic algorithms to identify and profile customers who are most likely interested in a particular product or service. The genetic algorithm in their model plays a key role in selecting input features for an artificial neural network.

A simple example provided by Berry and Linoff (1997) will help us to understand the basic concepts of genetic algorithms. Suppose we try to find the integer value of p maximizing $f(p) = 31p - p^2$ where the integer p varies between 0 and 31. We first represent a *solution* (e.g., the parameter value p from 0 to 31) as an array of bits. Five bits are required to represent all solutions (e.g., 0 to 31). For example, $p = 1$ can be represented by {00001}. The fitness function, defined over the genetic representation, measures the quality of the solution. In this example, the fitness function is $f(p)$. Once we have the genetic representation and defined the fitness function, the genetic algorithm generates an initial set of solutions randomly, and then improves them through repetitive application of selection, crossover and mutation. Suppose that we randomly generate the initial set of solutions below. The average fitness of the initial solutions is 117.75.

Solution	***p***	**Fitness**
10110	22	176
00011	3	87
00010	2	58
11001	25	150

We now improve the fitness of the initial solutions through three "operators." The first operator selects solutions with higher fitness for successive generations. That is, the chance of a solution surviving to the next generation is proportional to its fitness value. Specifically, the total fitness value of the above four solutions is 471. The ratio of the fitness of each solution to the

total fitness is 0.37, 0.19, 0.12 and 0.32, respectively. We now spin a roulette wheel four times in which each solution has an area of these four proportions. Each spin we select a solution (using sampling with replacement) to survive to the next generation. Below is the resulting set of solutions after selection through spinning a roulette wheel.

Solution	p	**Fitness**
10110	22	176
11001	25	150
00010	2	58
10110	22	176

Notice that the selection procedure results in more copies of the solutions with high fitness values and fewer for the less fit. One of the low fit solutions {00011} has not survived, but there are two copies of the solution with the highest fit {10110}. As a result, the average fitness value has increased from 117.25 to 140.

The second operator is crossover. This creates two new solutions (children) by selecting two existing ones (parents) and gluing pieces of each solution together. For example, suppose that two solutions {10110} and {00010} are selected from the selection process and a randomly chosen crossover position is between the second and the third position. Then the first two digits of the first solution (10) are replaced by the first two digits of the second solution (00). The resulting two children solutions – {00110} and {10010} – have a piece inherited from each of their parents solutions. The crossover probability of 0.5 is typically used. That is, once we select two solutions, we flip a coin to decide whether to apply the crossover operation to these two solutions.

The final operator is mutation. Its role is to prevent some valuable combinations from consideration in succeeding generations. Solutions generated by search and crossover depend on the initial set of solutions. Mutation provides an additional input not drawn from the initial set. Mutation therefore helps avoid premature convergence to a local optimum. The probability of a mutation is typically kept very small. When a mutation occurs, a bit changes from 0 to 1, or 1 to 0. For instance, if the mutation occurs in the solution {10010}, we randomly determine the position of mutation. If it is the third position, which is 0, it is changed to 1. The resulting solution becomes {10110}. Changes introduced by mutation may often decrease the average fitness value. But those mutated solution will not survive long.

Genetic algorithms improve the fitness of the population by selection, crossover and mutation as genes are passed from one generation to the next. The generational process continues until a termination condition (e.g., fixed number of generations) has been reached. Genetic algorithms do not guarantee the exact optimal solution, but they do a good job of getting close to the optimal solution quickly. See Chapter 27 for a detailed discussion of Gatarski (2002) use of a genetic algorithm to develop banner advertisements.

19.6 Bayesian Networks

Bayesian networks have made significant contributions in various fields such as software engineering, space navigation and medical diagnosis since Pearl (1988) published the first book on the topic (Haddawy 1999). Unlike traditional statistical models (e.g., logistic regression), Bayesian networks do not make any stringent assumptions on the types of data and their distributions (e.g., normality). In addition, they can effectively handle nonlinearity and take on any structure (Heckerman 1997; Cui et al. 2006).

Marketing researchers have begun to recognize Bayesian networks as an alternative to structural equations models. Cooper (2000) used Bayesian networks for strategic marketing planning of radically new products. The factors identified in an extensive situation analysis are woven into the economic webs surrounding the new product. The webs are mapped into Bayesian networks that can be updated as events unfold and used to simulate the impact that changes in assumptions underlying the web have on the prospects for the new product. Blodgett and Anderson (2000) applied a Bayesian network to the consumer complaint process. They generated several interesting findings. For example, the probability that a noncomplainer or a dissatisfied complainant will completely exit is quite low. On the other hand, the probability that a satisfied complainant will fully repatronize the retailer and engage in positive word of mouth is quite high. Finally, Cui et al. (2006) employed Bayesian networks (learned by evolutionary programming) to model consumer responses to direct marketing. They compared Bayesian networks with neural networks, decision trees (e.g., CART), and latent class regression, and found that Bayesian networks have advantages in accuracy of prediction,[6] transparency of procedures, interpretability of results, and explanatory insight.

The key feature of Bayesian networks is to provide a method of decomposing a complex joint probability distribution into a set of simple local distributions. Bayesian networks achieve this simplification through a directed acyclic graph. Nodes in the graph represent (random) variables and arcs represent the probabilistic correlation between the variables. Suppose we are interested in looking for relationships among a set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. The Bayesian network is a graphical model that efficiently represents ("encodes") the joint distribution for $\mathbf{X}$. It consists of (1) a network structure S that encodes a set of conditional independence assumptions on variables in $\mathbf{X}$, and (2) a set P of local probability distributions associated with each variable (Herkerman 1997). The structure S is a directed acyclic graph whose nodes represent variables X_i's. If there is an arc from variable X_i to X_j, then variable X_j depends directly on X_i and X_i is called a parent of X_j. We denote $parents(X_i)$ to be the parents of variable X_i in S as well as the variables corresponding to those parents. Given a structure$S$, the joint probability

[6] In their tenfold cross-validation comparison, Bayesian networks provide the highest average lift in the top decile, followed by latent class regression, neural networks, and CART. In the second decile, however, the neural networks have the highest cumulative lift, followed by Bayesian networks, CART, and latent class regression.

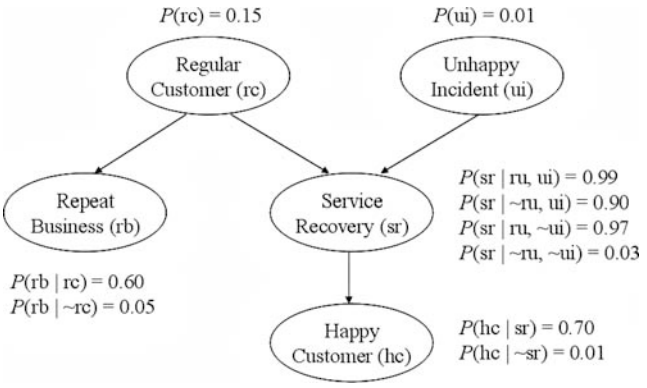

Fig. 19.3 A Bayesian network model for customer complaints (Reprinted by permission, from Cui, G., Wong, M.L., and Lui, H.K., "Machine Learning for Direct Marketing Response Models: Bayesian Networks with Evolutionary Programming", Management Science, Volume 52, Number 4 (April 2006), The Institute for Operations Research and the Management Sciences (INFORMS), 7240 Parkway Drive, Suite 310, Hanover, MD 21076 USA.)

distribution for $\mathbf{X}$ is

$$p(\mathbf{X}) = \prod_{i=1}^{n} p(X_i|parents(X_i)) \tag{19.5}$$

The local probability distributions P are the distributions of $p(X_i|parents(X_i))$. As a result, the pair (S, P) encodes the joint distribution $p(\mathbf{X})$.

One advantage of a Bayesian network is that it is intuitively easier to understand its direct dependencies and local distributions than the joint distribution. For example, Fig. 19.3 shows a Bayesian network model for handling customer complaints provided by Cui et al. (2006). There are five binary variables: regular customer, unhappy incident, service recovery, repeat business and happy customer. The joint probability distribution table (without the Bayesian network structure) would have $2^5 - 1 = 31$ entries. However, a Bayesian network model in Fig. 19.3 has only 10 probability values.

The success of a Bayesian network mainly depends on a network structure S that encodes a set of conditional independence assertions about variables in $\mathbf{X}$. In the model of customer complaints in Fig. 19.3, we have the conditional independencies

$$\begin{aligned} P(rc|ui) &= P(rc) \\ P(rb|rc, ui) &= P(rb|rc) \\ P(sr|rc, ui, rb) &= P(sr|rc, ui) \\ P(hc|sr, rc, ui, rb) &= P(hc|sr, rc, ui) \end{aligned}$$

For example, the first relationship depicts the assertion that being a regular customer is unrelated to whether an unhappy incident occurs. Given the above four conditional independencies, we can obtain the model structure for the customer complaint model in Fig. 19.3.

Although the structure of a Bayesian network is formally defined as a set of conditional independence assertions about variables, it is often constructed using the notion of causal relationships (Heckerman 1997). In particular, we simply draw arcs from cause variables to their effect variables. For instance, given the assertion that *Regular Customer* is a direct cause of *Repeat Business* and *Service Recovery*, *Unhappy Incident* is a direct cause of *Service Recovery*, and *Service Recovery* is the direct cause of *Happy Customer*, we obtain the network structure in Fig. 19.3.

Once we estimate the Bayesian network model, we can make probabilistic inferences directly from the local probabilities $p(X_i|parents(X_i))$.[7] The inference is based on the direct causes as depicted in the model specification, without making any restrictive distributional assumptions on the local probabilities. For example, *Service Recovery* (sr) leads to *Happy Customer* (hc) in Fig. 19.3. Then

$$\begin{aligned} P(sr) &= P(sr|rc,ui)P(rc)P(ui) + P(sr|\sim rc,ui)P(\sim rc)P(ui) \\ &\quad + P(sr|rc,\sim ui)P(rc)P(\sim ui) + P(sr|\sim rc,\sim ui)P(\sim rc)P(\sim ui) \\ &= (.99)(.15)(.01) + (.90)(.85)(.01) + (.97)(.15)(.99) + (.03)(.85)(.99) \\ &= 0.1784 \\ P(hc) &= P(hc|sr)P(sr) + P(hc\,|\sim sr)P(\sim sr) = (.7)(.1784) + (.01)(.8216) \\ &= 0.1331 \end{aligned}$$

We can also calculate the posterior probability of *Service Recovery* when the chosen customer is a *Happy Customer* by

$$P(sr|hc) = P(hc|sr)P(sr)/P(hc) = (.7)(.1784)/(.1331) = 0.9383$$

Blodgett and Anderson (2000) discuss the advantages of Bayesian networks over traditional causal models such as structural equation models. As mentioned, Bayesian networks are nonparametric so that no functional form or distributional assumptions are necessary for inference. In contrast, structural equation models are parametric in both distribution and function (e.g., normality, linearity and common factor theory). Another advantage of Bayesian networks is that both forward and backward inferences are possible. We can make predictions with forward inferences (e.g., $P(hc|sr)$) whereas backward inferences can address profile questions (e.g., $P(rc|hc)$). The main weaknesses of Bayesian networks are computational complexities and a relatively large sample size necessary for estimation.

19.7 Support Vector Machines

Support vector machines, pioneered by Vapnik and co-workers, originated from statistical learning theory (Boser et al. 1992; Vapnik 1995). In this

[7] Estimation of Bayesian networks is beyond the scope of this book. See Heckerman (1997) and Cui et al. (2006) for more discussions on estimation, and Haddawy (1999) for commercial software packages.

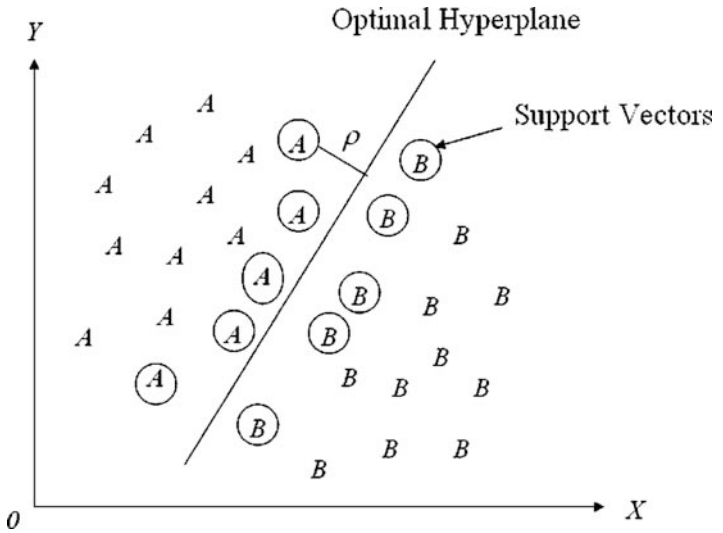

Fig. 19.4 Optimal hyperplane and support vectors in feature space.

section, we briefly describe the fundamental concepts of support vector machines and their advantages over traditional statistical methodologies since their complete treatment is mathematically demanding. See Burges (1998) for technical details of support vector machines and Cui and Curry (2005) for their marketing application.

In Chapter 15, we describe how discriminant analysis performs classification. Its key idea is to express the class as a linear combination of the independent variables (or attributes). For example, consider the training sample of size N, $\{\mathbf{x}_i, y_i\}$ where $\mathbf{x}_i$ is the vector of independent variables for observation i $(i = 1, \ldots, N)$ and y_i is the corresponding response. To simplify our exposition, we assume that the class represented by the subset $d_i = 1$ is linearly separable from the class of the subset $d_i = -1$. The hyperplane that linearly separates the two classes has the functional form $\boldsymbol{\beta}'\mathbf{x} = \beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_k x_k = 0$. That is, $\beta'\mathbf{x} > 0$ for $d_i = 1$ and $\beta'\mathbf{x} < 0$ for $d_i = -1$. The goal of support vector machine is to find the particular hyperplane (called the optimal hyperplane) which maximally separates two classes (Witten and Frank 2000; Flach 2001). Figure 19.4 illustrates the optimal hyperplane for 30 training instances with two independent variables.

Suppose weighting vector $\boldsymbol{\beta}_o$ defines the optimal hyperplane. That is, the optimal hyperplane is algebraically defined as $\boldsymbol{\beta}_o'\mathbf{x} = 0$. The observations that are closest to the optimal hyperplane are called support vectors $\mathbf{x}_s$.[8] These vectors play a critical role in support vector machines. Since support

[8] We can derive the optimal weighting vector by solving a quadratic optimization problem. Support vectors can be derived once the optimal weighting vector is determined. For detailed discussion, see Cui and Curry (2005) or Haykin (1999).

vectors lie closest to the decision surface, they are the most difficult to classify. Support vectors reduce the number of instances (or observations) required to predict or classify a new instance. Figure 19.4 shows the scatter plot of 30 training instances with two attributes. All instances above the optimal hyperplane are class *A*s while all instances below the hyperplane are class *B*s. Only the instances circled – called support instances – are required to classify new instances. The rest of the instances play no role in predicting the class of new instances. As a result, we reduce the number of instances from 30 to 10. The set of support vectors uniquely defines the optimal hyperplane.

The biggest disadvantage of the linear hyperplane is that it can only represent linear boundaries between classes (Witten and Frank 2000). One way of overcoming this restriction is to transform the instance space into a new "feature" space with a nonlinear mapping. A straight line in feature space does not look straight in the original instance space. That is, a linear model constructed in feature space can represent a nonlinear boundary in the original space. For example, given two independent variables, the linear model is $Y = \beta_1 x_1 + \beta_2 x_2$. If we allow for all products with two factors, we have $Y = \alpha_1 x_1^2 + \alpha_2 x_1 x_2 + \alpha_3 x_2^2$. The original observations (or instances) x's are mapped into a feature space of z's (e.g., $z_1 = x_1^2$, $z_2 = x_1 x_2$ and $z_3 = x_2^2$). The model is nonlinear in the original space whereas it is linear in feature space. We can add more flexibility to the model by assuming polynomials of sufficiently high degree (instead of two factors).

The idea of support vector machines is based on two mathematical operations: (1) nonlinear mapping of the original instance space into a high-dimensional feature space, and (2) construction of an optimal hyperplane to separate the classes. In other words, we need to derive the optimal hyperplane defined as a linear function of vectors drawn from the feature space rather than the original instance space (Haykin 1999). We construct this hyperplane in accordance with the principle of structural risk minimization (Vapnik 1995).

Let $\mathbf{x}$ denote a vector of independent variables drawn from the input space, assumed to be of dimension K. Let $\varphi_m(\mathbf{x})$ denote a set of nonlinear transformations from the input space to the feature space where the dimension of the feature space is M. Define the feature vector $\varphi(\mathbf{x}) = [\varphi_0(\mathbf{x}), \varphi_1(\mathbf{x}), \ldots, \varphi_M(\mathbf{x})]'$ where $\varphi_0(\mathbf{x}) = 1$ for all $\mathbf{x}$. Similar to the linear hyperplane, we can define a hyperplane given a set of features:

$$\boldsymbol{\beta}'\varphi(\mathbf{x}) = 0 \tag{19.6}$$

We can now derive the weighting vector $\boldsymbol{\beta}_o$ that defines the hyperplane in feature space that optimally separates the classes (see Haykin (1999) for its detailed derivation):

$$\boldsymbol{\beta}_o = \sum\nolimits_{i=1}^{I} \alpha_i d_i \varphi(\mathbf{x}_i) \tag{19.7}$$

where I is the number of support vectors, $\varphi(\mathbf{x}_i)$ denotes the feature vector corresponding to the input pattern $\mathbf{x}_i$ in the ith support vector, d_i is the

response (or class) indicator (1 or −1) and α_i is the Lagrange multiplier. Substituting Equation 19.7 into 19.6, we have the optimal hyperplane.

$$\sum_{i=1}^{I} \alpha_i d_i \varphi'(\mathbf{x}_i)\varphi(\mathbf{x}) = \sum_{i=1}^{I} \alpha_i d_i K(\mathbf{x}_i, \mathbf{x}) = 0 \tag{19.8}$$

The term in Equation 19.8 denoted by $K(\mathbf{x}_i, \mathbf{x}) = \varphi'(\mathbf{x}_i)\varphi(\mathbf{x}) = \sum_{m=0}^{M} \varphi_m(\mathbf{x})$ $\varphi_m(\mathbf{x}_i)$ is called the inner-product kernel. Note that M is the number of features plus the intercept. It represents the inner product of two vectors induced in the feature space by the input vector $\mathbf{x}$ and the input pattern $\mathbf{x}_i$ pertaining to the ith support vector. Various functions can be employed for the inner-product kernel. The most popular one may be a polynomial kernel where $K(\mathbf{x}_i, \mathbf{x}) = (\mathbf{x}'\mathbf{x}_i + 1)^p$ where power p is specified a priori by the user. A good way of choosing the value of p is to start with 1 (e.g., linear model) and increase the value until the errors cease to improve (Witten and Frank 2000). An alternative kernel function is the radial-basis function kernel where $K(\mathbf{x}_i, \mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2 / (2\sigma^2))$ where σ^2 is specified a priori by the user. The support vector machine with the radial-basis function kernel is simply a radial-basis function network that is a type of neural network specification (see Chapter 18). Finally, the support vector machine with the logistic kernel is a multilayer perceptron with no hidden layers which is another type of neural network. This is why support vector machines are often considered an extension of neural networks (Flach 2001). However, support vector machines offer a much more sophisticated mechanism to incorporate domain knowledge by means of kernels.

The support vector machine is an excellent semi-parametric technique that has great potential in prediction and classification. Cui and Curry (2005) introduced support vector machines to marketing, and showed it has significantly better predictive performance over the multinomial logit model in a simulation test. Its major drawbacks for database marketers may be that its implementation is mathematically complex and there is no off-the-shelf software available. However, we expect that in the near future database marketers will use support vector machines especially when the relationship between independent variables and the dependent variable is complex.

19.8 Combining Multiple Models: Committee Machines

When people make critical decisions, they usually consider opinions from several experts. In machine learning, each estimated model for a given set of training data constitutes an expert. We may be able to improve our decision (or prediction) if we combine the outputs from several different models. The combination of experts is said to constitute a "committee machine" (Haykin 1999). The original idea of a committee machine is traced back to Nilsson (1965), but was made popular by Breiman (1996). There are various ways of combining multiple models; they potentially perform better than a single

model. However, they all share a disadvantage of interpretation. It is difficult to understand which factors are contributing to improved prediction (Witten and Frank 2000).

Committee machines have received increasing attention in various disciplines (see Haykin (1999) and Lemmens and Croux (2006) for references). In marketing, Lemmens and Croux (2006) applied the bagging and boosting of binary logit to predicting churn in a wireless telecommunications company. They showed that both bagging and boosting significantly improved prediction accuracy compared to a binary logit.

19.8.1 Bagging

A simple way of combining the predictions from multiple models into a single prediction is to take a (weighted) vote for a classification task and a (weighted) average for numeric prediction. Bagging adopts this simple approach. We denote the estimation (or calibration) sample as $\mathbf{Z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ where N is the total number of observations, $\mathbf{x}_i$ represents k predictors for observation i ($\mathbf{x}_i = (x_{i1}, \ldots, x_{iK})$), and y_i represents the value of the dependent variable for observation i. From the original estimation sample, we generate B bootstrap samples of size $N, \mathbf{Z}_1, \mathbf{Z}_{2,} \ldots, \mathbf{Z}_B$. The term bagging actually stands for "bootstrap aggregating" (Breiman 1996). As we described in Chapter 11, each observation of size N in each bootstrap sample is randomly drawn from the original estimation sample, with replacement. Through this sampling procedure, each bootstrap sample deletes and replicates some observations in the original sample. Once B bootstrap samples are constructed, a model (e.g., logistic regression or decision trees) is applied to each bootstrap sample. Prediction is based on a vote for classification from each estimated model and an average value for numeric prediction.

The effect of bagging can be viewed through the statistical lens of bias-variance decomposition (Witten and Frank 2000). Let $\mathbf{x}$ denote a set of independent variables not seen before and y denote the corresponding dependent variable. That is, $\mathbf{x}$ and y are realizations of the random vector $\mathbf{X}$ and random variable Y. Let $F(\mathbf{x})$ denote the model predicting y. The expected mean squared error of $F(\mathbf{x})$ with respect to $E[Y|\mathbf{X} = \mathbf{x}]$ can be decomposed into the bias and the variance.

$$
\begin{aligned}
&E[MSE] \\
&\qquad = E[\mathrm{Pr}\, edictd - Actual] \\
&\qquad = E[(F(\mathbf{x}) - E[Y|\mathbf{X} = \mathbf{x}])^2] \\
&\qquad = (E[F(\mathbf{x})] - E[Y|\mathbf{X} = \mathbf{x}])^2 + E[(F(\mathbf{x}) - E[F(\mathbf{x})])^2] \\
&\qquad = B[F(\mathbf{x})] + V[F(\mathbf{x})]
\end{aligned}
\tag{19.9}
$$

where $B[F(\mathbf{x})]$ is the bias squared and $V[F(\mathbf{x})]$ is the variance. The bias of the model is a systematic error that cannot be eliminated even by employing an infinite number of observations for estimation. However, the variance component comes from the particular estimation sample used, which is randomly selected from the true population and so is not perfectly representative. Bagging reduces the mean square error by decreasing the variance of the model (Haykin 1999). That is, bagging averages out the instability of a model applied to a particular estimation sample by constructing several bootstrap samples.

19.8.2 Boosting

Bagging will be effective when the model's estimated parameters change drastically over different bootstrap samples. That is, bagging addresses the instability of the model. Therefore, bagging does not work well with a robust model (e.g., linear model) in which predictions of the model change very little across bootstrap samples. Intuitively, it is only reasonable to combine multiple models with significantly different predictions (Witten and Frank 2000). That is, combining methods will be effective when models complement each other. Boosting exploits this insight by searching for models that complement each other.

Boosting is similar to bagging in combining predictions (votes or numerical values) of individual models. However, the main difference between boosting and bagging lies in the sampling scheme. Boosting sequentially estimates a model applied to adaptively re-weighted samples. More specifically, the boosting algorithm starts with assigning equal weight to all observations in the calibration sample. Once the model is estimated, we re-weight each observation according to the model's output. The weight of correctly classified observations is decreased, and that of misclassified ones is increased (Witten and Frank 2000). With this weighting scheme, we attempt to focus on classifying observations with high weights that are hard-to-classify (or misclassified) observations correctly. Such observations become more important because there is greater incentive to classify them correctly. By assigning weights to each observation, boosting provides an elegant way of generating a series of experts that complement one another. In the next iteration, the model is applied to the re-weighted calibration sample. There are several weighting schemes available, but we only introduce a widely used method called AdaBoost.M1 (Freund and Schapire 1996). For each iteration, the weights of all observations are updated as

$$w_{i,t+1} = w_{i,t}\{(1 - D_{i,t}) + D_{i,t}e_t/(1 - e_t)\} \tag{19.10}$$

where $w_{i,t}$ represents the weight of observation i at iteration t, $D_{i,t}$ indicates whether the observation is correctly classified (= 1 if correctly classified, and

0 if misclassified), and e_t denotes the model's overall error (the percentage misclassified) for the tth re-weighted calibration sample. Since each successive sample is weighted more toward harder-to-classify observations, e_t tends to increase with t. We stop the iterations when classification becomes so difficult that e_t is worse than random prediction, i.e., greater than or equal to 0.5. So assuming $e_t \leq 0.5$ and using Equation 19.10, the weights do not change for misclassified observations while they decrease for correctly classified ones. After all weights have been updated, they are renormalized such that their sum at iteration $t+1$ remains the same as the sum of weights at iteration t. For example, the renormalized weights of observation i at iteration t become $w^*_{i,t+1} = w_{i,t+1}(\sum_i w_{i,t} / \sum_i w_{i,t+1})$. As a result, the renormalized weights of the misclassified observations increase whereas those of the correctly classified ones decrease.

19.8.3 Other Committee Machines

Although bagging and boosting are popular, there are other methods to combine outputs from multiple models. For example, Breiman (2001) proposed a new classifier called a random forest which combines bagging by Breiman (1996) and the random subspace method by Ho (1998). Similar to bagging, a random forest combines the predictions from multiple (tree) models into a single prediction by generating multiple bootstrap samples and taking a weighted vote for a classification task. However, for each node of the tree, it randomly chooses m input variables (out of $M > m$ input variables) on which to base the decision at that node. That is, a random forest incorporates two types of randomness, one from training samples and the other from the input variables.

On the other hand, Wolpert (1992) has proposed another interesting committee machine called the stacked generalization (called "stacking"). Its main difference from bagging and boosting is its capability of combining multiple models of different types. Stacking can combine different models (e.g., logistic regression and decision trees) by introducing a "meta learner". The predictions of the base models (called level-0 models) become the input to the meta model (also called level-1 model). The meta model attempts to discover which base modes are the reliable ones and how to combine the predictions of the base models. Stacking has not been used as often as bagging and boosting, partially because it is analytically more demanding. For further materials on stacking, see Witten and Frank (2000), and Ting and Witten (1997).

Other techniques of combining multiple models include error-correcting output codes (Dietterich and Bakiri 1995) and mixture of experts (Haykin 1999).