10

Classification[1]

Classification models are supervised learning methods for predicting the value of a categorical target attribute, unlike regression models which deal with numerical attributes. Starting from a set of past observations whose target class is known, classification models are used to generate a set of rules that allow the target class of future examples to be predicted.

Classification holds a prominent position in learning theory due to its theoretical implications and the countless applications it affords. From a theoretical viewpoint, the development of algorithms capable of learning from past experience represents a fundamental step in emulating the inductive capabilities of the human brain.

On the other hand, the opportunities afforded by classification extend into several different application domains: selection of the target customers for a marketing campaign, fraud detection, image recognition, early diagnosis of diseases, text cataloguing and spam email recognition are just a few examples of real problems that can be framed within the classification paradigm.

In this chapter, we will describe the general characteristics of classification problems, discussing the main criteria used to evaluate and to compare different models. We will then review the major classification methods: classification trees, Bayesian methods, neural networks, logistic regression and support vector machines.

10.1 Classification problems

In a classification problem, we have a dataset $\mathcal{D}$ containing m observations described in terms of n *explanatory* attributes and a categorical *target* attribute.

[1]This chapter was co-written by Carlotta Orsenigo.

Business Intelligence: *Data Mining and Optimization for Decision Making* *Carlo Vercellis*
 ISBN: 978-0-470-51138-1

The explanatory attributes, also called *predictive variables*, may be partly categorical and partly numerical. The target attribute is also called a *class* or *label*, while the observations are also termed *examples* or *instances*. Unlike regression, for classification models the target variable takes a finite number of values. In particular, we have a *binary* classification problem if the instances belong to two classes only, and a *multiclass* or *multicategory* classification if there are more than two classes.

The purpose of a classification model is to identify recurring relationships among the explanatory variables which describe the examples belonging to the same class. Such relationships are then translated into *classification rules* which are used to predict the class of examples for which only the values of the explanatory attributes are known. The rules may take different forms depending on the type of model used.

Example 10.1 – Retention in the mobile phone industry. Example 5.2 on the analysis of customer loyalty in the mobile phone industry is a binary classification problem in which the target attribute takes the value 1 if a customer has discontinued service and 0 otherwise. The features of each customer, described in Table 5.3, represent the predictive attributes. The purpose of a classification model is to derive general rules from the examples contained in the dataset and to apply these rules in order to assign the class to new instances for which the target value is unknown. In this way, the classification model may prove useful in identifying those customers who are likely to discontinue service, and therefore to drive a retention marketing campaign.

Example 10.2 – Segmentation of customers phoning a call center. Many services and manufacturing companies nowadays have a call center that their customers may call to request information or report problems. In order to size the staff and the activities of a call center and to verify the quality of the services offered, it is useful to classify customers based on the number of calls made to the call center. The target attribute may be obtained through a proper discretization of the numerical variable indicating the number of calls, setting for example: class $0 \equiv$ no calls, class $1 \equiv 1$ call, class $2 \equiv$ from 2 to 4 calls, class $4 \equiv$ more than 4 calls. Again predictive attributes are provided by the features of the customers. Hence, the segmentation of the customers with respect to the number of calls made to the call center is a multicategory classification problem.

From a mathematical viewpoint, in a classification problem m known examples are given, consisting of pairs $(\mathbf{x}_i, y_i)$, $i \in \mathcal{M}$, where $\mathbf{x}_i \in \mathbb{R}^n$ is the vector of the values taken by the n predictive attributes for the ith example and $y_i \in \mathcal{H} = \{v_1, v_2, \ldots, v_H\}$ denotes the corresponding target class. Each component x_{ij} of the vector $\mathbf{x}_i$ is regarded as a realization of the random variable X_j, $j \in \mathcal{N}$, which represents the attribute $\mathbf{a}_j$ in the dataset $\mathcal{D}$. In a binary classification problem one has $H = 2$, and the two classes may be denoted as $\mathcal{H} = \{0, 1\}$ or as $\mathcal{H} = \{-1, 1\}$, without loss of generality.

Let $\mathcal{F}$ be a class of functions $f(\mathbf{x}) : \mathbb{R}^n \mapsto \mathcal{H}$ called *hypotheses* that represent hypothetical relationships of dependence between y_i and $\mathbf{x}_i$. A *classification problem* consists of defining an appropriate hypothesis space $\mathcal{F}$ and an algorithm $A_{\mathcal{F}}$ that identifies a function $f^* \in \mathcal{F}$ that can optimally describe the relationship between the predictive attributes and the target class. The joint probability distribution $P_{\mathbf{x},y}(\mathbf{x}, y)$ of the examples in the dataset $\mathcal{D}$, defined over the space $\mathbb{R}^n \times \mathcal{H}$, is generally unknown and most classification models are nonparametric, in the sense that they do not make any prior assumption on the form of the distribution $P_{\mathbf{x},y}(\mathbf{x}, y)$.

The flow diagram shown in Figure 10.1 may clarify the probability assumptions concerning the three components of a classification problem: a *generator* of observations, a *supervisor* of the target class and a *classification algorithm*.

Generator. The task of the generator is to extract random vectors $\mathbf{x}$ of examples according to an unknown probability distribution $P_{\mathbf{x}}(\mathbf{x})$.

Supervisor. The supervisor returns for each vector $\mathbf{x}$ of examples the value of the target class according to a conditional distribution $P_{y|\mathbf{x}}(y|\mathbf{x})$ which is also unknown.

Algorithm. A classification algorithm $A_{\mathcal{F}}$, also called a *classifier*, chooses a function $f^* \in \mathcal{F}$ in the hypothesis space so as to minimize a suitably defined loss function.

Classification models, just like regression models, are suitable for both interpretation and prediction, as observed in Section 5.1 with general reference to

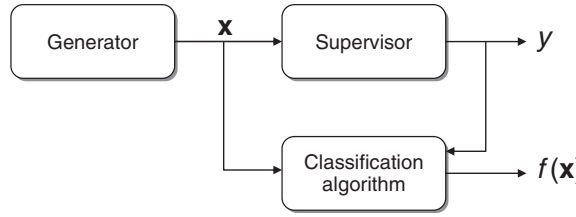

Figure 10.1 Probabilistic structure of the learning process for classification

supervised learning methods. Simpler models usually yield intuitive classification rules that can be easily interpreted, while more advanced models derive less intelligible rules although they usually achieve better prediction accuracy.

A portion of the examples in the dataset $\mathcal{D}$ is used for *training* a classification model, that is, for deriving the functional relationship between the target variable and the explicative variables expressed by means of the hypothesis $f^* \in \mathcal{F}$. What remains of the available data is used later to evaluate the accuracy of the generated model and to select the best model out of those developed using alternative classification methods.

The development of a classification model consists therefore of three main phases.

Training phase. During the *training* phase, the classification algorithm is applied to the examples belonging to a subset $\mathcal{T}$ of the dataset $\mathcal{D}$, called the *training set*, in order to derive classification rules that allow the corresponding target class y to be attached to each observation $\mathbf{x}$.

Test phase. In the *test* phase, the rules generated during the training phase are used to classify the observations of $\mathcal{D}$ not included in the training set, for which the target class value is already known. To assess the accuracy of the classification model, the actual target class of each instance in the *test set* $\mathcal{V} = \mathcal{D} - \mathcal{T}$ is then compared with the class predicted by the classifier. To avoid an overestimate of the model accuracy, the training set and the test set must be disjoint.

Prediction phase. The *prediction* phase represents the actual use of the classification model to assign the target class to new observations that will be recorded in the future. A prediction is obtained by applying the rules generated during the training phase to the explanatory variables that describe the new instance.

Figure 10.2 shows the logical flow of the learning process for a classification algorithm. In addition to the phases described above, Figure 10.2 also shows a portion of the training data, called the *tuning set*, which is used by some classification algorithms to identify the optimal value of some parameters appearing in the functions $f \in \mathcal{F}$.

10.1.1 Taxonomy of classification models

Before describing the different types of classifiers in the next sections, it may be useful to provide a taxonomy of classification models in order to place each individual algorithm in a broader framework. We can distinguish four main categories of classification models.

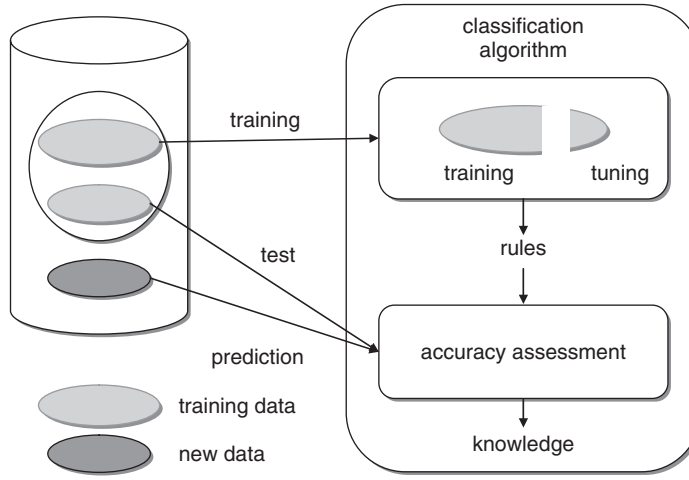

Figure 10.2 Phases of the learning process for a classification algorithm

Heuristic models. Heuristic methods make use of classification procedures based on simple and intuitive algorithms. This category includes *nearest neighbor* methods, based on the concept of distance between observations, and *classification trees*, which use *divide-and-conquer* schemes to derive groups of observations that are as homogeneous as possible with respect to the target class.

Separation models. Separation models divide the attribute space $\mathbb{R}^n$ into H disjoint regions $\{S_1, S_2, \ldots, S_H\}$, separating the observations based on the target class. The observations $\mathbf{x}_i$ in region S_h are assigned to class $y_i = v_h$. Each region S_h may comprise a composite set obtained from set-theoretic operations of union and intersection applied to regions of an elementary form, such as halfspaces or hyperspheres. However, the resulting regions should be kept structurally simple, to avoid compromising the generalization capability of the model. In general, it is hard to determine a collection of simple regions that exactly subdivide the observations of the dataset based on the value of the target class. Therefore, a loss function is defined to take the misclassified points into account, and an optimization problem is solved in order to derive a subdivision into regions that minimizes the total loss. The various classification models that belong to this category differ from one another in terms of the type of separation regions, the loss function, the algorithm used to solve the optimization problem. The most popular separation techniques include *discriminant analysis*, *perceptron methods*, *neural networks* and *support vector machines*. Some variants of classification trees can also be placed in this category.

Regression models. Regression models, described in Chapter 8 for the prediction of continuous target variables, make an explicit assumption concerning the functional form of the conditional probabilities $P_{y|\mathbf{x}}(y|\mathbf{x})$, which correspond to the assignment of the target class by the supervisor. For linear regression models, it is assumed that a linear relationship exists between the dependent variable and the predictors, and this leads to the derivation of the value of the regression coefficients. We will see in Section 10.5 that *logistic regression* is an extension of linear regression suited to handling binary classification problems.

Probabilistic models. In probabilistic models, a hypothesis is formulated regarding the functional form of the conditional probabilities $P_{\mathbf{x}|y}(\mathbf{x}|y)$ of the observations given the target class, known as *class-conditional probabilities*. Subsequently, based on an estimate of the *prior* probabilities $P_y(y)$ and using Bayes' theorem, the *posterior* probabilities $P_{y|\mathbf{x}}(y|\mathbf{x})$ of the target class assigned by the supervisor can be calculated. The functional form of $P_{\mathbf{x}|y}(\mathbf{x}|y)$ may be parametric or nonparametric. Naive Bayes classifiers and Bayesian networks are well-known families of probabilistic methods.

Besides the taxonomy described above, it should be mentioned that most classifiers also generate a *score function* $g(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ which associates each observation $\mathbf{x}$ with a real number. Possibly after standardization, the score function may be interpreted as an estimate of the probability that the class predicted by the classifier for the observation $\mathbf{x}$ is correct. This property is evident in probabilistic models. In separation models, the score of an observation may be set equal to its distance from the boundary of the region corresponding to its target class value. Finally, in classification trees the score function is associated with the density of observations having the same target class which have been included in the leaf node to which the observation itself has been assigned, as we will see in detail in Section 10.3.

Starting from the score function g, it is possible to derive a classification rule to predict the target class of the observation $\mathbf{x}$. For example, for the separation models applied to binary classification problems one may set $f(\mathbf{x}) = \text{sgn}(g(\mathbf{x}))$, where $\text{sgn}(\cdot)$ is a function indicating the sign of its argument, and therefore takes the values of the two classes $\{-1, 1\}$.

10.2 Evaluation of classification models

Within a classification analysis it is usually advisable to develop alternative models and then select the method affording the best prediction accuracy. To obtain alternative models one may vary the method, by using for instance classification trees, neural networks, Bayesian techniques or support vector

machines, and also modify the values of the parameters involved. Classification methods can be evaluated based on several criteria, as follows.

Accuracy. Evaluating the accuracy of a classification model is crucial for two main reasons. First, the accuracy of a model is an indicator of its ability to predict the target class for future observations. Based on their accuracy values, it is also possible to compare different models in order to select the classifier associated with the best performance. Let $\mathcal{T}$ be the training set and $\mathcal{V}$ the test set, and t and v be the number of observations in each subset, respectively. The relations $\mathcal{D} = \mathcal{T} \cup \mathcal{V}$ and $m = t + v$ obviously hold. The most natural indicator of the accuracy of a classification model is the proportion of observations of the test set $\mathcal{V}$ correctly classified by the model. If y_i denotes the class of the generic observation $\mathbf{x}_i \in \mathcal{V}$ and $f(\mathbf{x}_i)$ the class predicted through the function $f \in \mathcal{F}$ identified by the learning algorithm $A = A_{\mathcal{F}}$, the following loss function can be defined:

$$L(y_i, f(\mathbf{x}_i)) = \begin{cases} 0, & \text{if } y_i = f(\mathbf{x}_i), \\ 1, & \text{if } y_i \neq f(\mathbf{x}_i). \end{cases} \tag{10.1}$$

The accuracy of model A can be evaluated as

$$\text{acc}_A(\mathcal{V}) = \text{acc}_{A_{\mathcal{F}}}(\mathcal{V}) = 1 - \frac{1}{v} \sum_{i=1}^{v} L(y_i, f(\mathbf{x}_i)). \tag{10.2}$$

In some cases, it is preferable to use an alternative performance indicator given by the proportion of errors made by the classification algorithm:

$$\text{err}_A(\mathcal{V}) = \text{err}_{A_{\mathcal{F}}}(\mathcal{V}) = 1 - \text{acc}_{A_{\mathcal{F}}}(\mathcal{V}) = \frac{1}{v} \sum_{i=1}^{v} L(y_i, f(\mathbf{x}_i)). \tag{10.3}$$

Speed. Some methods require shorter computation times than others and can handle larger problems. However, classification methods characterized by longer computation times may be applied to a small-size training set obtained from a large number of observations by means of random sampling schemes. It is not uncommon to obtain more accurate classification rules in this way.

Robustness. A classification method is *robust* if the classification rules generated, as well as the corresponding accuracy, do not vary significantly as the choice of the training set and the test set varies, and if it is able to handle missing data and outliers.

Scalability. The scalability of a classifier refers to its ability to learn from large datasets, and it is inevitably related to its computation speed. Therefore, the remarks made in connection with sampling techniques for data reduction, which often result in rules having better generalization capability, also apply in this case.

Interpretability. If the aim of a classification analysis is to interpret as well as predict, then the rules generated should be simple and easily understood by knowledge workers and experts in the application domain.

10.2.1 Holdout method

The *holdout* estimation method involves subdividing the m observations available into two disjoint subsets $\mathcal{T}$ and $\mathcal{V}$, for training and testing purposes respectively, and then evaluating the accuracy of the model through the accuracy $\text{acc}_A(\mathcal{V})$ on the test set. In general, $\mathcal{T}$ is obtained through a simple sampling procedure, which randomly extracts t observations from $\mathcal{D}$, leaving the remaining examples for the test set $\mathcal{V}$. The portion of data used for training may vary based on the size of the dataset $\mathcal{D}$. Usually the value t falls somewhere between one half and two thirds of the total number m of observations.

The accuracy of a classification algorithm evaluated via the holdout method depends on the test set $\mathcal{V}$ selected, and therefore it may over- or underestimate the actual accuracy of the classifier as $\mathcal{V}$ varies. To obtain a more robust estimate, and consequently to evaluate the accuracy of the classification model with greater reliability, different alternative strategies can be followed as described in the following sections.

10.2.2 Repeated random sampling

The *repeated random sampling* method involves replicating the holdout method a number r of times. For each repetition a random independent sample $\mathcal{T}_k$ is extracted, which includes t observations, and the corresponding accuracy $\text{acc}_A(\mathcal{V}_k)$ is evaluated, where $\mathcal{V}_k = \mathcal{D} - \mathcal{T}_k$. At the end of the procedure, the accuracy of the classifier $A_{\mathcal{F}}$ is estimated using the sample mean

$$\text{acc}_A = \text{acc}_{A_{\mathcal{F}}} = \frac{1}{r} \sum_{k=1}^{r} \text{acc}_{A_{\mathcal{F}}}(\mathcal{V}_k). \tag{10.4}$$

The total number r of repetitions may be evaluated a priori using techniques from statistical inference for determining the appropriate sample size.

The method of repeated random sampling is preferred over the holdout method since it yields a more reliable estimate. However, there is no control over the number of times that each observation may appear in the training or in the test set. In this way, a dominant observation containing outliers for one or more attributes may cause undesired effects on both the classification rules generated and the resulting accuracy estimate.

10.2.3 Cross-validation

The method of *cross-validation* offers an alternative to repeated random sampling techniques and guarantees that each observation of the dataset $\mathcal{D}$ appears the same number of times in the training sets and exactly once in the test sets.

The cross-validation scheme is based on a partition of the dataset $\mathcal{D}$ into r disjoint subsets $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_r$, and requires r iterations. At the kth iteration of the procedure, subset $\mathcal{L}_k$ is selected as the test set and the union of all the other subsets in the partition as the training set, that is,

$$\mathcal{V}_k = \mathcal{L}_k, \qquad \mathcal{T}_k = \bigcup_{l \neq k} \mathcal{L}_l. \tag{10.5}$$

Once the partition has been generated, by randomly extracting the observations for each set $\mathcal{L}_k$, the classification algorithm $A_{\mathcal{F}}$ is applied r times, using each of the r training sets in turn and evaluating the accuracy each time on the corresponding test set. At the end of the procedure, the overall accuracy is computed as the arithmetic mean of the r individual accuracies, as in expression (10.4).

If $r = 2$, each observation is used precisely once for training and once for evaluating the accuracy on the test set. However, higher values of r are usually preferred in order to obtain a more robust estimate of the accuracy. A popular choice in practice is *tenfold cross-validation*, whereby the dataset $\mathcal{D}$ is partitioned into 10 subsets. Figure 10.3 illustrates the tenfold cross-validation scheme.

A different evaluation method, called *leave-one-out*, is obtained by choosing $r = m$. In this case each of the m test sets includes only one observation, and each example is used in turn to measure the accuracy. By virtue of a greater

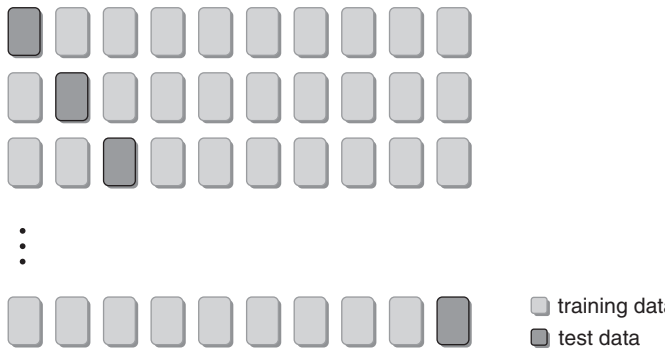

Figure 10.3 Evaluation of accuracy by means of tenfold cross-validation

computational effort, due to m executions of the classifier, the leave-one-out method affords the advantage of carrying out the training operation with a larger number of observations, equal to $m - 1$ for each iteration.

It is usually required that each subset $\mathcal{L}_k$ in the partition contains the same proportion of observations belonging to each target class. If for example the binary target class indicates by the value {1} that a customer has churned, it is appropriate that each subset $\mathcal{L}_k$ contain the same proportion of observations having the target class value {1} and that this proportion be the same as for the full dataset $\mathcal{D}$. In order to preserve in the partition the proportions originally contained in $\mathcal{D}$, stratified random sampling is used to generate the subsets $\mathcal{L}_k$.

10.2.4 Confusion matrices

The accuracy measurement methods described above are not always adequate for discriminating among models, and in some instances they may even yield paradoxical results, as shown in the following two examples.

Example 10.3 – Early medical diagnosis. Consider a binary classification problem in the field of medical diagnosis where observations referred to patients may belong to a binary target class: the value {1} means that a patient has a given medical condition, whereas the value {−1} indicates that a patient does not have the condition. The explanatory variables express the results of clinical tests, and the objective of the model is to classify the health status of patients based on test results, in order to obtain an early diagnosis of the condition to which the examples refer. Assume that the patients who have the condition represent 2% of the observations available in the whole dataset $\mathcal{D}$. The accuracy of the trivial rule claiming that 'the patient does not have the condition' is equal to 0.98, irrespective of test results, and a more sophisticated classifier could hardly achieve a better result. Here, the 2% set made up of observations corresponding to erroneous predictions precisely coincides with those patients who have the condition, and therefore the above-mentioned rule fails to meet the real diagnostic objective, even though it seems highly accurate.

Example 10.4 – Customer retention. In classification applications aiming at customer retention, the value {1} for the target class means that a customer has canceled the service or discontinued the purchase of

goods from the enterprise, while the value {0} indicates that a customer is still active. The explanatory variables refer to transactions recording purchases or usage of the service, and the objective of the model is to predict the future class of customers ahead of time, to allow marketing for retention purposes to be carried out. Assume again that the customers abandoning the service constitute 2% of the available examples in the dataset. The trivial rule claiming that 'all customers are loyal' achieves an accuracy of 0.98, irrespective of the predictive variables, and again it seems hard to beat. Yet, the marketing objective is to identify the highest proportion of customers at risk of churning, which can be found in the 2% of the customer base.

The above examples show that it is not enough to consider the number of accurate predictions, but also the *type* of error committed should be accounted for. For this purpose, it is useful to resort to decision tables, usually called *confusion matrices*, which for the sake of simplicity we will only describe in connection with binary classification, though they can be easily extended to multicategory classification.

Let us assume that we wish to analyze a binary classification problem where the values taken by the target class are $\{-1, 1\}$. We can then consider a 2×2 matrix whose rows correspond to the observed values and whose columns are associated with the values predicted using a classification model, as shown in Table 10.1.[2]

The elements of the confusion matrix have the following meanings: p is the number of correct predictions for the negative examples, called *true negatives*; u is the number of incorrect predictions for the positive examples, called *false negatives*; q is the number of incorrect predictions for the negative examples, called *false positives*; and v is the number of correct predictions for the positive

Table 10.1 Confusion matrix for a binary target attribute encoded with the class values $\{-1, 1\}$

		predictions		
		-1 (negative)	$+1$ (positive)	total
examples	-1 (negative)	p	q	$p+q$
	$+1$ (positive)	u	v	$u+v$
	total	$p+u$	$q+v$	m

[2]Definitions of the confusion matrix can also be found where rows and columns are transposed with respect to our definition.

examples, called *true positives*. Using these elements, further indicators useful for validating a classification algorithm can be defined.

Accuracy. The accuracy of a classifier may be expressed as

$$\text{acc} = \frac{p + v}{p + q + u + v} = \frac{p + v}{m}. \tag{10.6}$$

True negatives rate. The true negatives rate is defined as

$$\text{tn} = \frac{p}{p + q}. \tag{10.7}$$

False negatives rate. The false negatives rate is defined as

$$\text{fn} = \frac{u}{u + v}. \tag{10.8}$$

False positives rate. The false positives rate is defined as

$$\text{fp} = \frac{q}{p + q}. \tag{10.9}$$

True positives rate. The true positives rate, also known as *recall*, is defined as

$$\text{tp} = \frac{v}{u + v}. \tag{10.10}$$

Precision. The precision is the proportion of correctly classified positive examples, and is given by

$$\text{prc} = \frac{v}{q + v}. \tag{10.11}$$

Geometric mean. The geometric mean is defined as

$$\text{gm} = \sqrt{\text{tp} \times \text{prc}}, \tag{10.12}$$

and sometimes also as

$$\text{gm} = \sqrt{\text{tp} \times \text{tn}}. \tag{10.13}$$

In both cases, the geometric mean is equal to 0 if all the predictions are incorrect.

***F*-measure.** The F-measure is defined as

$$F = \frac{(\beta^2 - 1)\,\text{tp} \times \text{prc}}{\beta^2\,\text{prc} + \text{tp}}, \tag{10.14}$$

where $\beta \in [0, \infty)$ regulates the relative importance of the precision with respect to the true positives rate. The F-measure is also equal to 0 if all the predictions are incorrect.

With some classifiers it is possible to assign a matrix of costs for incorrectly classified examples, which we will refer to as *misclassification costs*. Obviously, the cost associated with the $p + v$ correctly classified observations is zero. By adequately regulating the relative weight of the two costs of misclassification corresponding to the u false negatives and to the q false positives, it is possible to direct a classifier toward the actual objectives of the decision makers. In Example 10.3 the cost of false negatives should be much higher than the cost of false positives, so as to identify the maximum number of patients who have the medical condition and apply preventive therapies to them. By the same token, in Example 10.4 the cost of false positives must be much higher than the cost of false negatives, so as to identify the maximum number of customers at risk of churning to whom a preventive retention action can be addressed.

10.2.5 ROC curve charts

Receiver operating characteristic (ROC) curve charts allow the user to visually evaluate the accuracy of a classifier and to compare different classification models. They visually express the information content of a sequence of confusion matrices and allow the ideal trade-off between the number of correctly classified positive observations and the number of incorrectly classified negative observations to be assessed. In this respect, they are an alternative to the assignment of misclassification costs.

An ROC chart is a two-dimensional plot with the proportion of false positives fp on the horizontal axis and the proportion of true positives tp on the vertical axis. The point (0,1) represents the ideal classifier, which makes no prediction error since its proportion of false positives is null ($\text{fp} = 0$) and its proportion of true positives is maximum ($\text{tp} = 1$). The point (0,0) corresponds to a classifier that predicts the class $\{-1\}$ for all the observations, while the point (1,1) corresponds to a classifier predicting the class $\{1\}$ for all the observations.

Most classifiers allow a few parameters in the optimal hypothesis $f^* \in \mathcal{F}$ to be adjusted so as to increase the number of true positives tp, albeit at the expense of a corresponding increase in the number of false positives fp. To obtain the trajectory of the ROC curve for a specific classifier, it is therefore necessary to use pairs of values (fp, tp) which have been empirically obtained for different values of the parameters in $f^* \in \mathcal{F}$. A classifier with no parameters to be tuned yields only one point in the diagram. Figure 10.4 shows an example of ROC curves chart. The area beneath the ROC curve gives a concise measurement comparing the accuracy of various classifiers: the classifier associated with the ROC curve with the greatest area is then preferable.

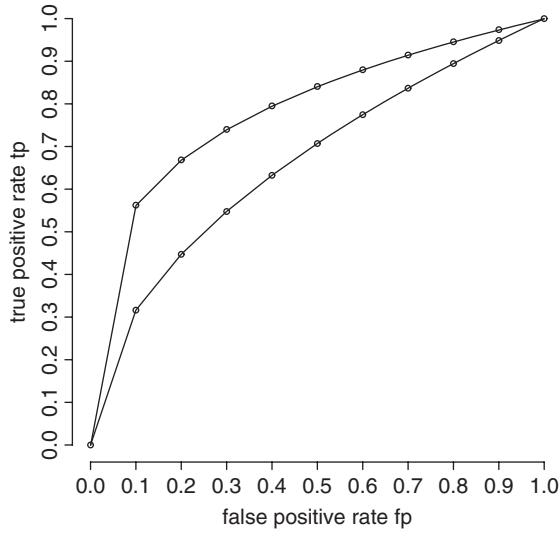

Figure 10.4 ROC curve chart for two classifiers

10.2.6 Cumulative gain and lift charts

The *lift* measure corresponds to the intuitive idea of evaluating the accuracy of a classifier based on the density of positive observations inside the set that has been identified based on model predictions.

To further clarify this point, let $\mathcal{S}$ be a subset of observations of cardinality s selected according to the classifier. For example, the set $\mathcal{S}$ may include those patients deemed potentially at risk of a given medical condition based on the scoring derived by a classification model. Analogously, $\mathcal{S}$ may contain those customers deemed at higher risk of churning for a company analyzing the loyalty of its customers. The lift is defined as the ratio between the proportion b/s of positive observations existing in $\mathcal{S}$ and the proportion a/m of positive observations in the whole dataset $\mathcal{D}$, that is,

$$\text{lift} = \frac{b/s}{a/m}, \tag{10.15}$$

where b and a are the number of positive observations in the sets $\mathcal{S}$ and $\mathcal{D}$, respectively.

Cumulative gain and lift diagrams allow the user to visually evaluate the effectiveness of a classifier. To describe the procedure for constructing the charts and to understand their meaning, consider an example of binary classification arising in a relational marketing analysis, in which we are interested in identifying a subset of customers to be recipients of a cross-selling campaign aimed at promoting a new service.

The company has a total of $m = 1\,000\,000$ customers and, based on similar past campaigns, estimates that the proportion of customers who might respond to the promotion by subscribing to the new service is equal to 2%. A campaign addressed to a random sample of s customers, where $s \in [0, 1\,000\,000]$, would therefore yield a number of positive responses equal to approximately $0.02s$, as an effect of the law of large numbers. The procedure for randomly selecting the recipients is clearly a benchmark to assess the effectiveness of any classifier, which must necessarily be more accurate than random extraction to be of any practical use.

The cumulative gain chart in Figure 10.5 shows on the horizontal axis the number of customers receiving the promotion and on the vertical the percentage of expected positive responses. The straight line in the diagram corresponds to the procedure for the selection of recipients based on random sampling.

If we use a classifier associated with a score function, the customers can be ranked by decreasing score, so that the first customers in the list have a greater probability of subscribing to the offer according to the prediction provided by the classifier. For each value s, consider the set $\mathcal{S}$ consisting of the first s customers of the list ordered by decreasing scores, correspondingly placing the percentage of positive responses along the vertical axis. Usually, the curve is obtained by considering the deciles on the horizontal axis, i.e. dividing into ten equal parts the interval between 0 and the size m of the entire population, represented in our example by the interval $[0, 1\,000\,000]$.

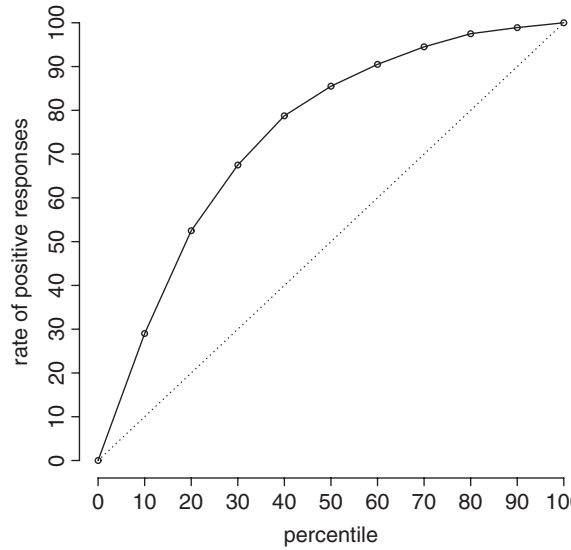

Figure 10.5 Cumulative gain chart

Table 10.2 Number of positive responses at customer base deciles

number of customers contacted	number of positive responses
0	0
100 000	5 800
200 000	10 500
300 000	13 500
400 000	15 900
500 000	17 100
600 000	18 100
700 000	18 900
800 000	19 500
900 000	19 700
1 000 000	20 000

Suppose that we have followed the procedure described above in order to evaluate the predictive effectiveness of a given classifier, obtaining Table 10.2 which shows the number of positive responses existing in subsets $\mathcal{S}$ whose cardinalities equal the deciles. The curve in Figure 10.5 represents the cumulative gain for the classifier.

To derive the corresponding lift curve we calculate for each value available on the horizontal axis, expressing the deciles in our example, the ratio between the proportion of positive responses obtained according to the scoring generated by the classifier and the proportion of positive responses obtained from random sampling. For each value on the horizontal axis, the lift curve is therefore obtained as the ratio between the two values on the cumulative gain curves. Figure 10.6 shows the lift curve for our example.

Two criteria based on cumulative gain and lift curves allow different classifiers to be compared. On the one hand, the area between the cumulative gain curve and the line corresponding to random sampling is evaluated: a greater area corresponds to a classification method that is more effective overall. However, it may be the case that a method producing a greater area is associated with a lower lift value than a different method at a specific value s on the horizontal axis that indicates the actual number of recipients of the marketing campaign, determined according to the available budget. The second criterion is therefore based on the maximum lift value at a specific value s on the horizontal axis.

10.3 Classification trees

Classification trees are perhaps the best-known and most widely used learning methods in data mining applications. The reasons for their popularity lie in their conceptual simplicity, ease of usage, computational speed, robustness with

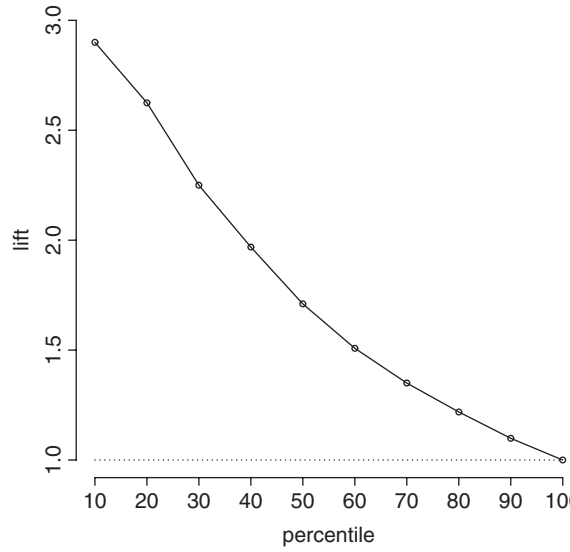

Figure 10.6 Lift chart

respect to missing data and outliers and, most of all, the interpretability of the rules they generate. To separate the observations belonging to different classes, methods based on trees obtain simple and explanatory rules for the relationship existing between the target variable and predictive variables.

The development of a classification tree corresponds to the training phase of the model and is regulated by a recursive procedure of heuristic nature, based on a divide-and-conquer partitioning scheme referred to as *top-down induction of decision trees*, described by Procedure 10.1. Some of the mechanisms governing the development of a tree may be implemented by following different approaches, so that classification trees actually represent a broad class of methods that we will first describe in general terms and then illustrate in detail in some specific cases.

Procedure 10.1 – Top-down induction of decision trees

1. In the initialization phase, each observation is placed in the root node of the tree. The root is included in the list L of active nodes.
2. If the list L is empty the procedure is stopped, otherwise a node J belonging to the list L is selected, is removed from the list and is used as the node for analysis.

3. The optimal rule to split the observations contained in J is then determined, based on an appropriate preset criterion. The splitting rule generated in this way is then applied, and descendant nodes are constructed by subdividing the observations contained in J. For each descendant node the conditions for stopping the subdivision are verified. If these are met, node J becomes a leaf, to which the target class is assigned according to the majority of the observations contained in J. Otherwise, the descendant nodes are added to the list L. Finally, step 2 is repeated.

The observations of the training set initially contained in the *root node* of the tree are divided into disjoint subsets that are tentatively placed in two or more descendant nodes (*branching*). At each of the nodes determined in this way, a check is applied to verify if the conditions for stopping the development of the node are satisfied. If at least one of these conditions is met, no further subdivision is performed and the node becomes a *leaf* of the tree. Otherwise, the actual subdivision of the observations contained in the node is carried out. At the end of the procedure, when no tree node can be further subdivided, each leaf node is labeled with the value of the class to which the majority of the observations in the node belong, according to a criterion called *majority voting*.

The subdivision of the examples in each node is carried out by means of a *splitting rule*, also termed a *separating rule*, to be selected based upon a specific evaluation function. By varying the metrics used to identify the splitting rule, different versions of classification trees can be obtained. Most of the proposed evaluation criteria share the objective of maximizing the uniformity of the target class for the observations that are placed in each node generated through the separation. The different splitting rules are usually based on the value of some explanatory variables that describe the observations, according to the specific methods that will be discussed in the next section.

At the end of the procedure, the set of splitting rules that can be found along the path connecting the tree root to a leaf node constitutes a *classification rule*.[3]

During the prediction phase, in order to assign the target class to a new observation, a path is followed from the root node to a leaf node by obeying the sequence of rules applied to the values of the attributes of the new observation. The predicted target class then coincides with the class by which the leaf node so reached has been labeled during the development phase, that is, with the class of the majority of the observations in the training set falling in that leaf node.

[3]A characteristic property of trees guarantees that there is one and only one path connecting the root node to each other node, in particular to each leaf node.

We observed in Section 10.1 that most classifiers associate with each observation a score function, which is then converted into a prediction of the target class. This is also true for classification trees, which associate each observation contained in a leaf node with the highest proportion of the target class for the observations contained in the leaf itself, which also determines its labeling by majority voting. For example, if in a leaf node there are 100 customers of a company and 85 of these have positively responded in the past to a marketing campaign, then the value 85% can be interpreted as the probability that a customer falling in that leaf node, based on the values of its attributes and the generated classification rules, positively responds to a similar campaign to be conducted in the future.

Starting from a training dataset it is possible to construct an exponential number of distinct classification trees. It can be shown that the problem of determining the optimal tree is NP-hard, that is, computationally difficult. As a consequence, the methods for developing classification trees are heuristic in nature.

As observed, the scheme for generating classification trees described in Procedure 10.1 is a general framework and requires that some steps be specified before deriving an implementable classification algorithm. In the following sections we will examine the following components of the top-down induction of decision trees procedure.

Splitting rules. For each node of the tree it is necessary to specify the criteria used to identify the optimal rule for splitting the observations and for creating the descendant nodes. As shown in the next section, there are several alternative criteria, which differ in the number of descendants, the number of attributes and the evaluation metrics.

Stopping criteria. At each node of the tree different *stopping* criteria are applied to establish whether the development should be continued recursively or the node should be considered as a leaf. In this case too, various criteria have been proposed, which result in quite different topologies of the generated trees, all other elements being equal.

Pruning criteria. Finally, it is appropriate to apply a few *pruning* criteria, first to avoid excessive growth of the tree during the development phase (*pre-pruning*), and then to reduce the number of nodes after the tree has been generated (*post-pruning*).

Figure 10.7 shows a classification tree obtained for the dataset described in Example 5.2, concerning the analysis of loyalty in the mobile phone industry, which will be analyzed in detail in Section 10.3.3.

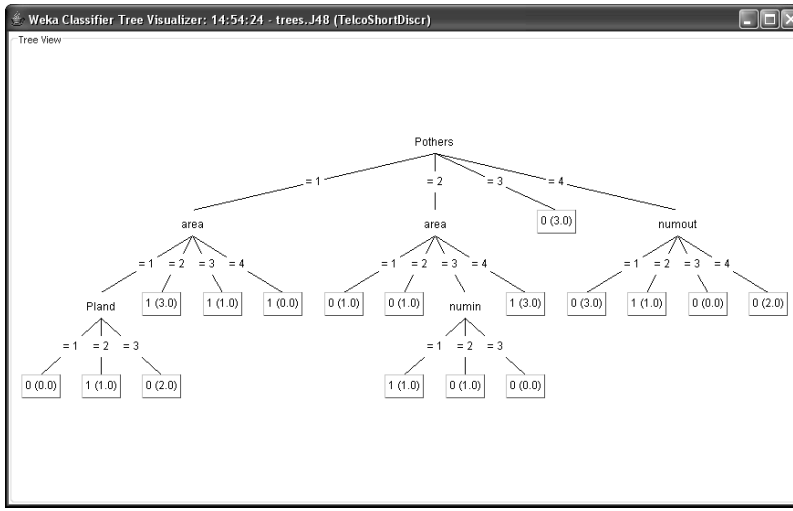

Figure 10.7 Example of a classification tree

10.3.1 Splitting rules

Classification trees can be divided into *binary* and *general* trees based on the maximum number of descendants that each node is allowed to generate.

Binary trees. A tree is said to be *binary* if each node has at most two branches. Binary trees represent in a natural way the subdivision of the observations contained at a node based on the value of a binary explanatory attribute. For example, the customers who have authorized mailing of promotional communications may be placed in the right descendant node and customers who have refused such communications in the left node. When dealing with categorical attributes with more than two classes, binary trees should necessarily form two groups of categories in order to perform a split. For example, the customers residing in areas $\{1, 2\}$ may be placed in the right branch and those residing in areas $\{3, 4\}$ in the left branch. Numerical attributes can be separated based on a threshold value. For example, customers whose age is less than 45 may be placed in the right node and older customers in the left one. Finally, binary trees may also be used to develop multicategory classification.

Multi-split classification trees. A tree is said to be *multi-split* if each node has an arbitrary number of branches. This allows multi-valued categorical explanatory attributes to be handled more easily. On the other hand, with numerical attributes, it is again necessary to group together adjacent values. The latter operation is basically equivalent to discretization, obtained in a dynamic way by the algorithm itself during the tree development phase.

Based on the empirical evidence, no significant differences seem to emerge in the predictive accuracy of classification trees in connection with the maximum number of descendant nodes.

Another relevant distinction within classification tree methods concerns the way the explanatory attributes contribute to the definition of the splitting rule at each node. In particular, we may distinguish between *univariate* and *multivariate* trees.

Univariate trees. For *univariate* trees the splitting rule is based on the value assumed by a single explanatory attribute X_j. If the selected attribute is categorical, the observations at a given node are divided by means of conditions of the form $X_j \in B_k$, where the collection $\{B_k\}$ is composed of disjoint and exhaustive subsets of the set of values assumed by the attribute X_j. For example, for a binary attribute taking the values {0, 1}, the two subsets B_0 and B_1 correspond to the values {0} and {1}, as shown in Figure 10.8. Figure 10.9 shows the partition for a non-binary categorical attribute. If X_j is a numerical attribute, the univariate partition consists of a rule in the form $X_j \leq b$ or $X_j > b$, as shown in Figure 10.10 for the age of a customer, depending on a threshold value b determined by the algorithm itself. Univariate trees are also referred to as *axis-parallel* trees, since the splitting rules induce a partition of the space of

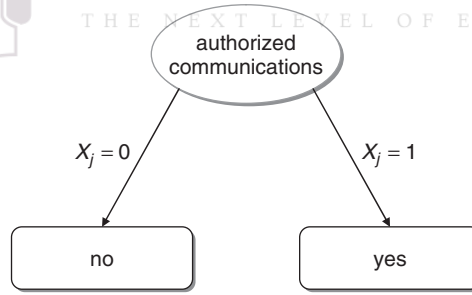

Figure 10.8 Univariate split for a binary attribute

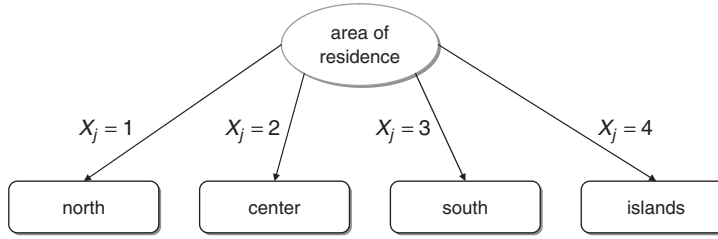

Figure 10.9 Univariate split for a nominal attribute

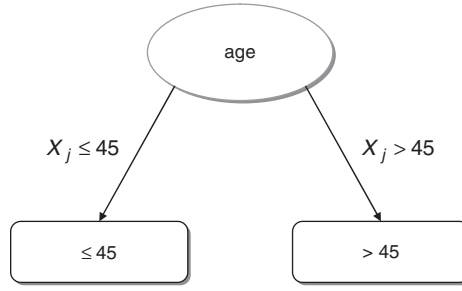

Figure 10.10 Univariate split for a numerical attribute

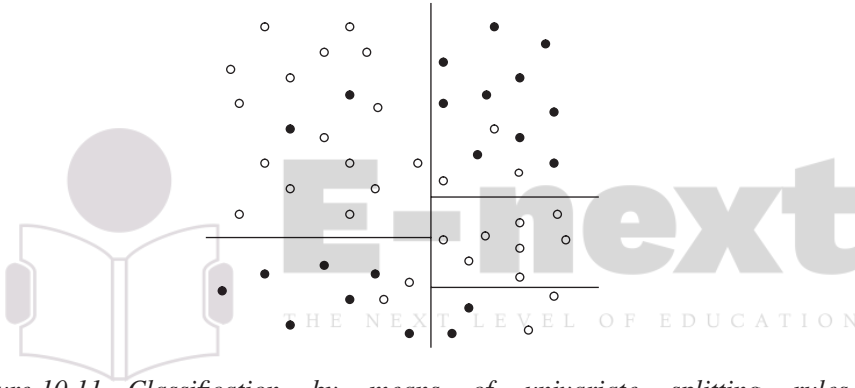

Figure 10.11 Classification by means of univariate splitting rules (axis-parallel). Each line corresponds to a splitting rule generated at a node in the development of the tree

the observations into hyper-rectangles, determined by the intersection of half-spaces whose support hyperplanes are parallel to the components of the vector of instances, as shown in Figure 10.11.

Multivariate trees. For multivariate trees, the partition of the observations at a given node is based on the value assumed by a function $\varphi(x_1, x_2, \ldots, x_n)$ of the attributes and leads to a rule of the form $\varphi(\mathbf{x}) \leq b$ or $\varphi(\mathbf{x}) > b$. Different methods have been proposed whereby the function φ represents a linear combination of the explanatory variables. In this case, the expression that leads to the separation of the observations takes the form

$$\sum_{j=1}^{n} w_j x_j \leq b, \tag{10.16}$$

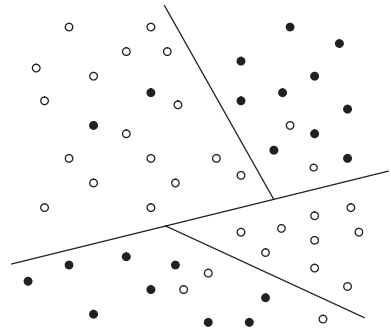

Figure 10.12 Classification by means of multivariate splitting rules (oblique). Each straight line corresponds to a splitting rule generated at a node in the development of the tree

where the threshold value b and the coefficients $w_1, w_2, \ldots, w_n$ of the linear combination have to be determined, for example by solving an optimization problem for each node, just like for classification trees generated by means of discrete variants of support vector machines. Multivariate trees are also referred to as *oblique decision trees*, as they generate polygonal partitions of the space of the observations by means of separating hyperplanes, as shown in Figure 10.12.

Oblique trees are usually characterized by greater predictive accuracy than univariate trees, against a lower interpretability of the classification rules generated. In many cases, a limited number of separating hyperplanes may be enough to classify with high accuracy the instances, whereas to achieve the same result an axis-parallel tree would require the partition of the space of the observations in several hyper-rectangles. Figure 10.13 shows an example of a two-dimensional dataset for which the two target classes may be easily separated by a single oblique line, while they would require a high number of univariate rules.

Notice also that the number of regions generated is a function of the number of leaves in the tree, and therefore of its depth. Accuracy being equal, oblique trees usually generate a lower number of classification rules with respect to univariate trees.

10.3.2 Univariate splitting criteria

Although they are usually characterized by a lower accuracy, the algorithms that develop classification trees based on univariate rules are more popular than their multivariate counterpart, partly because of the simplicity and interpretability of the rules generated and partly because they were proposed first.

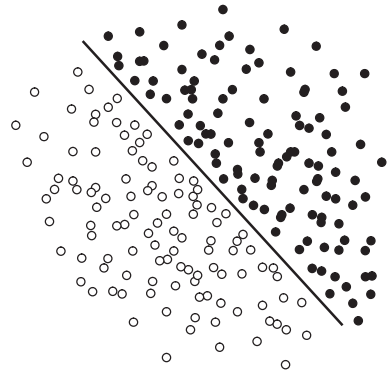

Figure 10.13 Classification by means of a single multivariate (oblique) splitting rule for a dataset that cannot be accurately split by univariate (axis-parallel) rules

The main component that differentiates the variants of univariate classification trees proposed so far is the splitting rule, used to identify the best explanatory attribute out of those available and to select the most effective partitioning criterion among the ones it induces. Usually, both choices are performed by calculating an *evaluation function*, for each attribute and for each possible partition, which provides a heterogeneity measure in the values of the target class between the examples belonging to the parent node and those belonging to the descendants. The maximization of the evaluation function therefore identifies the partition that generates descendant nodes that are more homogeneous within themselves than the parent node is.

Let p_h be the proportion of examples of target class v_h, $h \in \mathcal{H}$, at a given node q and let Q be the total number of instances at q. We have

$$\sum_{h=1}^{H} p_h = 1. \tag{10.17}$$

The heterogeneity index $I(q)$ of a node is usually a function of the relative frequencies p_h, $h \in \mathcal{H}$, of the target class values for the examples at the node, and it has to satisfy three requirements: it must take its maximum value when the examples at the node are distributed homogeneously among all the classes; it must take its minimum value when all the instances at the node belong to the same class; and it must be a symmetric function with respect to the relative frequencies p_h, $h \in \mathcal{H}$.

Among the heterogeneity indices of a node q that satisfy these properties, also referred to as *impurity* or *inhomogeneity* measures, the most popular are the *misclassification index*, the *entropy index* and the *Gini index*.

Misclassification index. The misclassification index is defined as

$$\text{Miscl}(q) = 1 - \max_h p_h, \tag{10.18}$$

and measures the proportion of misclassified examples when all the instances at node q are assigned to the class to which the majority of them belong, according to the majority voting principle.

Entropy index. The entropy is defined as

$$\text{Entropy}(q) = -\sum_{h=1}^{H} p_h \log_2 p_h; \tag{10.19}$$

note that, by convention, $0 \log_2 0 = 0$.

Gini index. The Gini index is defined as

$$\text{Gini}(q) = 1 - \sum_{h=1}^{H} p_h^2. \tag{10.20}$$

In a binary classification problem the impurity measures defined above reach their maximum value when $p_1 = p_2 = 1 - p_1 = 0.5$, and are 0 when $p_1 = 0$ or $p_1 = 1$, as shown in Figure 10.14.

The univariate splitting criteria based on the *information gain* compare one of the impurity indices evaluated for the parent node with the same index

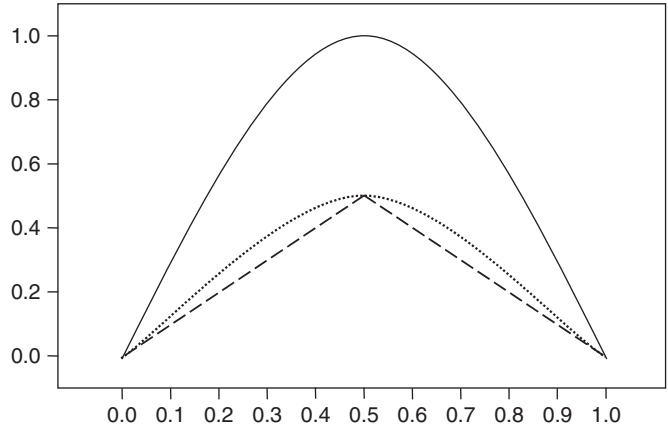

Figure 10.14 Graph of the misclassification index (dashed line), the Gini index (dotted line) and the entropy (full line) for a binary target attribute as the frequency of the examples in one class varies

computed for the set of descendant nodes, and then choose the attribute and the corresponding partition that maximize the difference. Since the value of the impurity index for the parent node is independent of the partition, the choice may be equivalently done by minimizing the overall impurity index for the descendant nodes.

Let $I(\cdot)$ be one of the impurity indices previously defined, and suppose that a splitting rule separates the examples contained at node q into K descendant nodes $\{q_1, q_2, \ldots, q_K\}$, each containing Q_k instances. If we analyze the partition originated by a categorical attribute X_j taking H_j distinct values, the set of examples at q can be separated into H_j disjoint subsets, as shown in Figure 10.9. In this way we have $K = H_j$, and the descendant node q_j contains the examples for which the explanatory variable X_j takes the value v_j. If $H_j > 2$, this partition is possible only if the tree is of general type. If a binary tree is developed, it is necessary to divide the H_j values into two non-empty sets and then compute the heterogeneity indices for all 2^{H_j-1} possible partitions. Finally, if the attribute X_j is numerical, the examples can be subdivided by intervals of values, as shown in Figure 10.10. To avoid the analysis of all possible threshold values for the separation, the algorithm operates a binary search among the values actually taken by the attribute X_j in the dataset $\mathcal{D}$.

The impurity of the descendant nodes, and therefore the impurity of the splitting rule, is defined as

$$I(q_1, q_2, \ldots, q_K) = \sum_{k=1}^{K} \frac{Q_k}{Q} I(q_k). \tag{10.21}$$

Hence, the impurity of the partition is expressed by a weighted sum of the impurities of each descendant node, where each weight equals the percentage of examples from the parent node that are placed in the corresponding descendant.

The algorithms for the development of univariate trees select for each node the rule and the corresponding attribute that determine the minimum value of expression (10.21). This choice is equivalent to maximizing the information gain $\Delta(\cdot)$, defined as

$$\begin{aligned} \Delta(q, q_1, q_2, \ldots, q_K) &= I(q) - I(q_1, q_2, \ldots, q_K) \\ &= I(q) - \sum_{k=1}^{K} \frac{Q_k}{Q} I(q_k). \end{aligned} \tag{10.22}$$

10.3.3 Example of development of a classification tree

In this section we will describe the development of two classification trees obtained by using respectively the entropy and the Gini index for the small

dataset from Example 5.2 comprising 23 observations, drawn as part of a retention analysis in the mobile phone industry.

Discretization of the dataset

Although it would be possible to apply the two classifiers to datasets containing numerical attributes, in order to make the description of the two algorithms easier it is appropriate to perform a discretization of the numerical predictive variables, as a preliminary step to the development of the classification trees.

The numerical attributes in the dataset have been discretized by subdividing the examples into equally wide classes based on the ranges of values. Table 10.3 shows the classes obtained, and Table 10.4 shows the new dataset consisting solely of categorical variables, obtained at the end of the discretization step.

Entropy index

By referring to Table 10.4 and using the definition of the entropy index in (10.19), it is possible to compute the amount of information required to establish if a generic example belongs to class {0} or to class {1} corresponding to the root node for the entire dataset. Indeed, the entropy of the root q is equal to

$$I_E(q) = \text{Entropy}(q) = -\frac{13}{23}\log_2\frac{13}{23} - \frac{10}{23}\log_2\frac{10}{23} = 0.988.$$

By splitting on the attribute *area*, the root node would be subdivided into four child nodes $\{q_1, q_2, q_3, q_4\}$ that correspond to the values $\{1, 2, 3, 4\}$, as shown in Figure 10.10. Hence, associated with the four descendant nodes are the following proportions of membership in the two target classes {0, 1}:

$$p_0(q_1) = 5/6, \quad p_0(q_2) = 2/5, \quad p_0(q_3) = 5/7, \quad p_0(q_4) = 1/5, \quad (10.23)$$

$$p_1(q_1) = 1/6, \quad p_1(q_2) = 3/5, \quad p_1(q_3) = 2/7, \quad p_1(q_4) = 4/5. \quad (10.24)$$

Table 10.3 Subdivision into classes for the discretization of the numerical attributes in Example 5.2

attribute	class 1	class 2	class 3	class 4
numin	[0,20)	[20,40)	[40, ∞)	
timein	[0,10 000)	[10 000,20 000)	[20 000,30 000)	[30 000, ∞)
numout	[0,30)	[30,60)	[60,90)	[90, ∞)
Pothers	[0,0.1)	[0.1,0.2)	[0.2,0.3)	[0.3, ∞)
Pmob	[0,0.2)	[0.2,0.4)	[0.4,0.6)	[0.6, ∞)
Pland	[0,0.25)	[0.25,0.5)	[0.5, ∞)	
numsms	[0,1)	[1,10)	[10,20)	[20, ∞)
numserv	[0,1)	[1,2)	[2,3)	[3, ∞)
numcall	[0,1)	[1,3)	[3, ∞)	

Table 10.4 Discretized input data for Example 5.2

area	numin	timein	numout	Pothers	Pmob	Pland	numsms	numserv	numcall	dropt	churner
2	1	1	2	1	4	1	3	2	2	0	1
1	1	3	3	2	4	1	4	2	3	0	0
3	2	1	2	2	4	1	3	2	1	0	0
1	2	3	2	3	4	1	1	2	1	0	0
2	3	4	4	4	1	1	3	2	1	0	0
3	3	4	1	4	2	1	4	3	1	0	0
3	3	3	4	4	3	1	4	3	1	1	0
1	1	1	1	1	3	2	1	1	1	0	1
2	2	2	2	1	3	2	2	3	1	1	1
4	2	1	3	2	3	2	1	2	1	1	1
3	1	1	2	2	2	2	2	2	1	0	1
4	3	4	4	2	1	2	2	4	1	1	1
2	1	1	3	2	3	2	2	4	1	1	0
4	2	1	2	3	2	2	2	2	1	0	0
3	3	4	4	3	2	2	2	4	1	1	0
1	1	2	1	4	2	2	2	2	1	0	0
4	1	1	2	4	2	2	4	2	1	0	1
1	1	1	1	1	1	3	1	1	1	0	0
3	1	1	1	1	1	3	1	1	1	0	1
2	3	4	3	1	1	3	1	1	1	0	1
1	3	3	3	1	2	3	4	2	1	0	0
4	2	2	2	2	2	3	1	1	1	0	1
3	3	2	1	4	1	3	1	1	1	0	0

The entropy index, which measures the heterogeneity of the four descendant nodes, is given by

$$I_E(q_1, q_2, q_3, q_4) = \frac{6}{23} I_E(q_1) + \frac{5}{23} I_E(q_2) + \frac{7}{23} I_E(q_3) + \frac{5}{23} I_E(q_4)$$
$$= \frac{6}{23} 0.650 + \frac{5}{23} 0.971 + \frac{7}{23} 0.863 + \frac{5}{23} 0.722 = 0.8.$$

Therefore, the gain achieved by separating the examples through the attribute *area* is

$$\Delta_E(area) = \Delta_E(q, q_1, q_2, q_3, q_4) = I_E(q) - I_E(q_1, q_2, q_3, q_4)$$
$$= 0.988 - 0.8 = 0.188.$$

Similarly, it is possible to compute the gain associated with the separation performed on the remaining attributes of the dataset:

$$\begin{aligned}
&\Delta_E(numin) = 0.057, \quad && \Delta_E(Pland) = 0.125,\\
&\Delta_E(timein) = 0.181, \quad && \Delta_E(numsms) = 0.080,\\
&\Delta_E(numout) = 0.065, \quad && \Delta_E(numserv) = 0.057,\\
&\Delta_E(Pothers) = 0.256, \quad && \Delta_E(numcall) = 0.089,\\
&\Delta_E(Pmob) = 0.043, \quad && \Delta_E(dirop) = 0.005.
\end{aligned}$$

The maximum information gain is achieved when the examples are separated by means of the attribute *Pothers*, which measures the proportion of calls placed to other mobile phone companies. By recursively performing the subsequent iterations in an analogous way, we obtain the classification tree depicted in Figure 10.7.

Gini index

The Gini index evaluated for the root q, corresponding to the entire dataset described in Table 10.4, is equal to

$$I_G(q) = \text{Gini}(q) = 1 - \left(\frac{13}{23}\right)^2 - \left(\frac{10}{23}\right)^2 = 0.491.$$

As outlined above, the attribute *area* separates the examples at the root node q into four descendant nodes $\{q_1, q_2, q_3, q_4\}$ corresponding to the values {1, 2, 3, 4}, and associated with the proportions of membership in the two target classes {0, 1} shown in (10.23) and (10.24).

The Gini index, which measures the heterogeneity of the four descendant nodes, is equal to

$$I_G(q_1, q_2, q_3, q_4) = \frac{6}{23} I_G(q_1) + \frac{5}{23} I_G(q_2) + \frac{7}{23} I_G(q_3) + \frac{5}{23} I_G(q_4)$$
$$= \frac{6}{23} 0.278 + \frac{5}{23} 0.480 + \frac{7}{23} 0.408 + \frac{5}{23} 0.528 = 0.320.$$

Therefore, the gain achieved by separating the examples through the attribute *area* is

$$\Delta_G(area) = \Delta_G(q, q_1, q_2, q_3, q_4) = I_G(q) - I_G(q_1, q_2, q_3, q_4)$$
$$= 0.491 - 0.370 = 0.121.$$

Similarly, it is possible to compute the gain associated with the separation performed on the remaining attributes of the dataset:

$$\begin{array}{ll} \Delta_G(numin) = 0.037, & \Delta_G(Pland) = 0.078, \\ \Delta_G(timein) = 0.091, & \Delta_G(numsms) = 0.052, \\ \Delta_G(numout) = 0.043, & \Delta_G(numserv) = 0.038, \\ \Delta_G(Pothers) = 0.146, & \Delta_G(numcall) = 0.044, \\ \Delta_G(Pmob) = 0.028, & \Delta_G(diropt) = 0.003. \end{array}$$

As for the entropy index, the examples at the root of the tree are best separated by means of the attribute *Pothers*, for which the increase in homogeneity is maximum.

10.3.4 Stopping criteria and pruning rules

Stopping criteria are a set of rules used at each node during the development of a tree in order to determine whether it is appropriate to create more branches and generate descendant nodes or whether the current node should become a leaf. There are two main reasons to limit the growth of a classification tree. First, a tree with too many ramifications usually achieves better accuracy with the training set but causes larger errors when used to make predictions on the test set or on future data. From an intuitive point of view, one may think that a tree with many branches excessively reflects the peculiarity of the examples in the training set and is therefore less capable of generalization. This phenomenon, usually referred to as *overfitting*, can be well explained from a theoretical perspective within the framework of statistical learning theory: a tree with too many ramifications actually represents a broader space of hypotheses, and therefore reduces the empirical error for the training set, but at the same time increases the generalization error.

Furthermore, a more ramified tree implies a proliferation of leaves and hence generates deep classification rules, obtained by combining a large number of splitting rules along the path leading from the root node to a leaf. This reduces the overall interpretability of the resulting classification model.

In principle, node splitting might be stopped when there is only one observation in the node. The stopping criteria followed in practice prevent this situation from happening, by introducing some restrictions on the minimum number of observations that a node must contain to be partitioned. Additionally, it can be

assigned a minimum threshold on the uniformity, sometimes called *purity*, of the target class proportion within a node to be split.

More precisely, a node becomes a leaf of the tree when at least one of the following conditions occurs.

Node size. The node contains a number of observations that is below a preset minimum threshold value.

Purity. The proportion of observations at the node and belonging to the same class is above a preset maximum threshold value that corresponds to the accuracy that one wishes to achieve.

Improvement. The possible subdivision of the node would generate a gain $\Delta(\cdot)$ that is below a preset minimum threshold value.

The stopping criteria described above are also referred to as *pre-pruning* rules because their aim is to prune a priori the tree by limiting its growth. However, there are other *post-pruning*, or simply *pruning*, techniques which are applied upon completion of tree development to reduce the number of ramifications without worsening, and hopefully even improving, the predictive accuracy of the resulting model. At each iteration during the post-pruning phase, the possible advantage deriving from the removal of a given branch is evaluated, by comparing the predictive accuracy of the original tree with that of the reduced tree in classifying the observations of the tuning set. At the end of the evaluation, the reduced tree associated with the minimum prediction error is selected.

10.4 Bayesian methods

Bayesian methods belong to the family of probabilistic classification models. They explicitly calculate the *posterior* probability $P(y|\mathbf{x})$ that a given observation belongs to a specific target class by means of Bayes' theorem, once the *prior* probability $P(y)$ and the class conditional probabilities $P(\mathbf{x}|y)$ are known.

Unlike other methods described in this chapter, which are not based on probabilistic assumptions, Bayesian classifiers require the user to estimate the probability $P(\mathbf{x}|y)$ that a given observation may occur, provided it belongs to a specific class. The learning phase of a Bayesian classifier may therefore be identified with a preliminary analysis of the observations in the training set, to derive an estimate of the probability values required to perform the classification task.

Let us consider a generic observation $\mathbf{x}$ of the training set, whose target variable y may take H distinct values denoted as $\mathcal{H} = \{v_1, v_2, \ldots, v_H\}$.

Bayes' theorem is used to calculate the posterior probability $P(y|\mathbf{x})$, that is, the probability of observing the target class y given the example $\mathbf{x}$:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{\sum_{l=1}^{H} P(\mathbf{x}|y)P(y)} = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \tag{10.25}$$

In order to classify a new instance $\mathbf{x}$, the Bayes classifier applies a principle known as the *maximum a posteriori hypothesis* (MAP), which involves calculating the posterior probability $P(y|\mathbf{x})$ using (10.25) and assigning the example $\mathbf{x}$ to the class that yields the maximum value $P(y|\mathbf{x})$, that is,

$$y_{\text{MAP}} = \arg\max_{y \in \mathcal{H}} P(y|\mathbf{x}) = \arg\max_{y \in \mathcal{H}} \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \tag{10.26}$$

Since the denominator $P(\mathbf{x})$ is independent of y, in order to maximize the posterior probability it is enough to maximize the numerator of (10.26). Therefore, the observation $\mathbf{x}$ is assigned to the class v_h if and only if

$$P(\mathbf{x}|y = v_h)P(y = v_h) \geq P(\mathbf{x}|y = v_l)P(y = v_l), \quad l = 1, 2, \ldots, H.$$

The prior probability $P(y)$ can be estimated using the frequencies m_h with which each value of the target class v_h appears in the dataset $\mathcal{D}$, that is,

$$P(y = v_h) = \frac{m_h}{m}. \tag{10.27}$$

Given a sufficiently large sample the estimates of the prior probabilities obtained through (10.27) will be quite accurate.

Unfortunately an analogous sample estimate of the class conditional probabilities $P(\mathbf{x}|y)$ cannot be obtained in practice due to the computational complexity and the huge number of sample observations that it would require. To see this, consider the following hypothetical situation: if the dataset includes 50 binary categorical attributes, there would be $2^{50} \approx 10^{15}$ possible combinations of attribute values that constitute the cells of the joint density of $\mathbf{x}$ for which the relative frequencies should be estimated. To obtain a reliable estimate it would be necessary to have at least 10 observations for each combination. Hence, the dataset $\mathcal{D}$ should contain at least 10^{16} observations. If the attributes were nominal categorical, or numerical, this number would grow exponentially fast.

To overcome the computational difficulty described, it is possible to introduce two simplifying hypotheses that lead to *naive Bayesian* classifiers and to *Bayesian networks* respectively, described next.

10.4.1 Naive Bayesian classifiers

Naive Bayesian classifiers are based on the assumption that the explanatory variables are conditionally independent given the target class. This hypothesis

allows us to express the probability $P(\mathbf{x}|y)$ as

$$P(\mathbf{x}|y) = P(x_1|y) \times P(x_2|y) \times \cdots \times P(x_n|y) = \prod_{j=1}^{n} P(x_j|y). \qquad (10.28)$$

The probabilities $P(x_j|y)$, $j \in \mathcal{N}$, can be estimated using the examples from the training set, depending on the nature of the attribute considered.

Categorical or discrete numerical attributes. For a categorical or discrete numerical attribute $\mathbf{a}_j$ which may take the values $\{r_{j1}, r_{j2}, \ldots, r_{jK_j}\}$, the probability $P(x_j|y) = P(x_j = r_{jk}|y = v_h)$ is evaluated as the ratio between the number s_{jhk} of instances of class v_h for which the attribute $\mathbf{a}_j$ takes the value r_{jk}, and the total number m_h of instances of class v_h in the dataset $\mathcal{D}$, that is,

$$P(x_j|y) = P(x_j = r_{jk}|y = v_h) = \frac{s_{jhk}}{m_h}. \qquad (10.29)$$

Numerical attributes. For a numerical attribute $\mathbf{a}_j$, the probability $P(x_j|y)$ is estimated assuming that the examples follow a given distribution. For example, one may consider a Gaussian density function, for which

$$P(x_j|y = v_h) = \frac{1}{\sqrt{2\pi}\sigma_{jh}} e^{-\frac{(x_j - \mu_{jh})^2}{2\sigma_{jh}^2}}, \qquad (10.30)$$

where μ_{jh} and σ_{jh} respectively denote the mean and standard deviation of the variable X_j for the examples of class v_h, and may be estimated on the basis of the examples contained in $\mathcal{D}$.

In spite of the simplifying assumption of conditional independence of the attributes, which makes it easy to compute the conditional probabilities, the empirical evidence shows that Bayesian classifiers are often able to achieve accuracy levels which are not lower than those provided by classification trees or even by more complex classification methods.

10.4.2 Example of naive Bayes classifier

In order to describe the usage of the naive Bayes classifier, consider again the dataset in Table 10.4, obtained by discretization of the numerical attributes in the small dataset comprising the 23 observations of Example 5.2, concerning a retention analysis in the mobile phone industry.

The relative frequencies of the sample attribute values given the target class are as follows:

area

$$P\,(area = 1|\, churner = 0) = \frac{5}{13}, \quad P\,(area = 1|\, churner = 1) = \frac{1}{10},$$
$$P\,(area = 2|\, churner = 0) = \frac{2}{13}, \quad P\,(area = 2|\, churner = 1) = \frac{3}{10},$$
$$P\,(area = 3|\, churner = 0) = \frac{5}{13}, \quad P\,(area = 3|\, churner = 1) = \frac{2}{10},$$
$$P\,(area = 4|\, churner = 0) = \frac{1}{13}, \quad P\,(area = 4|\, churner = 1) = \frac{4}{10}.$$

numin

$$P\,(numin = 1|\, churner = 0) = \frac{4}{13}, \quad P\,(numin = 1|\, churner = 1) = \frac{5}{10},$$
$$P\,(numin = 2|\, churner = 0) = \frac{3}{13}, \quad P\,(numin = 2|\, churner = 1) = \frac{3}{10},$$
$$P\,(numin = 3|\, churner = 0) = \frac{6}{13}, \quad P\,(numin = 3|\, churner = 1) = \frac{2}{10}.$$

timein

$$P\,(timein = 1|\, churner = 0) = \frac{4}{13}, \quad P\,(timein = 1|\, churner = 1) = \frac{6}{10},$$
$$P\,(timein = 2|\, churner = 0) = \frac{2}{13}, \quad P\,(timein = 2|\, churner = 1) = \frac{2}{10},$$
$$P\,(timein = 3|\, churner = 0) = \frac{4}{13}, \quad P\,(timein = 3|\, churner = 1) = 0,$$
$$P\,(timein = 4|\, churner = 0) = \frac{3}{13}, \quad P\,(timein = 4|\, churner = 1) = \frac{2}{10}.$$

numout

$$P\,(numout = 1|\, churner = 0) = \frac{4}{13}, \quad P\,(numout = 1|\, churner = 1) = \frac{2}{10},$$
$$P\,(numout = 2|\, churner = 0) = \frac{3}{13}, \quad P\,(numout = 2|\, churner = 1) = \frac{5}{10},$$
$$P\,(numout = 3|\, churner = 0) = \frac{3}{13}, \quad P\,(numout = 3|\, churner = 1) = \frac{2}{10},$$
$$P\,(numout = 4|\, churner = 0) = \frac{3}{13}, \quad P\,(numout = 4|\, churner = 1) = \frac{1}{10}.$$

Pothers

$$P\,(Pothers = 1|\, churner = 0) = \frac{2}{13}, \quad P\,(Pothers = 1|\, churner = 1) = \frac{5}{10},$$
$$P\,(Pothers = 2|\, churner = 0) = \frac{3}{13}, \quad P\,(Pothers = 2|\, churner = 1) = \frac{4}{10},$$

$$P\,(Pothers = 3|\ churner = 0) = \frac{3}{13}, \quad P\,(Pothers = 3|\ churner = 1) = 0,$$

$$P\,(Pothers = 4|\ churner = 0) = \frac{5}{13}, \quad P\,(Pothers = 4|\ churner = 1) = \frac{1}{10}.$$

Pmob

$$P\,(Pmob = 1|\ churner = 0) = \frac{3}{13}, \quad P\,(Pmob = 1|\ churner = 1) = \frac{3}{10},$$

$$P\,(Pmob = 2|\ churner = 0) = \frac{5}{13}, \quad P\,(Pmob = 2|\ churner = 1) = \frac{3}{10},$$

$$P\,(Pmob = 3|\ churner = 0) = \frac{2}{13}, \quad P\,(Pmob = 3|\ churner = 1) = \frac{3}{10},$$

$$P\,(Pmob = 4|\ churner = 0) = \frac{3}{13}, \quad P\,(Pmob = 4|\ churner = 1) = \frac{1}{10}.$$

Pland

$$P\,(Pland = 1|\ churner = 0) = \frac{6}{13}, \quad P\,(Pland = 1|\ churner = 1) = \frac{1}{10},$$

$$P\,(Pland = 2|\ churner = 0) = \frac{4}{13}, \quad P\,(Pland = 2|\ churner = 1) = \frac{6}{10},$$

$$P\,(Pland = 3|\ churner = 0) = \frac{3}{13}, \quad P\,(Pland = 3|\ churner = 1) = \frac{3}{10}.$$

numsms

$$P\,(numsms = 1|\ churner = 0) = \frac{3}{13}, \quad P\,(numsms = 1|\ churner = 1) = \frac{5}{10},$$

$$P\,(numsms = 2|\ churner = 0) = \frac{4}{13}, \quad P\,(numsms = 2|\ churner = 1) = \frac{3}{10},$$

$$P\,(numsms = 3|\ churner = 0) = \frac{2}{13}, \quad P\,(numsms = 3|\ churner = 1) = \frac{1}{10},$$

$$P\,(numsms = 4|\ churner = 0) = \frac{4}{13}, \quad P\,(numsms = 4|\ churner = 1) = \frac{1}{10}.$$

numserv

$$P\,(numserv = 1|\ churner = 0) = \frac{2}{13}, \quad P\,(numserv = 1|\ churner = 1) = \frac{4}{10},$$

$$P\,(numserv = 2|\ churner = 0) = \frac{7}{13}, \quad P\,(numserv = 2|\ churner = 1) = \frac{4}{10},$$

$$P\,(numserv = 3|\ churner = 0) = \frac{2}{13}, \quad P\,(numserv = 3|\ churner = 1) = \frac{1}{10},$$

$$P\,(numserv = 4|\ churner = 0) = \frac{2}{13}, \quad P\,(numserv = 4|\ churner = 1) = \frac{1}{10}.$$

numcall

$$P\,(numcall = 1|\,churner = 0) = \frac{12}{13}, \quad P\,(numcall = 1|\,churner = 1) = \frac{9}{10},$$
$$P\,(numcall = 2|\,churner = 0) = 0, \quad P\,(numcall = 2|\,churner = 1) = \frac{1}{10},$$
$$P\,(numcall = 3|\,churner = 0) = \frac{1}{13}, \quad P\,(numcall = 3|\,churner = 1) = 0.$$

diropt

$$P\,(diropt = 0|\,churner = 0) = \frac{10}{13}, \quad P\,(diropt = 0|\,churner = 1) = \frac{7}{10},$$
$$P\,(diropt = 1|\,churner = 0) = \frac{3}{13}, \quad P\,(diropt = 1|\,churner = 1) = \frac{3}{10}.$$

Once the conditional probabilities of each attribute given the target class have been estimated, suppose that we wish to predict the target class of a new observation, represented by the vector $\mathbf{x} = (1, 1, 1, 2, 1, 4, 2, 1, 2, 1, 0)$. With this aim in mind, we compute the posterior probabilities $P(\mathbf{x}|0)$ and $P(\mathbf{x}|1)$:

$$P\,(\mathbf{x}|0) = \frac{5}{13}\cdot\frac{4}{13}\cdot\frac{4}{13}\cdot\frac{3}{13}\cdot\frac{2}{13}\cdot\frac{3}{13}\cdot\frac{4}{13}\cdot\frac{3}{13}\cdot\frac{7}{13}\cdot\frac{12}{13}\cdot\frac{10}{13} = 0.81\cdot 10^{-5},$$
$$P\,(\mathbf{x}|1) = \frac{1}{10}\cdot\frac{5}{10}\cdot\frac{6}{10}\cdot\frac{5}{10}\cdot\frac{5}{10}\cdot\frac{1}{10}\cdot\frac{6}{10}\cdot\frac{5}{10}\cdot\frac{4}{10}\cdot\frac{9}{10}\cdot\frac{7}{10} = 5.67\cdot 10^{-5}.$$

Since the relative frequencies of the two classes are given by

$$P\,(churner = 0) = \frac{13}{23} = 0.56, \qquad P\,(churner = 1) = \frac{10}{23} = 0.44,$$

we have

$$\begin{aligned} P(churner = 0|\mathbf{x}) &= P(\mathbf{x}|0)P\,(churner = 0) \\ &= 0.81\cdot 10^{-5}\cdot 0.56 = 0.46\cdot 10^{-5}, \\ P(churner = 1|\mathbf{x}) &= P(\mathbf{x}|1)P\,(churner = 1) \\ &= 5.67\cdot 10^{-5}\cdot 0.44 = 2.495\cdot 10^{-5}. \end{aligned}$$

The new example $\mathbf{x}$ is then labeled with the class value {1}, since this is associated with the maximum a posteriori probability.

10.4.3 Bayesian networks

Bayesian networks, also called *belief networks*, allow the hypothesis of conditional independence of the attributes to be relaxed, by introducing some reticular

hierarchical links through which it is possible to assign selected stochastic dependencies that experts of the application domain deem relevant.

A Bayesian network comprises two main components. The first is an *acyclic oriented graph* in which the nodes correspond to the predictive variables and the arcs indicate relationships of stochastic dependence. In particular, it is assumed that the variable X_j associated with node $\mathbf{a}_j$ in the network is dependent on the variables associated with the predecessor nodes of $\mathbf{a}_j$, and conditionally independent of the variables associated with the nodes that are not directly reachable from $\mathbf{a}_j$.

The second component consists of a table of conditional probabilities assigned for each variable. In particular, the table associated with the variable X_j indicates the conditional distribution of $P(X_j|\mathcal{C}_j)$, where $\mathcal{C}_j$ represents the set of explanatory variables associated with the predecessor nodes of node $\mathbf{a}_j$ in the network and is estimated based on the relative frequencies in the dataset. The complexity required to compute all possible combinations of predictor values for estimating the conditional probabilities, already pointed out above, is limited in Bayesian networks, since the calculation simply reduces to those conditioning relationships determined by the precedence links included in the network. Consequently, the number of precedence links should be restricted to avoid the overwhelming computational effort mentioned above.

10.5 Logistic regression

Logistic regression is a technique for converting binary classification problems into linear regression ones, described in Chapter 8, by means of a proper transformation.

Suppose that the response variable y takes the values {0,1}, as in a binary classification problem. The logistic regression model postulates that the posterior probability $P(y|\mathbf{x})$ of the response variable conditioned on the vector $\mathbf{x}$ follows a *logistic function*, given by

$$P(y = 0|\mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}'\mathbf{x}}}, \tag{10.31}$$

$$P(y = 1|\mathbf{x}) = \frac{e^{\mathbf{w}'\mathbf{x}}}{1 + e^{\mathbf{w}'\mathbf{x}}}. \tag{10.32}$$

Here we suppose that the matrix $\mathbf{X}$ and the vector $\mathbf{w}$ have been extended to include the intercept as described in Section 8.3. The *standard logistic function* $S(t)$, also known as the *sigmoid* function, can be found in many applications of statistics in the economic and biological fields and is defined as

$$S(t) = \frac{1}{1 + e^{-t}}. \tag{10.33}$$

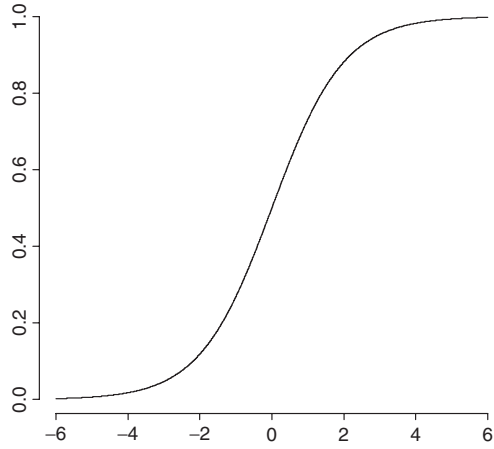

Figure 10.15 Graph of the standard logistic function (sigmoid)

The function $S(t)$ has the graphical shape shown in Figure 10.15.

By inverting expressions (10.31) and (10.32), we observe that the logarithm of the ratio between the conditional probabilities of the two classes depends linearly on the predictive variables, that is

$$\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \mathbf{w}'\mathbf{x}. \tag{10.34}$$

Consequently, by setting

$$z = \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}, \tag{10.35}$$

the binary classification problem is traced back to the identification of a linear regression model between the dependent variable z and the original explanatory attributes. Once the linear regression coefficients have been calculated and the significance of the model verified, using the validation tests described in Section 8.4, one may use the model to predict the target class of a new observation $\mathbf{x}$. The coefficients $\mathbf{w}$ are computed using an iterative method, usually aimed at maximizing the likelihood, by minimizing the sum of logarithms of predicted probabilities.

In general, logistic regression models present the same difficulties described in connection with regression models, from which they derive. To avoid multicollinearity phenomena that jeopardize the significance of the regression coefficients it is necessary to proceed with attribute selection. Moreover, the accuracy of logistic regression models is in most cases lower than that obtained using other classifiers and usually requires a greater effort for the development of the model. Finally, it appears computationally cumbersome to treat large datasets, both in terms of number of observations and number of attributes.

10.6 Neural networks

Neural networks are intended to simulate the behavior of biological systems composed of neurons. Since the 1950s, when the simplest models were proposed, neural networks have been used for predictive purposes, not only for classification but also for regression of continuous target attributes.

A neural network is an oriented graph consisting of nodes, which in the biological analogy represent neurons, connected by arcs, which correspond to dendrites and synapses. Each arc is associated with a *weight*, while at each node an *activation function* is defined which is applied to the values received as input by the node along the incoming arcs, adjusted by the weights of the arcs. The training stage is performed by analyzing in sequence the observations contained in the training set one after the other and by modifying at each iteration the weights associated with the arcs.

10.6.1 The Rosenblatt perceptron

The *perceptron*, shown in Figure 10.16, is the simplest form of neural network and corresponds to a single neuron that receives as input the values $(x_1, x_2, \ldots, x_n)$ along the incoming connections, and returns an output value $f(\mathbf{x})$. The input values coincide with the values of the explanatory attributes, while the output value determines the prediction of the response variable y. Each of the n input connections is associated with a weight w_j. An activation function g and a constant ϑ, called the *distortion*, are also assigned.

Suppose that the values of the weights and the distortion have already been determined during the training phase. The prediction for a new observation $\mathbf{x}$ is then derived by performing the following steps.

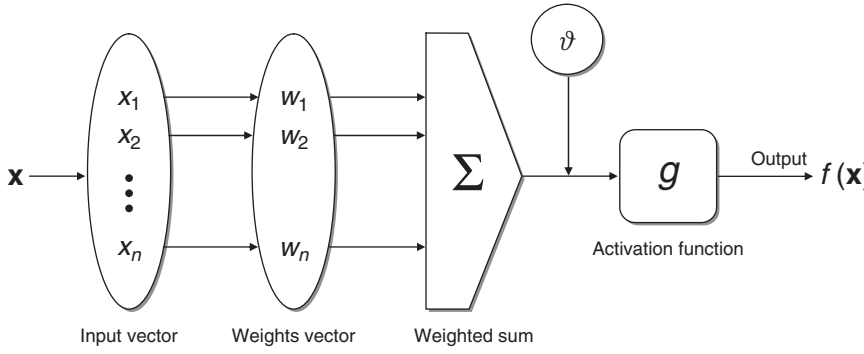

Figure 10.16 Operation of a single unit in a neural network

First, the weighted linear combination of the values of the explanatory variables for the new observation is calculated and the distortion is subtracted from it:

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - \vartheta = \mathbf{w}'\mathbf{x} - \vartheta. \tag{10.36}$$

The prediction $f(\mathbf{x})$ is then obtained by applying the activation function g to the linear combination of the predictors:

$$f(\mathbf{x}) = g(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - \vartheta) = g(\mathbf{w}'\mathbf{x} - \vartheta). \tag{10.37}$$

The purpose of the function g is to map the linear combination into the set $\mathcal{H} = \{v_1, v_2, \ldots, v_H\}$ of the values assumed by the target variable, usually by means of a sigmoid profile. For binary classification problems we have $\mathcal{H} = \{-1, 1\}$, so that one may select $g(\cdot) = \text{sgn}(\cdot)$, making the prediction coincide with the sign of the weighted sum in (10.36):

$$f(\mathbf{x}) = \text{sgn}(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - \vartheta) = g(\mathbf{w}'\mathbf{x} - \vartheta). \tag{10.38}$$

An iterative algorithm is then used to determine the values of the weights w_j and the distortion ϑ, examining the examples in sequence, one after the other. For each example $\mathbf{x}_i$ the prediction $f(\mathbf{x}_i)$ is calculated, and the value of the parameters is then updated using recursive formulas that take into account the error $y_i - f(\mathbf{x}_i)$.

For binary classification problems it is possible to give a geometrical interpretation of the prediction obtained using a Rosenblatt perceptron. Indeed, if we place the m observations of the training dataset in the space $\mathbb{R}^n$, the weighted linear combination in (10.36) calculated for $\mathbf{x}_i$ expresses the slack between the observation and the hyperplane:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - \vartheta = \mathbf{w}'\mathbf{x} - \vartheta. \tag{10.39}$$

The purpose of the activation function $g(\cdot) = \text{sgn}(\cdot)$ is therefore to establish if the point associated with the example $\mathbf{x}_i$ is placed in the lower or upper halfspace with respect to the separating hyperplane. Hence, the Rosenblatt perceptron corresponds to a linear separation of the observations based on the target class. The aim of the iterative procedure is therefore to determine the coefficients of the separating hyperplane. Notice, however, that these coefficients can be derived more efficiently by solving a single optimization problem, as we will show in Section 10.7 devoted to support vector machines.

10.6.2 Multi-level feed-forward networks

A *multi-level feed-forward* neural network, shown in Figure 10.17, is a more complex structure than the perceptron, since it includes the following components.

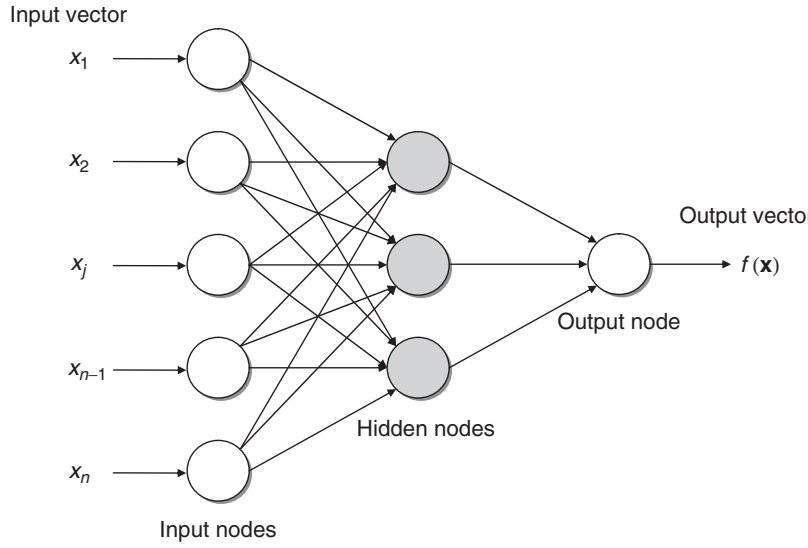

Figure 10.17 Example of neural network

Input nodes. The purpose of the input nodes is to receive as input the values of the explanatory attributes for each observation. Usually, the number of input nodes equals the number of explanatory variables.

Hidden nodes. Hidden nodes apply given transformations to the input values inside the network. Each node is connected to incoming arcs that go from other hidden nodes or from input nodes, and it is connected with outgoing arcs to output nodes or to other hidden nodes.

Output nodes. Output nodes receive connections from hidden nodes or from input nodes and return an output value that corresponds to the prediction of the response variable. In classification problems, there is usually only one output node.

Each node of the network basically operates as a perceptron, in the sense that given weights are associated with the input arcs, while each node is associated with a distortion coefficient and an activation function. In general, the activation function may assume forms that are more complex than the sign function sgn(·), such as a linear function, a sigmoid or a hyperbolic tangent.

The method that determines the weights of all the arcs and the distortions at the nodes, called the *backpropagation* algorithm, follows a logic that is not dissimilar from that described for the single perceptron. The weights are initialized in an arbitrary way, for instance by setting their value equal to randomly generated numbers. The examples of the training set are therefore

examined in sequence, using at each iteration the current values of the weights, in order to calculate the prediction and the corresponding misclassification error. This latter is used to recursively correct the values of the weights, used at a later time to analyze the subsequent example within the procedure. The weights are updated using a descent algorithm, which is a variant of the gradient method.

One of the strengths of neural networks is that they are a learning mechanism applicable to both classification and regression problems. Furthermore, they perform attribute selection automatically, since irrelevant or redundant variables can be excluded from the analysis by looking at the coefficients assuming negligible values. However, neural networks require very long times for model training, provide results with modest interpretability that are dependent upon the order in which the examples are analyzed, and also present a lower robustness with respect to data affected by noise.

10.7 Support vector machines

Support vector machines are a family of separation methods for classification and regression developed in the context of statistical learning theory. They have been shown to achieve better performance in terms of accuracy with respect to other classifiers in several application domains, and to be efficiently scalable for large problems. A further important feature is concerned with the interpretation of the classification rules generated. Support vector machines identify a set of examples, called *support vectors*, which appear to be the most representative observations for each target class. In a way, they play a more critical role than the other examples, since they define the position of the separating surface generated by the classifier in the attribute space.

10.7.1 Structural risk minimization

As already observed, a classification algorithm $A_{\mathcal{F}}$ defines an appropriate hypothesis space $\mathcal{F}$ and a function $f^* \in \mathcal{F}$ which optimally describes the relationship between the class value y and the vector of explanatory variables $\mathbf{x}$. In order to describe the criteria for selecting the function f^*, let $V(y, f(\mathbf{x}))$ denote a loss function which measures the discrepancy between the values returned by the predictive function $f(\mathbf{x})$ and the actual values of the class y.

To select an optimal hypothesis $f^* \in \mathcal{F}$, decision theory suggests minimizing the *expected risk* functional, defined as

$$R(f) = \frac{1}{2} \int V(y, f(\mathbf{x})) dP(\mathbf{x}, y), \tag{10.40}$$

where $P(\mathbf{x}, y) = P_{\mathbf{x},y}(\mathbf{x}, y)$ denotes the joint probability distribution over $\mathbb{R}^n \times \mathcal{H}$ of the examples $(\mathbf{x}, y)$ from which the instances in the dataset $\mathcal{D}$ are assumed to be independently drawn.

Since the distribution $P(\mathbf{x}, y)$ is generally unknown, in place of the expected risk one is naturally led to minimize the *empirical risk* over the training set $\mathcal{T}$, defined as

$$R_{emp}(f) = \frac{1}{t}\sum_{i=1}^{t} V(y_i, f(\mathbf{x}_i)). \tag{10.41}$$

However, this induction principle for selecting the function f^*, called *empirical risk minimization* (ERM), suffers from two critical drawbacks, *overfitting* and *ill-posedness*, as described in what follows.

If the space $\mathcal{F}$ is chosen too broad, the empirical error can be significantly reduced, eventually to zero, but the optimal hypothesis f^* has a low generalization capability in predicting the output value of the instances in the test set $\mathcal{V}$ or of future unseen examples. In these cases, the expected risk may still be large even if the empirical risk approaches 0, since the minimization of $R_{emp}(f)$ does not necessarily imply the minimization of $R(f)$.

This phenomenon, known as *overfitting*, has been investigated in detail in the framework of statistical learning theory, leading to probability bounds on the difference between expected risk and empirical risk. In general, these bounds are inversely dependent on the size t of the training set $\mathcal{T}$ and directly dependent on the *capacity*, known as the *Vapnik–Chervonenkis dimension* (VC), which measures the complexity of the hypothesis space $\mathcal{F}$.

The VC dimension of the hypothesis space $\mathcal{F}$ is defined as the maximum number of training points that can be correctly classified by a function from $\mathcal{F}$, wherever the points are placed in the space and whatever their binary class values $\{-1, 1\}$ are. If the VC dimension is h, then there exists at least one set of $h + 1$ points that cannot be correctly classified by the hypotheses belonging to the space $\mathcal{F}$. For instance, if the training examples are represented by points in two-dimensional space, the class of functions represented by oriented straight lines has dimension VC $= 3$, since there exist sets of four points which cannot be *shattered* by means of a single separating line, as depicted in Figure 10.18. More generally, it has been shown that the class of separating hyperplanes in the space $\mathbb{R}^n$ has dimension VC $= n + 1$.

According to the most common bound on the expected risk, for a binary classification problem the inequality

$$\begin{aligned} R(f) &\leq \sqrt{\frac{\gamma \log(2t/\gamma) + 1 - \log(\eta/4)}{t}} + R_{emp}(f) \\ &= \Psi(t, \gamma, \eta) + R_{emp}(f), \end{aligned} \tag{10.42}$$

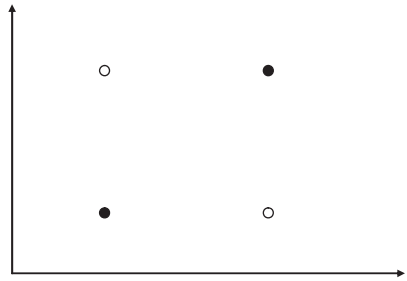

Figure 10.18 Four points in a plane which cannot be shattered by a straight line. The binary class value of each example is provided by its color

holds with probability $1 - \eta$, where $0 \leq \eta \leq 1$ is a predefined confidence level, γ is the VC dimension of the chosen hypothesis space $\mathcal{F}$, and $\Psi(t, \gamma, \eta)$ is a function called the *VC confidence* which represents the generalization error.

As a consequence, in order to control the classification error against broader hypothesis space, a larger training set is required, according to a proportion which is regulated by theoretical bounds on the error, cast in the form (10.42). Hence, in order to identify a hypothesis $f^* \in \mathcal{F}$ capable of achieving a high level of accuracy on the training set and a good generalization capability on the test set and on future sets of examples, one may resort to the *structural risk minimization* (SRM) principle, which formally establishes the concept of choosing the function $f^* \in \mathcal{F}$ which minimizes the right hand-side in (10.42).

Furthermore, the minimization of the empirical risk is an ill-conditioned problem, in the sense that the parameters of the optimal hypothesis f^* may vary significantly for small changes in the input data.

A way to circumvent the second weakness has been devised in the context of regularization theory and also applied in statistical learning theory to prevent overfitting. It relies on the solution of a well-posed problem which corresponds to the structural risk minimization principle, and chooses as the optimal hypothesis $f^* \in \mathcal{F}$ the one that minimizes a modified risk functional

$$\hat{R}(f) = \frac{1}{t} \sum_{i=1}^{t} V(y_i, f(\mathbf{x}_i)) + \lambda \| f \|_K^2, \tag{10.43}$$

where K is a given symmetric positive definite function called *kernel*, $\| f \|_K^2$ denotes the norm of f in the reproducing kernel Hilbert space[4] induced by K and λ is a parameter that controls the trade-off between the empirical error

[4]A *Hilbert space* is a linear space in which an inner product is defined, so that a norm is assigned which is also *complete* – that is, such that every Cauchy sequence is convergent to a point in the space. The simplest example of a finite-dimensional Hilbert space is the Euclidean space $\mathbb{R}^n$. However, there are also Hilbert spaces of infinite dimension.

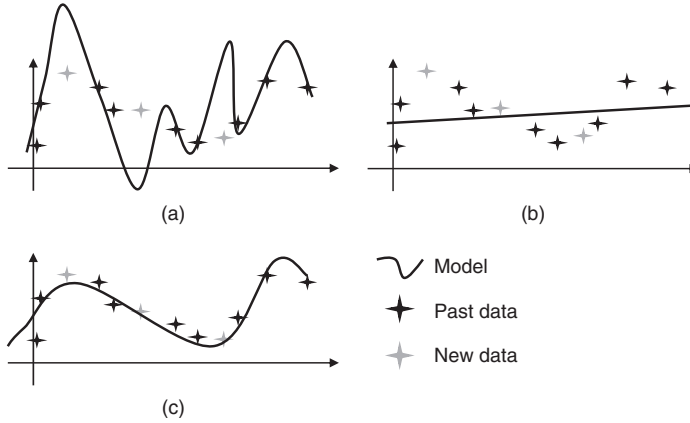

Figure 10.19 Finding the best trade-off between empirical error and generalization capability

and the generalization capability. The parameter λ can be interpreted as a penalty against a hypothesis f with high complexity VC and low generalization capability, and also as a smoothing regularizer for transforming the variational problem into a well-posed one. Figure 10.19 provides an intuitive explanation of how it is possible to find an ideal trade-off between the accuracy on the training set and the generalization capability to future unseen examples. The dark gray points represent the training examples, while the light gray ones correspond to new instances. The hypothesis shown Figure 10.19(a) belongs to a too broad space $\mathcal{F}$, which may be composed, for example, of high order polynomials; it is associated with a negligible classification error on the training set but it is not able to generalize well to future instances. On the other hand, the hypothesis in Figure 10.19(b) originates from a too narrow space, whereas in Figure 10.19(c) an ideal trade-off is achieved between the ability to provide an accurate classification of the training examples and the future generalization capability.

Figure 10.20 shows the conflicting effects produced on the two right-hand-side terms in (10.42) by an increase in the complexity of the hypothesis space $\mathcal{F}$, measured by the VC dimension along the horizontal axis: an increase in the complexity corresponds to a decrease in the empirical error but, at the same time, to an increase in the VC confidence which represents the generalization error.

The classical theory of support vector machines developed in the theoretical framework outlined above is based on the so called *soft-margin* loss function, given by

$$V(y_i, f(\mathbf{x}_i)) = |1 - y_i g(\mathbf{x}_i)|_+, \tag{10.44}$$

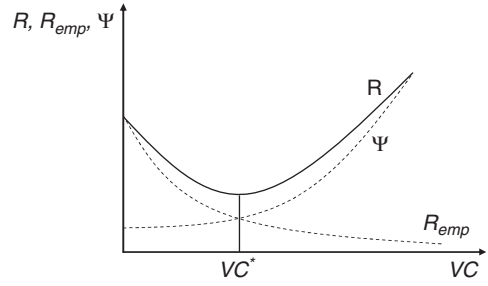

Figure 10.20 Searching for the optimal VC dimension

where g is a score function such that $f(\mathbf{x}) = \text{sgn}(g(\mathbf{x}))$ and $|t|_+ = t$ if t is positive and zero otherwise. In particular, the loss function (10.44) computes the sum of the slacks of the misclassified examples from the separating surface.

However, one may resort to alternative loss functions. For example, a different family of classification models, termed *discrete support vector machines*, is motivated by the following loss function that counts the number of misclassified examples:

$$V(y_i, f(\mathbf{x}_i)) = c_i \theta(-y_i g(\mathbf{x}_i)), \tag{10.45}$$

where $\theta(t) = 1$ if t is positive and zero otherwise, while c_i is a penalty for the misclassification of the example $\mathbf{x}_i$.

10.7.2 Maximal margin hyperplane for linear separation

Once the theoretical framework for the supervised learning process has been established, we are able to provide a geometrical interpretation of the structural risk minimization principle, and to formulate an optimization problem which aims to determine a linear separating surface for solving binary classification problems.

For linear separating functions represented by the hyperplanes in the space $\mathbb{R}^n$, the minimization of the right-hand side in (10.42) can be traced back to the maximization of the *margin of separation*, which will be described with reference to Figures 10.21 and 10.22.

Two sets of points belonging to binary target classes, like the ones depicted in Figure 10.21, are said to be *linearly separable* if there exists a hyperplane capable of separating them in the space $\mathbb{R}^n$, reducing to a line in the two-dimensional case. As shown in Figure 10.21, there are an infinite number of linear separating functions that appear equivalent to each other in terms of empirical error on the training set, which is equal to zero for all the lines drawn in the figure. However, lines (a) and (c), which lie near the training examples,

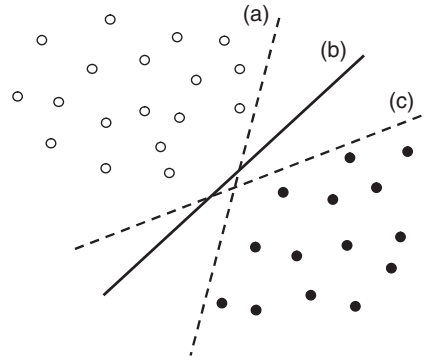

Figure 10.21 The lines represent possible separating hyperplanes in the two-dimensional space for the points in the diagram

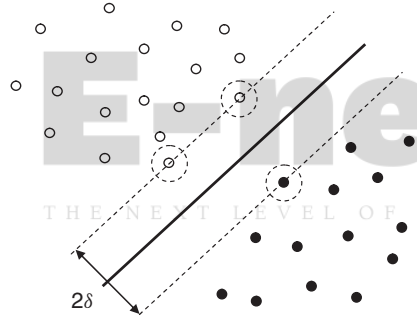

Figure 10.22 Maximal margin separating hyperplanes and canonical hyperplanes for a linearly separable dataset

have lower generalization capability with respect to a line, such as (b), that is equally distant from the nearest points of the two sets.

The optimal separating line, endowed with the maximum generalization capability and the minimum empirical error, is drawn in Figure 10.22. The margin of separation is defined as the distance between the pair of parallel *canonical supporting hyperplanes*, as shown in the same figure. Hence, the margin is equal to twice the minimum distance between the training points and the separating hyperplane. Those training points which are at the minimum distance from the separating hyperplane, and thus lie on the canonical hyperplanes, are called *support vectors* and play a more prominent role with respect to the other examples, since it is precisely these points that determine the classification rule.

Letting $\mathbf{w}$ denote the vector of the hyperplane coefficients and b the intercept, the separating hyperplane is given by

$$\mathbf{w}'\mathbf{x} = b, \tag{10.46}$$

while the two canonical supporting hyperplanes are

$$\mathbf{w}'\mathbf{x} - b - 1 = 0, \qquad \mathbf{w}'\mathbf{x} - b + 1 = 0. \tag{10.47}$$

The margin of separation δ is defined as

$$\delta = \frac{2}{\|w\|}, \quad \text{where} \quad \|w\| = \sqrt{\sum_{j \in \mathcal{N}} w_j^2}. \tag{10.48}$$

In order to determine the coefficients $\mathbf{w}$ and b of the optimal separating hyperplane, a quadratic optimization problem with linear constraints can be solved:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \tag{10.49}$$

$$\text{s.to} \quad y_i(\mathbf{w}'\mathbf{x}_i - b) \geq 1, \quad i \in \mathcal{M}. \tag{10.50}$$

The aim of the objective function is to maximize the margin of separation through the minimization of its reciprocal, while the constraints (10.50) force each point $\mathbf{x}_i$ to lie in the halfspace corresponding to the class value y_i.

It is most likely that the m points of a dataset cannot be linearly separable, as for the set of examples depicted in Figure 10.23. In these cases, it is necessary to relax the constraints (10.50) by replacing them with weaker conditions which allow for the presence of possible misclassification errors, and to modify the objective function of the optimization problem. This can be done by introducing a new set of *slack variables* d_i, $i \in \mathcal{M}$, which measure the positive differences between the values of the misclassified examples on the vertical axis and the ordinate values along the canonical hyperplane that defines the region associated with the class value y_i, as geometrically illustrated in Figure 10.23.

In the non-separable case, we can therefore formulate the following optimization problem:

$$\min_{\mathbf{w},b,d} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \lambda \sum_{i=1}^{m} d_i \tag{10.51}$$

$$\text{s.to} \quad y_i(\mathbf{w}'\mathbf{x}_i - b) \geq 1 - d_i, \quad i \in \mathcal{M}, \tag{10.52}$$

$$d_i \geq 0, \qquad i \in \mathcal{M}. \tag{10.53}$$

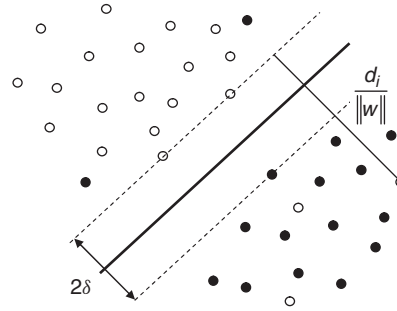

Figure 10.23 Maximal margin separating hyperplanes and canonical hyperplanes for a nonlinearly separable dataset

The objective function is composed of the weighted sum of two terms representing respectively the reciprocal of the margin of separation and the empirical error. The parameter λ is introduced in order to regulate the trade-off between the generalization capability, represented by the reciprocal of the margin, and the accuracy on the training set, evaluated as the sum of the slack variables.

The quadratic problem (10.51) can be solved via Lagrangian duality. Among other advantages, this allows us to identify the support vectors, which are associated with positive Lagrange multipliers in the optimal solution of the dual problem.

Denote by $\alpha_i \geq 0$ the Lagrangian multipliers of the constraints (10.52) and by $\mu_i \geq 0$ the multipliers of the constraints (10.53). The Lagrangian function of problem (10.51) is given by

$$L(\mathbf{w}, b, \mathbf{d}, \boldsymbol{\alpha}\, \boldsymbol{\mu}) = \frac{1}{2}\|\mathbf{w}\|^2 + \lambda \sum_{i=1}^{m} d_i - \sum_{i=1}^{m} \alpha_i [y_i(\mathbf{w}'\mathbf{x}_i - b) - 1 + d_i] - \sum_{i=1}^{m} \mu_i d_i. \tag{10.54}$$

In order to find the optimal solution, the derivatives with respect to the variables $\mathbf{w}, \mathbf{d}, b$ of the primal problem (10.51) must be set to 0,

$$\frac{\partial L(\mathbf{w}, b, \mathbf{d}, \boldsymbol{\alpha}\, \boldsymbol{\mu})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = \mathbf{0}, \tag{10.55}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{d}, \boldsymbol{\alpha}\, \boldsymbol{\mu})}{\partial \mathbf{d}} = \lambda - \alpha_i - \mu_i = 0, \tag{10.56}$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{d}, \boldsymbol{\alpha}\, \boldsymbol{\mu})}{\partial b} = \sum_{i=1}^{m} \alpha_i y_i = 0, \tag{10.57}$$

leading to the conditions

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{10.58}$$

$$\lambda = \alpha_i + \mu_i, \tag{10.59}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0. \tag{10.60}$$

By substituting these equality constraints into the Lagrangian function (10.54), we obtain the objective function of the dual problem

$$L(\mathbf{w}, b, \mathbf{d}, \boldsymbol{\alpha}\, \boldsymbol{\mu}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{h=1}^{m} y_i y_h \alpha_i \alpha_h \mathbf{x}_i' \mathbf{x}_h, \tag{10.61}$$

with the additional constraints $\alpha_i \leq \lambda, i \in \mathcal{M}$. The Karush–Kuhn–Tucker complementarity conditions applied to the pair of primal–dual problems lead to the equalities

$$\alpha_i [y_i (\mathbf{w}' \mathbf{x}_i - b) - 1 + d_i] = 0, \qquad i \in \mathcal{M}, \tag{10.62}$$

$$\mu_i (\alpha_i - \lambda) = 0, \qquad i \in \mathcal{M}. \tag{10.63}$$

In particular, conditions (10.63) allow us to identify the support vectors, which are the most representative points in determining the classification rules learned from the training set. Indeed, the examples $\mathbf{x}_i$ whose Lagrangian multipliers satisfy the condition $0 < \alpha < \lambda$ are at a distance $1/\|\mathbf{w}\|$ from the separating hyperplane, and thus are located on one of the two canonical hyperplanes.

Finally, observe that for support vector machines the decision function which classifies a new observation $\mathbf{x}$ is given by

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}'\mathbf{x} - b). \tag{10.64}$$

By substituting (10.58) into (10.64), we can reformulate the decision function as

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \mathbf{x}' \mathbf{x}_i - b \right). \tag{10.65}$$

10.7.3 Nonlinear separation

Linear separating functions are not able to perform accurate classifications when the set of examples is intrinsically characterized by a nonlinear pattern, like that shown in Figure 10.24(a).

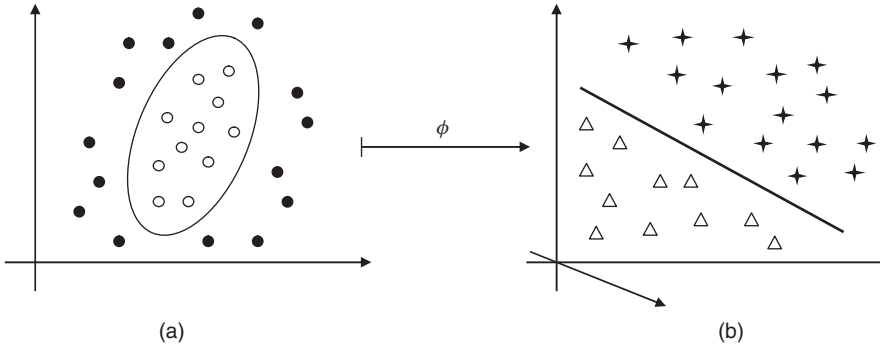

Figure 10.24 Nonlinear separation achieved by means of transformations in the feature space

In such cases, one may resort to mappings of the attributes which allow one to obtain linearly separable datasets in the transformed space, which is called the *feature space*. Figure 10.24 shows a possible transformation enjoying this property. In Figure 10.24(a) the examples of opposite classes can be separated by an elliptical function represented by a second-order polynomial in the explanatory variables. If the original two-dimensional space is transformed into a five-dimensional space which contains the second-order monomials x_1x_2, x_1^2, x_2^2, in addition to the original attributes x_1 and x_2, it is possible to linearly separate the transformed points by means of a hyperplane defined in the space $\mathbb{R}^5$. Notice that the linear separation achieved in this way is made possible by an increase in the number of attributes in the feature space, which may be quite relevant for some applications.

Besides the intuitive justification based on Figure 10.24, the mapping from the original attribute space to a feature space of high – sometimes even infinite – dimensionality can also be justified from a theoretical point of view by Cover's theorem. We already know that m points in a general position in the n-dimensional space can be linearly separated if $m \leq n + 1$, since the VC dimension of the class of hyperplanes is $n + 1$, so that the number of possible linear separations is 2^m. If, however, $m > n + 1$, Cover's theorem states that the number of linear separations is given by

$$2\sum_{i=0}^{n}\binom{m-1}{i}. \tag{10.66}$$

As n increases, the number of terms in the sum increases in turn, so that there are more linear separations.

This line of reasoning may, however, entail some practical difficulties. Indeed, the transformation must be carried out very efficiently even if the

feature space is of high or infinite dimension. Fortunately, there exists a wide class of mappings which can be evaluated in a very efficient way, consisting of *kernel functions*, for which the mapping of the original observations into the feature space is not explicitly computed. The use of kernels is based on the dual formulation of problem (10.51), and implicitly involves a linear separation of the examples in high-dimensional functional spaces – or even infinite-dimensional for some families of kernels.

Specifically, consider a map $\mathbf{\Phi} : \mathbb{R}^n \mapsto \mathcal{B}$ that transforms the examples given in the space $\mathbb{R}^n$ of the original attributes into a Hilbert space $\mathcal{B}$, called the *feature space*.

Observe now that the arguments set out in Section 10.7.2 to derive the maximal margin hyperplane can easily be adapted to achieve an optimal linear separation in the feature space. To do this, we simply have to substitute $\mathbf{\Phi}(\mathbf{x})$ wherever we wrote $\mathbf{x}$ in expressions (10.46) through (10.65). In particular, the Lagrangian dual problem takes the form

$$L(\mathbf{w}, b, \mathbf{d}, \boldsymbol{\alpha}\, \boldsymbol{\mu}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{h=1}^{m} y_i y_h \alpha_i \alpha_h \mathbf{\Phi}(\mathbf{x}_i)' \mathbf{\Phi}(\mathbf{x}_h), \tag{10.67}$$

and we obtain the following decision function in the feature space:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \mathbf{\Phi}(\mathbf{x})' \mathbf{\Phi}(\mathbf{x}_i) - b \right). \tag{10.68}$$

Now, the key point is that the data appear in the Lagrangian dual (10.67) and in the corresponding decision function (10.68) only in the form of inner products $\mathbf{\Phi}(\mathbf{x}_i)' \mathbf{\Phi}(\mathbf{x}_h)$ and $\mathbf{\Phi}(\mathbf{x})' \mathbf{\Phi}(\mathbf{x}_i)$. Hence, the potentially expensive computation of the images $\mathbf{\Phi}(\mathbf{x}_i)$, $i \in \mathcal{M}$, of the observations under the mapping $\mathbf{\Phi}$ can be dramatically simplified if there exists a positive definite *kernel function* k such that

$$\mathbf{\Phi}(\mathbf{x}_i)' \mathbf{\Phi}(\mathbf{x}_h) = k(\mathbf{x}_i, \mathbf{x}_h). \tag{10.69}$$

If such a kernel function to represent the mapping $\mathbf{\Phi}$ exists, then the dual problem takes the form

$$L(\mathbf{w}, b, \mathbf{d}, \boldsymbol{\alpha}\, \boldsymbol{\mu}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{h=1}^{m} y_i y_h \alpha_i \alpha_h k(\mathbf{x}_i, \mathbf{x}_h), \tag{10.70}$$

with the additional constraints $\alpha_i \leq \lambda$, $i \in \mathcal{M}$. Since k is positive definite, the matrix $\mathbf{Q} = [Q_{ij}] = [y_i y_k k(\mathbf{x}_i, \mathbf{x}_h)]$ is positive definite in turn, so that problem (10.70) is convex and can be solved efficiently to derive the dual multipliers

α_i, $i \in \mathcal{M}$. Therefore, this allows us to identify the support vectors, and to express the decision function in the form

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) - b\right). \tag{10.71}$$

The linear separation in the feature space hence corresponds to a nonlinear separation in the space of the original attributes, which by means of kernel functions can be computed with basically the same effort as a linear separation in the original space.

There remains the question of the existence of meaningful kernel functions that enable accurate nonlinear separations to be computed. Sufficient conditions for the existence of pairs $(k, \mathbf{\Phi})$ consisting of a kernel function and a map $\mathbf{\Phi}$ represented by k, are provided by Mercer's theorem, whose formulation goes beyond the scope of our discussion and for which the reader is directed to the references suggested at the end of this chapter. Mercer's theorem also implies that the linear combination of kernel functions is in turn a kernel function, so that it provides a rule for deriving more complex kernels starting from simple ones.

Many kernels have been proposed in the literature, among which the most popular parametric families are *polynomial* kernels of degree d,

$$k(\mathbf{x}_i, \mathbf{x}_h) = (\mathbf{x}_i' \mathbf{x}_h + 1)^d; \tag{10.72}$$

radial basis function kernels, also called *Gaussian* kernels, of width $\sigma > 0$,

$$k(\mathbf{x}_i, \mathbf{x}_h) = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_h||^2}{2\sigma^2}\right); \tag{10.73}$$

and *neural networks* kernels with a hyperbolic tangent activation function, $\kappa > 0$,

$$k(\mathbf{x}_i, \mathbf{x}_h) = \tanh(\kappa \mathbf{x}_i' \mathbf{x}_h - \delta). \tag{10.74}$$

Notice that nonlinear separations can be also attained in the original space of attributes where the examples are defined, without resorting to mappings carried out by means of kernel functions.

For example, some authors have proposed polyhedral methods, described in Figure 10.25, which have been shown to perform very accurate classifications even in the presence of a small number of training examples. Alternatively, nonlinear separations may be achieved by means of hybrid classification methods, which recursively partition the data with linear classifiers embedded within classification trees, as depicted in Figure 10.26. These methods appear to be quite effective for solving multicategory classification problems.

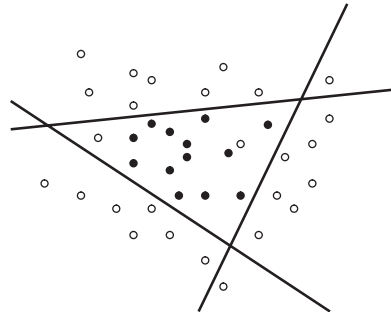

Figure 10.25 Classification by means of polyhedral separating regions

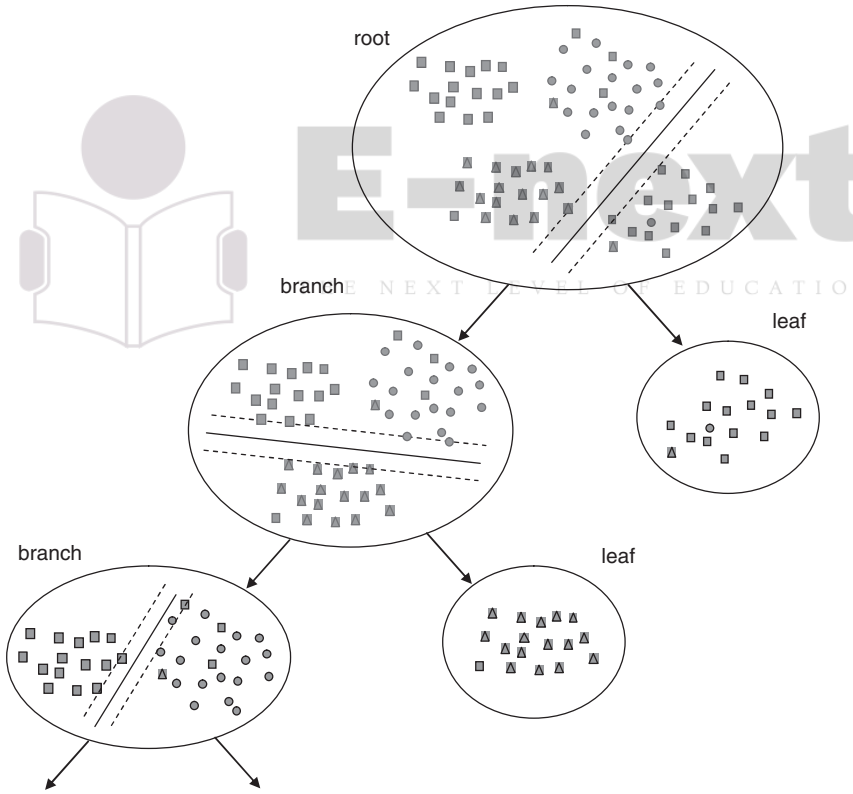

Figure 10.26 Nonlinear separation by means of oblique classification trees

In conclusion, the theoretical developments in statistical learning theory and the classification algorithms based on support vector machines have played a prominent role in defining a rigorous methodological framework for supervised learning methods. The extent of the application domain and the considerable classification accuracy make support vector machines and their variants the most effective methods currently available for supervised learning analysis.

10.8 Notes and readings

The first works on classification go back to early classical statistics, under the heading of *discriminant analysis* (Fisher, 1936).

Classification models play a prominent role in artificial intelligence, where they are designated as *pattern recognition* and *machine learning*. Several classical problems in artificial intelligence can be traced back to the classification of objects: from voice and sound recognition, to character and text recognition, to static and dynamic image recognition. There are several texts devoted to pattern recognition, among which Duda *et al*. (2001), Fukunaga (1990), Baldi and Brunak (2001), Cherkassky and Mulier (1998), Jensen and Cohen (2000) and Kulkarni *et al*. (1998) are of particular note.

For axis-parallel classification trees the reader can refer, for example, to Breiman *et al*. (1984), Quinlan (1993), Breslow and Aha (1997), Murthy (1998) and Safavian and Landgrebe (1998). For oblique trees see Murthy *et al*. (1994) and Orsenigo and Vercellis (2003), Orsenigo and Vercellis (2006). For logistic regression refer to Pampel (2000) and Scott Long (1997). For Bayesian classifiers see Ramoni and Sebastiani (2001) and Domingos and Pazzani (1997). Thorough descriptions of neural networks can be found in Fausett (1994), Bishop (1995), Anderson (1995), Ripley (1996), Schurmann (1996), Haykin (1998) and Raudys (2001); see also Rosenblatt (1962) for its historical value.

The literature on support vector machines is very broad; see Vapnik (1996, 1998), Cristianini and Shawe-Taylor (2000, 2004), Schölkopf and Smola (2001) and Burges (1998). The relationship between statistical learning theory and regularization theory is clearly analyzed in Evgeniou *et al*. (2002). Discrete support vector machines have been proposed and developed in Orsenigo and Vercellis (2004, 2007, 2009). Mangasarian (1997), Bradley *et al*. (1999) and Mirkin (1996) focus on the role of optimization models in classification.

For a discussion of the criteria for evaluating and comparing different classifiers see Kohavi (1995).