CHAPTER

13 Basic classification concepts

Pattern recognition (PR) is a task that humans are able to achieve "at a glance" with little apparent effort. Much of PR is structural, being achieved essentially by analyzing shape. In contrast, statistical PR (SPR) treats sets of extracted features as abstract entities that can be used to classify objects on a statistical basis, often by mathematical similarity to sets of features for objects with known classes. This chapter explores the subject, presenting relevant theory where appropriate, and shows how artificial neural networks (ANNs) are able to help with recognition tasks.

Look out for:

- The nearest neighbor (NN) algorithm—probably the most intuitive of all SPR techniques
- Bayes' theory, which forms the ideal minimum error classification system
- The relation linking the NN algorithm to Bayes' theory
- The reason why the optimum number of features will always be finite
- The distinction between supervised and unsupervised learning
- The cluster analysis approach to unsupervised learning
- The support vector machine (SVM) approach to supervised learning
- How ANNs can be trained, avoiding problems of inadequate training and overfitting to training data.

SPR is a core methodology in the design of practical vision systems. As such, it has to be used in conjunction with structural PR methods and many other relevant techniques. In Chapter 14, Machine Learning: Probabilistic Methods, we shall see how these foundations are extended to cover probabilistic PR (PPR) and the modern subject of machine learning (ML).

13.1 INTRODUCTION

Parts 1 and 2 of this book tackle the task of interpreting images on the basis that when suitable cues have been found, the identities and positions of various objects will emerge in a natural and straightforward way. When the objects that appear in

Computer Vision. DOI: http://dx.doi.org/10.1016/B978-0-12-809284-2.00013-7

an image have simple shapes, just one stage of processing may be required—as in the case of circular washers on a conveyor. For more complex objects such as flat brackets and hinges, location requires at least two stages—as when graph matching methods are used. For situations where the full complexity of three dimensions occurs, more subtle procedures are usually required, as will be seen in Part 4. Indeed, the very ambiguity involved in interpreting 2-D images from a set of 3-D objects generally requires cues to be sought and hypotheses to be proposed before any serious attempt can be made with the task. Thus, cues are vital to keying into the complex data structures of many images. However, for simpler situations, concentration on small features is valuable in permitting image interpretation to be carried out efficiently and rapidly; neither must it be forgotten that in many applications of computer vision, the task is made simpler by the fact that the main interest lies in specific types of object, e.g., the widgets to be inspected on a widget line or the vehicles to be seen on a highway. Indeed, there are many more constrained types of situation in which the various possible solutions can be evaluated carefully before a final interpretation is reached. These situations arise practically where small relevant parts of images can be segmented and interpreted in isolation. One such case is that of optical character recognition (OCR): a commonly used approach for tackling it is that of SPR.

FROM "STATISTICAL PATTERN RECOGNITION" TO "MACHINE LEARNING"

In practical situations, measurements of prominent features allow most objects to be classified straightforwardly. In fact, this is commonly achieved with varying degrees of certainty, but by comparing object features with those of a great many other known objects, we arrive at classifications that are statistically the most likely ones. Hence, this sort of classification procedure is called *SPR*.

A good number of early PR techniques and algorithms have followed this statistical approach without proceeding to the next stage—that of determining the solution that is mathematically the most *probable* one. Thus, an important goal has been to move the subject from *statistical PR* to *PPR*, i.e., from solutions that are not merely statistically good to those that are known to be probabilistically optimal. Indeed, it is a key aim of ML to progress toward probabilistic optimization of measurement and recognition. In Chapter 14, Machine Learning: Probabilistic Methods, we shall see how ML achieves this aim in a number of important cases. The purpose of the present chapter is to lay firm foundations that lead to the core of ML.

The following sections study the principles of SPR. A description of all the work that has been carried out in this area would take several volumes to cover and cannot be attempted here. Fortunately, SPR has been researched for well over four decades and an overview chapter at this early stage in this book will be useful. We start by describing the NN approach to SPR and then go on to consider Bayes' decision theory that provides a more general model of the underlying process.

13.2 THE NEAREST NEIGHBOR ALGORITHM

The principle of the NN algorithm is that of comparing input image patterns against a number of paradigms and then classifying them according to the class of the paradigm that gives the closest match (Fig. 13.1). An instructive but rather trivial example is shown in Fig. 1.1. Here, a number of binary patterns are presented to the computer in the training phase of the algorithm: then the test patterns are presented one at a time and compared bit by bit against each of the training patterns. It is clear that this gives a generally reasonable result, the main problems arise when (1) training patterns of different classes are close together in Hamming distance (i.e., they differ in too few bits to be readily distinguishable), and (2) minor translations, rotations, or noise cause variations that inhibit accurate recognition. More generally, problem (2) means that the training patterns are insufficiently representative of what will appear during the test phase. The latter statement encapsulates an exceptionally important principle, and it implies that there must be sufficient patterns in the training set for the algorithm to be able to generalize over all possible patterns of each class. However, problem (1) implies that patterns of two different classes may in some cases be so similar as to be indistinguishable by any algorithm, and then it is inevitable that erroneous classifications will be made. It is seen below that this is because the underlying distributions in feature space overlap. (Note that a number of the methods discussed in this chapter—such as the NN algorithm outlined above—are very general and can be applied to the recognition of widely different datasets, including for example speech and electrocardiograph waveforms.)

The example of Fig. 1.1 is rather trivial but nevertheless carries important lessons. Note that general images have many more pixels than those in Fig. 1.1 and

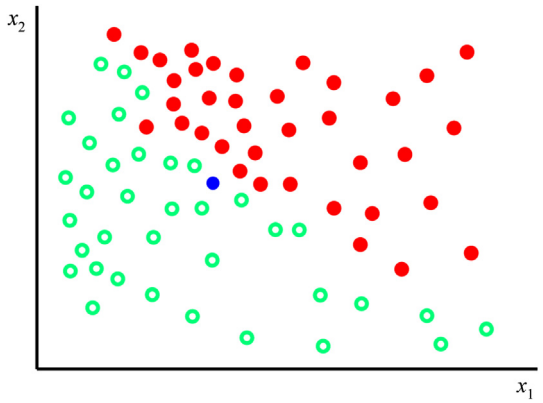

FIGURE 13.1

Principle of the nearest neighbor algorithm for a two-class problem: the class 1 training set patterns are shown in red, the class 2 training set patterns are shown in green, and the single test pattern is shown in blue.

also are not merely binary. However, as it is pertinent to simplify the data as far as possible to save computation, it is usual to concentrate on various features of a typical image and classify on the basis of these. One example is provided by a more realistic version of the OCR problem where characters have linear dimensions of at least 32 pixels (although we continue to assume that the characters have been located *reasonably* accurately so that it remains only to classify the subimages containing them). We can start by thinning the characters to their skeletons and making measurements on the skeleton nodes and limbs (see also Chapter 1: Vision, the Challenge and Chapter 8: Binary Shape Analysis): this gives (1) the numbers of nodes and limbs of various types, (2) the lengths and relative orientations of limbs, and perhaps (3) information on curvatures of limbs. Thus, we arrive at a set of numerical features that describe the character in the subimage.

The general technique is now to plot the characters in the training set in a multidimensional feature space and to tag the plots with the classification index. Then, test patterns are placed in turn in the feature space and classified according to the class of the nearest training set pattern. Clearly, this generalizes the method adopted in Fig. 13.1. In the general case, the distance in feature space is no longer Hamming distance but some more general measures such as Mahalanobis distance (Duda and Hart, 1973). In fact, a problem arises as there is no reason why the different dimensions in feature space should contribute equally to distance; rather, they should each have different weights in order to match the physical problem more closely. The problem of weighting cannot be discussed in detail here, and the reader is referred to other texts such as that by Duda and Hart (1973). Suffice to say that with an appropriate definition of distance, the generalization of the method outlined above is adequate to cope with a variety of problems.

In order to achieve a suitably low error rate, large numbers of training set patterns are normally required. This then leads to significant storage and computation problems. Means have been found for reducing these problems by several important strategies. Notable amongst these is that of pruning the training set by eliminating patterns that are not near the boundaries of class regions in feature space, as such patterns do not materially help in reducing the misclassification rate.

An alternative strategy for obtaining equivalent performance at lower computational cost is to employ a piecewise linear or other functional classifier instead of the original training set. Clearly, the NN method itself can be replaced, with no change in performance, by a set of planar decision surfaces that are the perpendicular bisectors (or their analogues in multidimensional space) of the lines joining pairs of training patterns of different classes that are on the boundaries of class regions. If this system of planar surfaces is simplified by any convenient means, then the computational load may be reduced further (Fig. 13.2). This may be achieved either indirectly by some smoothing process, as implied above, or directly by finding training procedures that act to update the positions of decision surfaces immediately on receipt of each new training set pattern. The latter

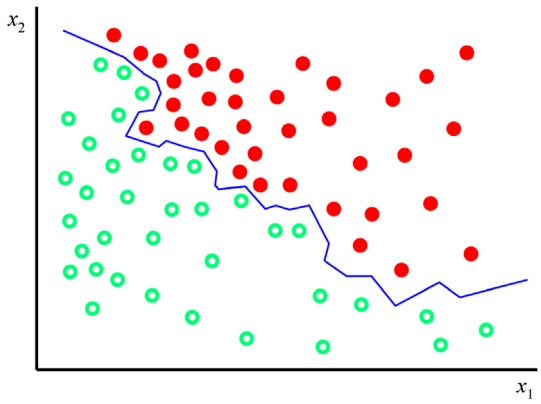

FIGURE 13.2

Use of planar decision surfaces for pattern classification; in this example, the "planar decision surface" reduces to a piecewise linear decision boundary in two dimensions. Once the decision boundary is known, the training set patterns themselves need no longer be stored.

approach is in many ways more attractive, as it drastically cuts down storage requirements—although it must be confirmed that a training procedure is selected that converges sufficiently rapidly. Again, discussion of this well-researched topic is left to other texts (Nilsson, 1965; Duda and Hart, 1973; Devijver and Kittler, 1982).

We now turn to a more generalized approach—that of Bayes' decision theory—as it underpins all the possibilities thrown up by the NN method and its derivatives.

13.3 BAYES' DECISION THEORY

The basis of Bayes' decision theory will now be examined. If we are trying to get a computer to classify objects, a sound approach is to get it to measure some prominent feature of each object such as its length and to use this feature as an aid to classification. Sometimes, such a feature may give very little indication of the pattern class—perhaps because of the effects of manufacturing variation. For example, a hand-written character may be so ill formed that its features are of little help in interpreting it; it then becomes much more reliable to make use of the known relative frequencies of letters, or to invoke context. In fact, either of these strategies can give a greatly increased probability of correct interpretation. In other words, when feature measurements are found to be giving an error rate above a certain threshold, it is more reliable to employ the a priori probability of a given pattern appearing.

The next step in improving recognition performance is to combine the information from feature measurements and from a priori probabilities; this is achieved by applying Bayes' rule. For a single feature x, this takes the form:

$$P(C_i|x) = p(x|C_i)P(C_i)/p(x) \tag{13.1}$$

where

$$p(x) = \sum_j p(x|C_j)P(C_j) \tag{13.2}$$

Mathematically, the variables here are (1) the a priori probability of class C_i, $P(C_i)$; (2) the probability density for feature x, $p(x)$; (3) the class-conditional probability density for feature x in class C_i, $p(x|C_i)$—i.e., the probability that feature x arises for objects known to be in class C_i; and (4) the a posteriori probability of class C_i when x is observed, $P(C_i|x)$.

The notation $P(C_i|x)$ is a standard one, being defined as the probability that the class is C_i when the feature is known to have the value x. Bayes' rule says that to find the class of an object, we need to know two sets of information about the objects that might be viewed: the first is the basic probability $P(C_i)$ that a particular class might arise; the second is the distribution of values of the feature x for each class. Fortunately, each of these sets of information can be found straightforwardly by observing a sequence of objects, e.g., as they move along a conveyor. As before, such a sequence of objects is called a training set.

Many common image analysis techniques give features that may be used to help identify or classify objects. These include the area of an object, its perimeter, the number of holes it possesses, and so on. It is important to note that classification performance may be improved not only by making use of the a priori probability but also by employing a number of features simultaneously. Generally, increasing the number of features helps to resolve object classes and reduce classification errors (Fig. 13.3); however, the error rate is rarely reduced to zero merely by adding more and more features, and indeed the situation eventually deteriorates for reasons explained in Section 13.5.

Bayes' rule can be generalized to cover the case of a generalized feature $\mathbf{x}$, in multidimensional feature-space, by using the modified formula:

$$P(C_i|\mathbf{x}) = p(\mathbf{x}|C_i)P(C_i)/p(\mathbf{x}) \tag{13.3}$$

where $P(C_i)$ is the a priori probability of class C_i, and $p(\mathbf{x})$ is the overall probability density for feature vector $\mathbf{x}$:

$$p(\mathbf{x}) = \sum_j p(\mathbf{x}|C_j)P(C_j) \tag{13.4}$$

The classification procedure is then to compare the values of all the $P(C_j|\mathbf{x})$ and to classify an object as class C_i if:

$$P(C_i|\mathbf{x}) > P(C_j|\mathbf{x}) \quad \text{for all } j \neq i \tag{13.5}$$

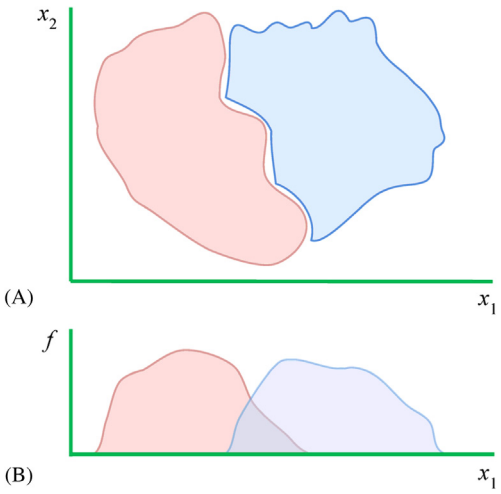

FIGURE 13.3

Use of several features to reduce classification errors: (A) the two regions to be separated in 2-D (x_1, x_2) feature space; (B) frequencies of occurrence of the two classes when the pattern vectors are projected onto the x_1-axis. Clearly, error rates will be high when either feature is used on its own but will be reduced to a low level when both features are employed together.

13.3.1 THE NAÏVE BAYES' CLASSIFIER

Many classification methods, including the NN algorithm and the Bayes' classifier, can involve substantial amounts of storage and computation if the amount of training is to be sufficient to achieve low error rates. Hence, there is considerable value in employing methods that minimize computation while retaining adequate classification accuracy. In fact, the naïve Bayes' classifier is able to achieve this in many applications—in particular, those where individual features can be selected that are approximately independent. Features in this category include roundness, size, and redness in the case of oranges.

To understand this, take the expression $p(\mathbf{x}|C_i)P(C_i) = p(x_1, x_2, \ldots, x_N|C_i)\ P(C_i)$ in Eq. (13.4), and reexpress it as appropriate for independent (uncorrelated) features $x_1, x_2, \ldots, x_N$:

$$p(\mathbf{x}|C_i)P(C_i) = p(x_1|C_i)p(x_2|C_i)\ldots p(x_N|C_i)\cdot P(C_i) = \prod_j p(x_j|C_i)\cdot P(C_i) \tag{13.6}$$

This is valid because the overall probability of a set of independent variables is the product of the individual probabilities. First, note that this is a significant simplification of the original general expression. Second, its computation involves only the means and variances of the N individual variables and not the whole $N \times N$ covariance matrix. Clearly, reducing the number of parameters makes the

naïve Bayes' classifier less powerful. However, this is counterbalanced by the fact that, if the same training set is used, the remaining parameters will be much more accurately determined. The result is that, given the right combination of features, the naïve Bayes' classifier can indeed be highly effective in practice.

13.4 RELATION OF THE NEAREST NEIGHBOR AND BAYES' APPROACHES

When Bayes' theory is applied to simple PR tasks, it is immediately clear that a priori probabilities are important in determining the final classification of any pattern, as these probabilities arise explicitly in the calculation. However, this is not so for the NN type of classifier. Indeed, the whole idea of the NN classifier appears to be to get away from such considerations; instead, classifying patterns on the basis of training set patterns that lie nearby in feature space: In this respect, the NN type of classifier seems to fall neatly in the SPR camp. However, there must be a definite answer to the question of whether a priori probabilities are or are not taken into account *implicitly* in the NN formulation, and therefore, whether an adjustment needs to be made to the NN classifier to minimize the error rate. As it is clearly important to have a categorical statement of the situation, the next subsection is devoted to provide such a statement, together with necessary analysis.

13.4.1 MATHEMATICAL STATEMENT OF THE PROBLEM

This subsection considers in detail the relation between the NN algorithm and Bayes' theory. For simplicity (and with no ultimate loss of generality), we here take all dimensions in feature space to have equal weight, so that the measure of distance in feature space is not a complicating factor.

For greatest accuracy of classification, many training set patterns will be used and it will be possible to define a density of training set patterns in feature space, $D_i(\mathbf{x})$, for position $\mathbf{x}$ in feature space and class C_i. Clearly, if $D_k(\mathbf{x})$ is large at position $\mathbf{x}$ in class C_k, then training set patterns lie close together, and a test pattern at $\mathbf{x}$ will be likely to fall in class C_k. More particularly, if

$$D_k(\mathbf{x}) = \max_i D_i(\mathbf{x}) \tag{13.7}$$

then our basic statement of the NN rule implies that the class of a test pattern $\mathbf{x}$ will be C_k.

However, according to the outline given above, this analysis is flawed in not showing explicitly how the classification depends on the a priori probability of class C_k. To proceed, note that $D_i(\mathbf{x})$ is closely related to the conditional probability density $p(\mathbf{x}|C_i)$ that a training set pattern will appear at position $\mathbf{x}$ in

feature space if it is in class C_i. Indeed, the $D_i(\mathbf{x})$ are merely nonnormalized value of the $p(\mathbf{x}|C_i)$:

$$p(\mathbf{x}|C_i) = \frac{D_i(\mathbf{x})}{\int D_i(\mathbf{x})\, d\mathbf{x}} \tag{13.8}$$

The standard Bayes' formulae (Eqs. (13.3) and (13.4)) can now be used to calculate the a posteriori probability of class C_i.

So far, it has been seen that the a priori probability should be combined with the training set density data before valid classifications can be made using the NN rule; as a result, it seems invalid merely to take the nearest training set pattern in feature space as an indicator of pattern class. However, note that when clusters of training set patterns and the underlying within-class distributions scarcely overlap, there is anyway a rather low probability of error in the overlap region, and the result of using $p(\mathbf{x}|C_i)$ rather than $P(\mathbf{x}|C_i)$ to indicate class often introduces only a very small bias in the decision surface. Hence, although invalid *mathematically*, the error introduced need not be disastrous.

We now consider the situation in more detail, finding how the need to multiply by the a priori probability affects the NN approach. In fact, multiplying by the a priori probability can be achieved either *directly*, by multiplying the densities of each class by the appropriate $P(\mathbf{x}|C_i)$, or *indirectly*, by providing a suitable amount of additional training for classes with high a priori probability. It may now be seen that the amount of additional training required is *precisely* the amount that would be obtained if the training set patterns were allowed to appear with their natural frequencies (see equations below). For example, if objects of different classes are moving along a conveyor, we should not first separate them and then train with equal numbers of patterns from each class; we should instead allow them to proceed normally and train on them all at their normal frequencies of occurrence in the training stream. Clearly, if training set patterns do not appear for a time with their proper natural frequencies, this will introduce a bias into the properties of the classifier. Thus, we must make every effort to permit the training set to be representative not only of the *types* of pattern of each class but also of the *frequencies* with which they are presented to the classifier during training.

The following proof of the density-based decision rule (Eqs. (13.3)–(13.4)) may be bypassed on a first reading.

The above ideas for *indirect* inclusion of a priori probabilities may be expressed as follows:

$$P(C_i) = \frac{\int D_i(\mathbf{x})\ d\mathbf{x}}{\sum_j \int D_j(\mathbf{x})\ d\mathbf{x}} \tag{13.9}$$

Hence

$$P(C_i|\mathbf{x}) = \frac{D_i(\mathbf{x})}{\left(\sum_j \int D_j(\mathbf{x}) d\mathbf{x}\right) p(\mathbf{x})} \tag{13.10}$$

where

$$p(\mathbf{x}) = \frac{\sum_k D_k(\mathbf{x})}{\sum_j \int D_j(\mathbf{x})\, \mathrm{d}\mathbf{x}} \tag{13.11}$$

Substituting for $p(\mathbf{x})$ now gives

$$P(C_i|\mathbf{x}) = \frac{D_i(\mathbf{x})}{\sum_k D_k(\mathbf{x})} \tag{13.12}$$

so the decision rule to be applied is to classify an object as class C_i if

$$D_i(\mathbf{x}) > D_j(\mathbf{x}) \quad \text{for all } j \neq i \tag{13.13}$$

The following conclusions have now been arrived at:

1. The NN classifier may well not include a priori probabilities and hence could give a classification bias;
2. It is in general wrong to train a NN classifier in such a way that an equal number of training set patterns of each class are applied;
3. The correct way to train a NN classifier is to apply training set patterns at the natural rates at which they arise in raw training set data.

The third conclusion is perhaps the most surprising and the most gratifying. Essentially, it adds further fire to the principle that training set patterns should be representative of the class distributions from which they are taken, although we now see that it should be generalized to the following: *training sets should be fully representative of the populations from which they are drawn*, where "fully representative" includes ensuring that the frequencies of occurrence of the various classes are representative of those in the whole population of patterns. Phrased in this way, the principle becomes a general one which is relevant to many types of trainable classifier.

13.4.2 THE IMPORTANCE OF THE NEAREST NEIGHBOR ALGORITHM

The NN algorithm is important in being perhaps the simplest of all classifiers to implement on a computer; in addition, it has the advantage of being guaranteed to give an error rate within a factor of two of the ideal error rate (obtainable with a Bayes' classifier). By modifying the method to base classification of any test pattern on the most commonly occurring class amongst the k nearest training set patterns (giving the "k-NN" method), the error rate can be reduced further until it is

arbitrarily close to that of a Bayes' classifier (note that Eq. (13.12) can be interpreted as covering this case too). However, both the NN and (a fortiori) the k-NN methods have the disadvantage that they often require enormous storage to record enough training set pattern vectors, and correspondingly large amounts of computation to search through them to find an optimal match for each test pattern—hence, necessitating the pruning and other methods mentioned earlier for cutting down the load.

Finally, we can conclude that the implicit incorporation of a priori probabilities into the NN and k-NN types of classifier essentially moves them out of the SPR category and into the realm of PPR (see box in Section 13.1).

13.5 THE OPTIMUM NUMBER OF FEATURES

It was stated in Section 13.3 that error rates can be reduced by increasing the number of features used by a classifier, but that there is a limit to this, after which performance actually deteriorates. We here consider why this should happen. Basically, the reason is similar to the situation where many parameters are used to fit a curve to a set of D data points. As the number of parameters P is increased, the fit of the curve becomes better and better, and in general becomes perfect when $P = D$. However, by that stage, the significance of the fit is poor, as the parameters are no longer overdetermined and no averaging of their values is taking place. Essentially, all the noise in the raw input data is being transferred to the parameters. The same thing happens with training set patterns in feature space. Eventually, training set patterns are so sparsely packed in feature space that the test patterns have reduced probability of being nearest to a pattern of the same class, so error rates become very high. This situation can also be regarded as due to a proportion of the features having negligible statistical significance, i.e., they add little additional information and serve merely to add uncertainty to the system.

However, an important factor is that the optimum number of features depends on the amount of training a classifier receives. If the number of training set patterns is increased, more evidence is available to support the determination of a greater number of features and hence to provide more accurate classification of test patterns. Indeed, in the limit of very large numbers of training set patterns, performance continues to increase as the number of features is increased.

This situation was first clarified by Hughes (1968) and verified in the case of n-tuple PR (a variant of the NN classifier due to Bledsoe and Browning, 1959) by Ullmann (1969). Both workers produced clear curves showing the initial improvement in classifier performance as the number of features increased, this improvement being followed by a fall in performance for large numbers of features.

Before leaving this topic, note that the above arguments relate to the number of features that should be used but not to their selection. Clearly, some features are more significant than others, the situation being very data-dependent. It is left

as a topic for experimental tests to determine in any instance which subset of features will minimize classification errors (see also Chittineni, 1980).

13.6 COST FUNCTIONS AND ERROR–REJECT TRADEOFF

In the foregoing sections it has been implied that the main criterion for correct classification is that of maximum a posteriori probability. However, although probability is always relevant, in a practical engineering environment, it can be more important to minimize costs. Hence, it is necessary to compare the costs involved in making correct or wrong decisions. Such considerations can be expressed mathematically by invoking a loss function $L(C_i|C_j)$ that represents the cost involved in making a decision C_i when the true class for feature $\mathbf{x}$ is C_j.

To find a modified decision rule based on minimizing costs, we first define a function known as the conditional risk:

$$R(C_i|\mathbf{x}) = \sum_j L(C_i|C_j)P(C_j|\mathbf{x}) \tag{13.14}$$

This function expresses the expected cost of deciding on class C_i when $\mathbf{x}$ is observed. As it is wished to minimize this function, we decide on class C_i only if:

$$R(C_i|\mathbf{x}) < R(C_i|\mathbf{x}) \text{ for all } j \neq i \tag{13.15}$$

If we were to choose a particularly simple cost function, of the form:

$$L(C_i|C_j) = \begin{cases} 0 \text{ for } i = j \\ 1 \text{ for } i \neq j \end{cases} \tag{13.16}$$

then the result would turn out to be identical to the previous probability-based decision rule, relation (13.5). Clearly, it is only when certain errors lead to relatively large (or small) costs that it pays to deviate from the normal decision rule. Such cases arise when we are in a hostile environment and must, for example, give precedence to the sound of an enemy tank over that of other vehicles—it is better to be oversensitive and risk a false alarm than to retain a small chance of not noticing the hostile agent. Similarly, on a production line, it may in some circumstances be better to reject a small number of good products than to risk selling a defective product. Cost functions therefore permit classifications to be biased in favor of a safe decision in a rigorous, predetermined, and controlled manner, and the desired balance of properties obtained from the classifier.

Another way of minimizing costs is to arrange for the classifier to recognize when it is "doubtful" about a particular classification, because two or more classes are almost equally likely. Then, one solution is to make a safe decision, the decision plane in feature space being biased away from its position for maximum probability classification. An alternative is to reject the pattern, i.e., place it into an "unknown" category; in that case, some other means can be employed for making an appropriate classification. Such a classification could be made by

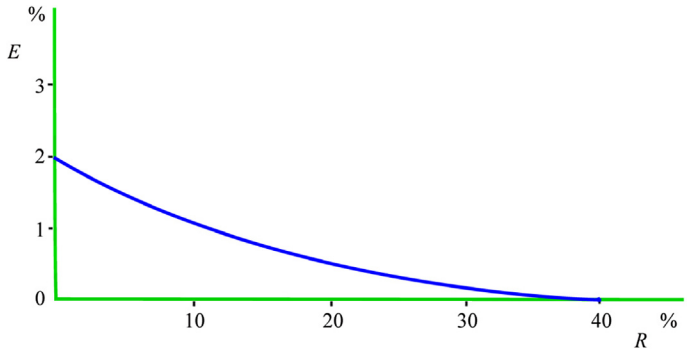

FIGURE 13.4

An error-reject tradeoff curve (*E*, error rate; *R*, reject rate). In this example, the error rate *E* drops substantially to zero for a reject rate *R* of 40%. More usually, *E* cannot be reduced to zero until *R* is 100%.

going back to the original data and measuring further features, but in many cases, it is more appropriate for a human operator to be available to make the final decision. Clearly, the latter approach is more expensive, and so introducing a "reject" classification can incur a relatively large cost factor. A further problem is that the error rate is reduced only by a fraction of the amount that the rejection rate is increased. (Here, all errors and reject rates are assumed to be calculated as proportions of the total number of test patterns to be classified.) Indeed, in a simple two-class system, the initial decrease in error rate is only one half the initial increase in reject rate (i.e., a 1% decrease in error rate is obtained only at the expense of a 2% increase in reject rate), and the situation gets rapidly worse as progressively lower error rates are attempted (Fig. 13.4). Thus, very careful cost analysis of the error−reject tradeoff curve must be made before an optimal scheme can be developed. Finally, note that the overall error rate of the classification system depends on the error rate of the classifier that examines the rejects (e.g., the human operator), and this needs to be taken into account in determining the exact tradeoff to be used.

13.7 SUPERVISED AND UNSUPERVISED LEARNING

In the earlier parts of this chapter, we made the implicit assumption that the classes of all the training set patterns are known, and in addition, that they should be used in training the classifier. Indeed, this assumption might be thought of as inescapable. However, classifiers may actually use two approaches for learning—*supervised learning* (in which the classes are known and used in training) and *unsupervised learning* (in which they are either unknown or else known and not used in training). Unsupervised learning can

frequently be advantageous in practical situations. For example, a human operator is not required to label all the products coming along a conveyor, as the computer can find out for itself both how many classes of product there are and which categories they fall into; in this way considerable operator effort is eliminated. In addition, it is possible that a number of errors would thereby be circumvented. On the contrary, unsupervised learning involves a number of difficulties, as will be seen in the following sections.

Before proceeding, we give two other reasons why unsupervised learning is useful. First, when the characteristics of objects vary with time—for example, beans change in size and color as the season develops—it will be necessary to track these characteristics within the classifier, and unsupervised learning provides a valuable means of approaching this task. Second, when setting up a recognition system, the characteristics of objects, and in particular their most important parameters (e.g., from the point of view of quality control) may well be unknown, and it will be useful to gain some insight into the nature of the data. Thus, types of fault will need to be logged, and permissible variants on objects will need to be noted. As an example, many OCR fonts (such as Times Roman) have a letter "a" with a stroke bent over the top from right to left, though other fonts (such as Monaco) do not have this feature. An unsupervised classifier will be able to flag this up by locating a cluster of training set patterns in a totally separate part of feature space (see Fig. 13.5). In general, unsupervised learning is about the location of clusters in feature space.

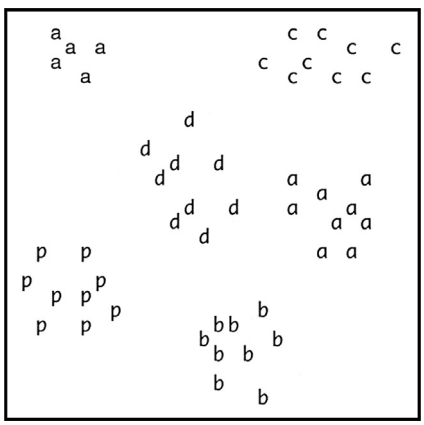

FIGURE 13.5

Location of clusters in feature space. Here, the letters correspond to samples of characters taken from various fonts. The small cluster of a's with strokes bents over the top from right to left appear at a separate location in feature space; this type of deviation should be detectable by cluster analysis.

13.8 CLUSTER ANALYSIS

As indicated above, an important reason for performing cluster analysis is characterization of the input data. However, the underlying motivation is normally to classify test data patterns reliably. To achieve these aims, it will be necessary both to partition feature space into regions corresponding to significant clusters, and to label each region (and cluster) according to the type of data involved. In practice, this can happen in the following two ways:

1. By performing cluster analysis, and then labeling the clusters by specific queries to human operators on the classes of a small number individual training set patterns.
2. By performing supervised learning on a small number of training set patterns, and then performing unsupervised learning to expand the training set to realistic numbers of examples.

In either case, there is ultimately no escape from the need for supervised classification. However, by placing the main emphasis on unsupervised learning, we limit tedium and the possibility of preconceived ideas about possible classes from affecting the final recognition performance.

Before proceeding further, notice that there are cases where we may have absolutely no idea in advance about the number of clusters in feature space; this occurs in classifying the various regions in satellite images. Such cases are in direct contrast with applications such as OCR or recognizing chocolates being placed in a chocolate box.

Cluster analysis involves a number of very significant problems. Not least is the visualization problem. First, in one, two, or even three dimensions, we can easily visualize and decide on the number and location of any clusters, but this capability is misleading; we cannot extend this capability to feature spaces of many dimensions. Second, computers do not visualize as we do, and special algorithms will be needed to enable them to do so. Although computers could be made to emulate our capability in low-dimensional feature spaces, a combinatorial explosion would occur if we attempted this for high-dimensional spaces. This means that we will have to develop algorithms that operate on *lists* of feature vectors, if we are to produce automatic procedures for cluster location.

Available algorithms for cluster analysis fall into two main groups—*agglomerative* and *divisive*. Agglomerative algorithms start by taking the individual feature points (training set patterns, excluding class) and progressively grouping them together according to some similarity function until a suitable target criterion is reached. Divisive algorithms start by taking the whole set of feature points as a single large cluster, and progressively dividing it until some suitable target criterion is reached. Let us assume that there are P feature points. Then, in the worst case, the number of comparisons between pairs of individual feature point

positions which will be required to decide whether to combine a pair of clusters in an agglomerative algorithm will be

$$^{P}C_{2} = \tfrac{1}{2}P(P-1) \tag{13.17}$$

while the number of iterations required to complete the process will be of order $P - K$ (here we are assuming that the final number of clusters to be found is K, where $K \leq P$). On the other hand, for a divisive algorithm, the number of comparisons between pairs of individual feature point positions will be reduced to:

$$^{K}C_{2} = \tfrac{1}{2}K(K-1) \tag{13.18}$$

while the number of iterations required to complete the process will be of order K.

Although it would appear that divisive algorithms require far less computation than agglomerative algorithms, this is not so. This is because any cluster containing p feature points will have to be examined for a huge number of potential splits into subclusters, the actual number being of order:

$$\sum_{q=1}^{p} {}^{p}C_{q} = \sum_{q=1}^{p} \frac{p!(p-q)!}{q!} \tag{13.19}$$

This means that in general, the agglomerative approach will have to be adopted. In fact, the type of agglomerative approach outlined above is exhaustive and rigorous, and a less exacting, iterative approach can be used. First, a suitable number K of cluster centers are set (these can be decided from a priori considerations, or by making arbitrary choices). Second, each feature vector is assigned to the closest cluster center. Third, the cluster centers are recalculated. This process is repeated if any feature points have moved from one cluster to another during the iteration, though termination can also be instituted if the quality of clustering ceases to improve. A basic form of the algorithm originated by Forgy (1965) is given in Table 13.1.

Clearly, the effectiveness of this algorithm will be highly data-dependent—in particular, with regard to the order in which the data points are presented. In addition, the result could be oscillatory or nonoptimal (in the sense of not arriving at the best solution). This could happen if at any stage a single cluster center arose near the center of a pair of small clusters. In addition, the method gives no indication of the most appropriate number of clusters. Accordingly, a number of variant and alternative algorithms have been devised. One such algorithm is the

Table 13.1 Basis of Forgy's Algorithm for Cluster Analysis

```
choose target number K of clusters;
set initial cluster centres;
calculate quality of clustering;
do {
    assign each data point to the closest cluster centre;
    recalculate cluster centres;
    recalculate quality of clustering;
} until no further change in the clusters or the quality of the clusters;
```

Table 13.2 Basis of MacQueen's *K-means* Algorithm

```
choose target number K of clusters;
set the K initial cluster centres at K data points;
for all other data points { // first pass
    assign data point to closest cluster centre;
    recalculate relevant cluster centre;
}
for all data points // second pass
    re-assign data point to closest cluster centre;
```

ISODATA algorithm (Ball and Hall, 1966); this is similar to Forgy's method, but is able to merge clusters which are close together, and to split elongated clusters.

Another disadvantage of iterative algorithms is that it may not be obvious when they should terminate; as a result, they are liable to be too computation intensive. Thus, there has been some support for noniterative algorithms. MacQueen's *K-means* algorithm (MacQueen, 1967) is one of the best known noniterative clustering algorithms: it involves two runs over the data points, one being required to find the cluster centers and the other being required to finally classify the patterns (see Table 13.2). Again, the choice of which data points are to act as the initial cluster centers can be either arbitrary or on some more informed basis.

Noniterative algorithms are, as indicated earlier, very dependent on the order of presentation of the data points. With image data, this is especially problematic, as the first few data points are quite likely to be similar (e.g., all derived from sky or other background pixels). A useful way of overcoming this problem is to randomize the choice of data points, so that they can arise from anywhere in the image. In general, noniterative clustering algorithms are less effective than iterative algorithms, because they are overinfluenced by the order of presentation of the data.

Overall, one of the main problems with the algorithms described above is the lack of indication they give of the most appropriate value of K. However, if a range of possible values for K is known, all of them can be tried, and the one giving the best performance in respect of some suitable target criterion can be taken as providing an optimal result. In that case, we will have found the set of clusters which, in some specified sense, gives the best overall description of the data. Alternatively, some method of analyzing the data to determine K can be used before final cluster analysis: the Zhang and Modestino (1990) approach falls into this category.

Finally, note that none of the above discussion on cluster analysis involves any probabilistic formulation, so the approaches covered in this section are purely those of SPR. However, Chapter 14, Machine Learning: Probabilistic Methods presents new theory and methodology showing how this problem is overcome.

13.9 THE SUPPORT VECTOR MACHINE

The SVM is a new paradigm for SPR, and emerged during the 1990s as an important contender for practical applications. The basic concept relates to linearly separable feature spaces and is illustrated in Fig. 13.6A. The idea is to find the pair

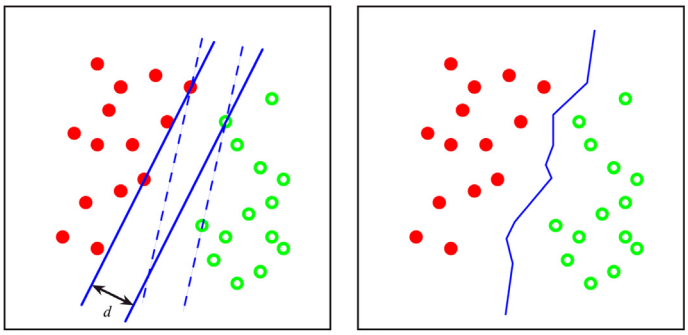

FIGURE 13.6

Principle of the support vector machine. Part (A) shows two sets of linearly separable feature points: the two parallel hyperplanes have the maximum possible separation *d* and should be compared with alternatives such as the pair shown dashed. Part (B) shows the optimal piecewise linear solution that would be found by the nearest neighbor method.

of parallel hyperplanes that lead to the maximum separation between two classes of feature so as to provide the greatest protection against errors. In Fig. 13.6A, the dashed set of hyperplanes has lower separation and thus represents a less ideal choice, with reduced protection against errors. Each pair of parallel hyperplanes is characterized by specific sets of feature points—the so-called support vectors. In the feature space shown in Fig. 13.6A, the planes are fully defined by three support vectors, though clearly this particular value only applies for 2-D feature spaces: in N dimensions, the number of support vectors required is $N + 1$. This provides an important safeguard against overfitting, as, however, many data points exist in a feature space, the maximum number of vectors needed to describe it is $N + 1$.

For comparison, Fig. 13.6B shows the situation that would exist if the NN algorithm were employed. In this case, the protection against errors would be higher, as each position on the separating surface is optimized to the highest local separation distance. However, this increase in accuracy comes at quite high cost in the much larger number of defining example patterns. Indeed, as indicated above, much of the gain of the SVM comes from its use of the smallest possible number of defining example patterns (the support vectors). The disadvantage is that the basic method only works when the dataset is linearly separable.

To overcome this problem, it is usual to transform the training and test data to a feature space of higher dimension where the data do become linearly separable. In fact, this approach will tend to reduce or even eliminate the main advantage of the SVM and lead to overfitting of the data and to poor generalizing ability. However, if the transformation that is employed is nonlinear, the final (linearly separable) feature space could have a manageable number of dimensions, and the advantage of the SVM may not be eroded. Nevertheless, there comes a point where the restriction of linear separability has to be questioned. At that point, it

has been found useful to build "slack" variables s_i into the optimization equations to represent the amount by which the separability constraint can be violated. This is engineered by adding a cost term $C\ \Sigma_i s_i$ to the normal error function: C is adjustable and acts as a regularizing parameter, which is optimized by monitoring the performance of the classifier on a range of training data.

For further information on this topic, the reader should consult either the original papers by Vapnik, including Vapnik (1998), or the specialized text by Cristianini and Shawe-Taylor (2000), or else other texts on SPR, such as Webb (2002).

13.10 ARTIFICIAL NEURAL NETWORKS

The concept of an ANN that could be useful for PR started in the 1950s and continued right through the 1960s. For example, Bledsoe and Browning (1959) developed the "n-tuple" type of classifier that involved bit-wise recording and lookup of binary feature data, leading to the "weightless" or "logical" type of ANN. Although the latter type of classifier maintained a following for some years, there can be little exaggeration in saying that Rosenblatt's "perceptron" (1958, 1962) has had a far greater influence on the subject.

The simple perceptron is a linear classifier that classifies patterns into two classes. It takes a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ as its input, and produces a single scalar output $\sum_{i=1}^{N} w_i x_i$, the classification process being completed by applying a threshold (Heaviside step) function at θ (see Fig. 13.7). The mathematics is simplified by writing $-\theta$ as w_0, and taking it to correspond to an input x_0 which is maintained at a constant value of unity. The output of the linear part of the classifier is then written in the form:

$$d = \sum_{i=1}^{N} w_i x_i - \theta = \sum_{i=1}^{N} w_i x_i + w_0 = \sum_{i=0}^{N} w_i x_i \tag{13.20}$$

and the final output of the classifier is given by:

$$y = f(d) = f\left(\sum_{i=0}^{N} w_i x_i\right) \tag{13.21}$$

This type of neuron can be trained using a variety of procedures, such as the *fixed increment rule* given in Table 13.3. (The original fixed increment rule used a learning rate coefficient η equal to unity.) The basic concept of this algorithm was to try to improve the overall error rate by moving the linear discriminant plane a fixed distance toward a position where no misclassification would occur—but only doing this when a classification error had occurred:

$$w_i(k+1) = w_i(k) \quad y(k) = \omega(k) \tag{13.22}$$

$$w_i(k+1) = w_i(k) + \eta[\omega(k) - y(k)]x_i(k) \quad y(k) \neq \omega(k) \tag{13.23}$$

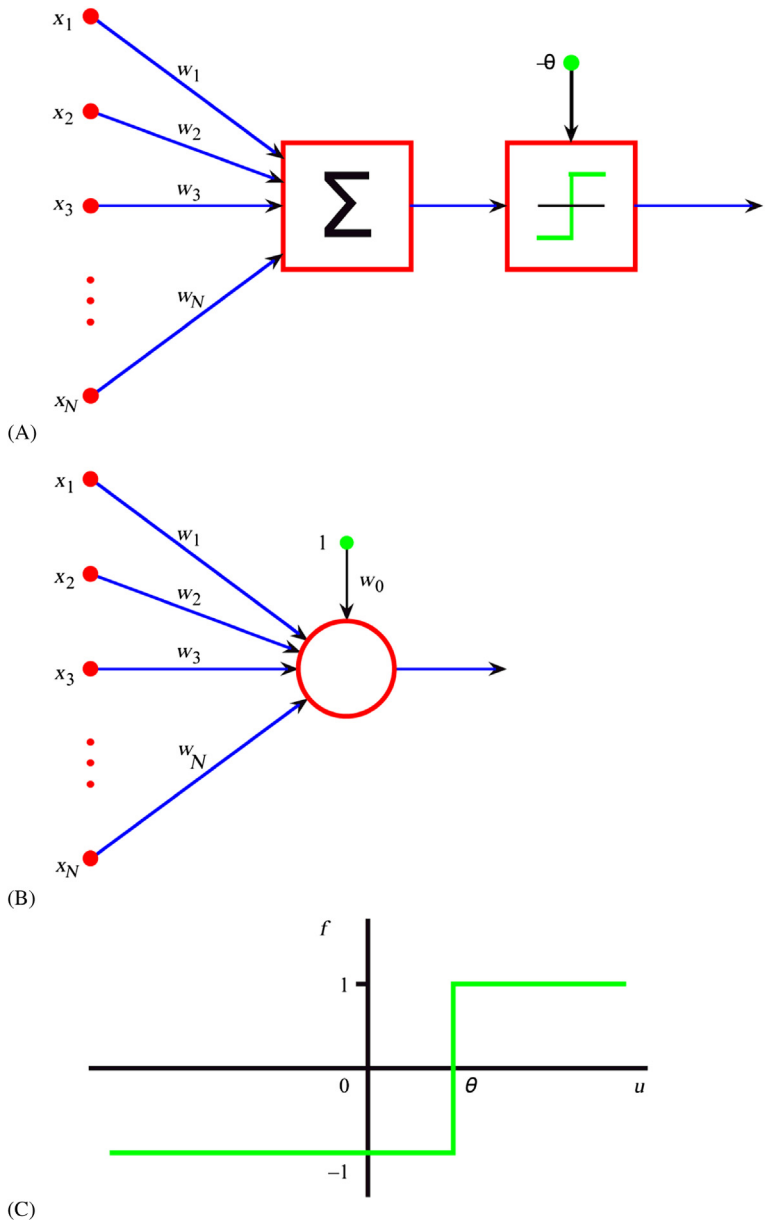

FIGURE 13.7

Simple perceptron. Part (A) shows the basic form of a simple perceptron: input feature values are weighted and summed, and the result fed via a threshold unit to the output connection. Part (B) gives a convenient shorthand notation for the perceptron, and Part (C) shows the activation function of the threshold unit.

Table 13.3 Perceptron *Fixed Increment* Algorithm

```
initialise weights with small random numbers;
select suitable value of learning rate coefficient η in the range 0 to 1;
do {
    for all patterns in the training set {
    obtain feature vector x and class ω;
    compute perceptron output y;
    if (y != ω) adjust weights according to w_i = w_i + η(ω − y)x_i;
   }
} until no further change;
```

In these equations, the parameter k represents the kth iteration of the classifier, and $\omega(k)$ is the class of the kth training pattern. It is clearly important to know whether this training scheme is effective in practice. In fact, it is possible to show that if the algorithm is modified so that its main loop is applied sufficiently many times, *and* if the feature vectors are linearly separable, then the algorithm will converge to a correct error-free solution.

On the contrary, most sets of feature vectors are not linearly separable; thus, it is necessary to find an alternative procedure for adjusting the weights. This is achieved by the Widrow−Hoff delta rule that involves making changes in the weights in proportion to the error $\delta = \omega - d$ made by the classifier. (Note that the error is calculated *before* thresholding to determine the actual class: i.e., δ is calculated using d rather than $f(d)$). Thus, we obtain the Widrow−Hoff delta rule in the form:

$$w_i(k+1) = w_i(k) + \eta\delta x_i(k) = w_i(k) + \eta[\omega(k) - d(k)]x_i(k) \tag{13.24}$$

There are two important ways in which the Widrow−Hoff rule differs from the fixed increment rule:

1. An adjustment is made to the weights whether or not the classifier makes an actual classification error.
2. The output function d used for training is different from the function $y = f(d)$ used for testing.

These differences underline the revised aim of being able to cope with nonlinearly separable feature data. Fig. 13.8 clarifies the situation by appealing to a 2-D case. Fig. 13.8A shows separable data which are straightforwardly fitted by the fixed increment rule. However, the fixed increment rule is not designed to cope with nonseparable data of the type shown in Fig. 13.8B and results in instability during training, and inability to arrive at an optimal solution. On the other hand, the Widrow−Hoff rule copes satisfactorily with this type of data. An interesting addendum to the case of Fig. 13.8A is that although the fixed increment rule apparently reaches an optimal solution, the rule becomes "complacent" once a zero error situation has occurred, whereas an ideal classifier would arrive at a solution which minimizes the probability of error. Clearly, the Widrow−Hoff rule goes some way to solving this problem.

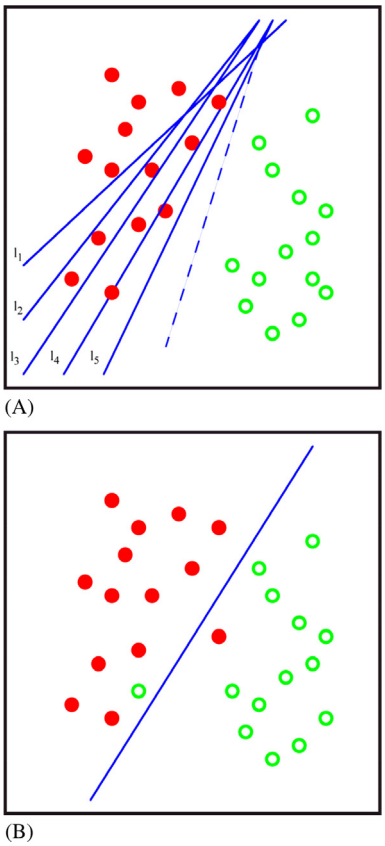

FIGURE 13.8

Separable and nonseparable data. Part (A) shows two sets of pattern data: lines l_1–l_5 indicate possible successive positions of a linear decision surface produced by the fixed increment rule. Note that the latter is satisfied by the final position l_5. The dotted line shows the final position that would have been produced by the Widrow–Hoff delta rule. Part (B) shows the stable position that would be produced by the Widrow–Hoff rule in the case of nonseparable data; in this case, the fixed increment rule would oscillate over a range of positions during training.

So far, we have considered what can be achieved by a simple perceptron. Clearly, though it is only capable of dichotomizing feature data, a suitably trained array of simple perceptrons—the "single-layer perceptron" of Fig. 13.9—should be able to divide feature space into a large number of subregions bounded (in multidimensional space) by hyperplanes. However, in a multiclass application, this approach would require a very large number of simple perceptrons—up to ${}^{c}C_2 = \frac{1}{2}c(c-1)$ for a c-class system. Hence, there is a need to generalize the

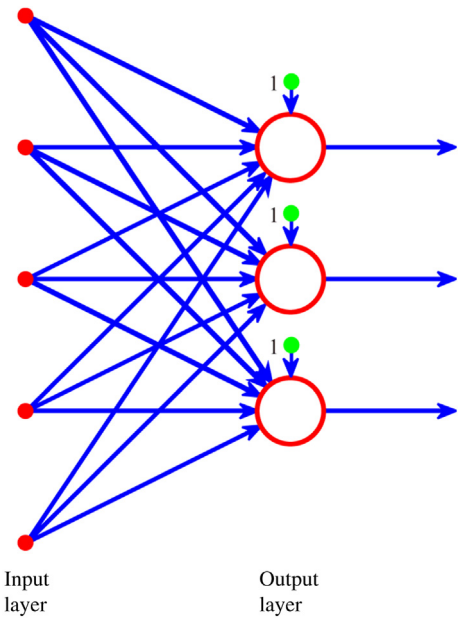

FIGURE 13.9

Single-layer perceptron. The single-layer perceptron employs a number of simple perceptrons in a single layer. Each output indicates a different class (or region of feature space). In more complex diagrams, the bias units (labeled "1") are generally omitted for clarity.

approach by other means. In particular, multilayer perceptron (MLP) networks (see Fig. 13.10)—which would emulate the neural networks in the brain—seem poised to provide a solution as they should be able to recode the outputs of the first layer of simple perceptrons.

Rosenblatt himself proposed such networks, but was unable to find a general means for training them systematically. In 1969, Minsky and Papert published their famous monograph, and in discussing the MLP raised the specter of "the monster of vacuous generality;" they drew attention to certain problems that apparently would never be solved using MLPs. For example, diameter-limited perceptrons (those that view only small regions of an image within a restricted diameter) would be unable to measure large-scale connectedness within images. These considerations discouraged effort in this area, and for many years, attention was diverted to other areas such as expert systems. It was not until 1986 that Rumelhart *et al.* were successful in proposing a systematic approach to the training of MLPs. Their solution is known as the back-propagation algorithm.

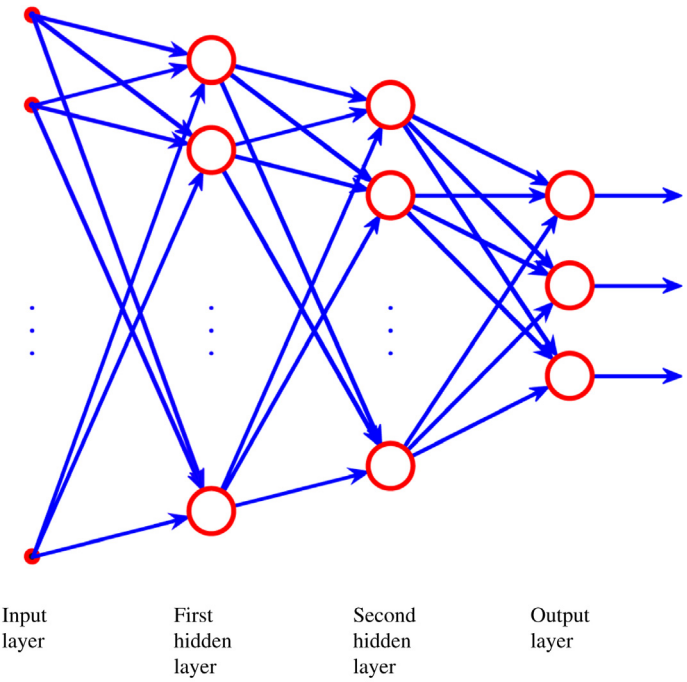

FIGURE 13.10

Multilayer perceptron. The multilayer perceptron employs several layers of perceptrons. In principle, this topology permits the network to define more complex regions of feature space and thus performs much more precise pattern recognition tasks. Finding systematic means of training the separate layers becomes the vital issue. For clarity, the bias units have been omitted from this and later diagrams.

13.11 THE BACK-PROPAGATION ALGORITHM

The problem of training a MLP can be simply stated: a general layer of a MLP obtains its feature data from the lower layers and receives its class data from higher layers. Hence, if all the weights in the MLP are potentially changeable, the information reaching a particular layer cannot be relied upon. There is no reason why training a layer in isolation should lead to overall convergence of the MLP toward an ideal classifier (however defined). Although it might be thought that this is a rather minor difficulty, in fact, this is not so; indeed, this is but one example of the so-called credit assignment problem. This is probably not a good first example by which to define the credit assignment problem (in this case, it would appear to be more of a *deficit* assignment problem). The credit assignment problem is the problem of correctly determining the local origins of global

properties and making the right assignments of rewards, punishments, corrections, and so on, thereby permitting the whole system to be optimized systematically.

One of the main difficulties in predicting the properties of MLPs and hence of training them reliably is the fact that neuron outputs swing suddenly from one state to another as their inputs change by infinitesimal amounts. Hence, we might consider removing the thresholding functions from the lower layers of MLP networks to make them easier to train. On the contrary, this would result in these layers acting together as larger linear classifiers, with far less discriminatory power than the original classifier (in the limit we would have a single linear classifier with a single thresholded output connection, so the overall MLP would act as a single-layer perceptron).

The key to solving to these problems was to modify the perceptrons composing the MLP by giving them a less "hard" activation function than the Heaviside function. As we have seen, a linear activation function would be of little use, but one of "sigmoid" shape, such as the tanh function (Fig. 13.11) is effective, and indeed is almost certainly the most widely used of the available functions. (We do not here make a marked distinction between symmetrical activation functions and alternatives that are related to them by shifts of axes, though the symmetrical formulation seems preferable as it emphasizes bidirectional functionality. In fact, the tanh function, which ranges from -1 to 1, can be expressed in the form: $\tanh u = (e^u - e^{-u})/(e^u + e^{-u}) = 1 - 2/(1 + e^{2u})$ and is thereby closely related to the commonly used function $(1+e^{-v})^{-1}$. It can now be deduced that the latter function is symmetrical, though it ranges from 0 to 1 as v goes from $-\infty$ to ∞.)

Once these softer activation functions were used, it became possible for each layer of the MLP to "feel" the data more precisely and thus training procedures could be set up on a systematic basis. In particular, the rate of change of the data at each individual neuron could be communicated to other layers that could then be trained appropriately—though only on an incremental basis. We shall not go through the detailed mathematical procedure, or proof of convergence, beyond stating that it is equivalent to energy minimization and gradient descent on a (generalized) energy surface. Instead, we give an outline of the backpropagation algorithm (see Table 13.4). Nevertheless, some notes on the algorithm are in order:

1. The outputs of one node are the inputs of the next, and an arbitrary choice is made to label all variables as output (y) parameters rather than as input (x) variables; all output parameters are in the range 0 to 1.
2. The class parameter ω has been generalized as the target value t of the output variable y.
3. For all except the final outputs, the quantity δ_j has to be calculated using the formula $\delta_j = y_j(1 - y_j)(\sum_m \delta_m w_{jm})$, the summation having to be taken over all the nodes in the layer *above* node j.
4. The sequence for computing the node weights involves starting with the output nodes and then proceeding downwards one layer at a time.

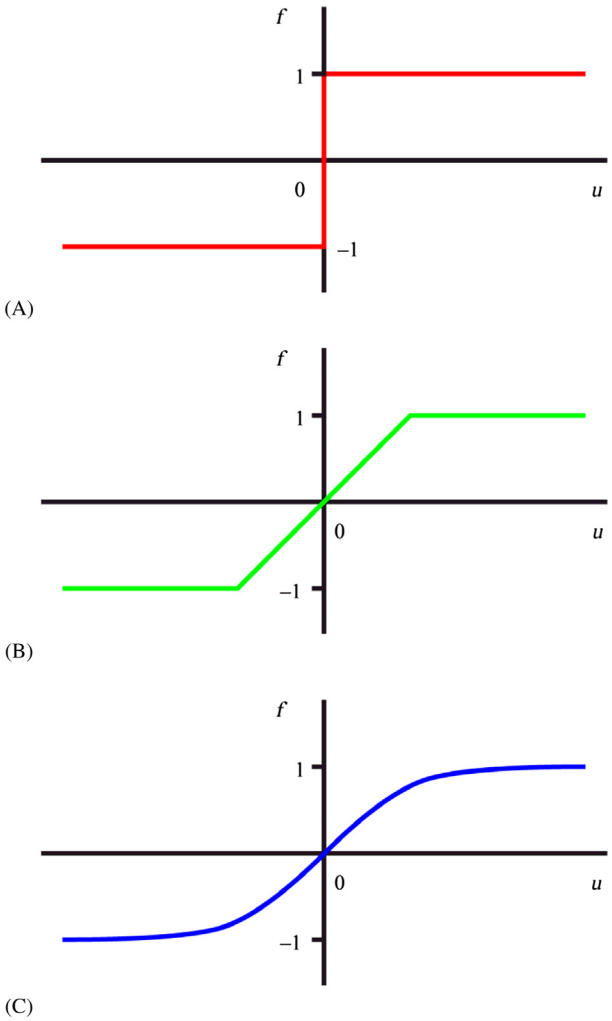

FIGURE 13.11

Symmetric activation functions. This figure shows a series of symmetric activation functions. Part (A) shows the Heaviside activation function used in the simple perceptron. Part (B) shows a linear activation function, which is, however, limited by saturation mechanisms. Part (C) shows a sigmoidal activation function which approximates to the hyperbolic tangent function.

5. If there are no hidden nodes, the formula reverts to the Widrow−Hoff delta rule, except that the input parameters are now labeled y_i, as indicated above.
6. It is important to initialize the weights with random numbers to minimize the chance of the system becoming stuck in some symmetrical state from which it might be difficult to recover.

Table 13.4 The Back-Propagation Algorithm

```
initialise weights with small random numbers;
select suitable value of learning rate coefficient η in the range 0 to 1;
do {
    for all patterns in the training set
        for all nodes j in the MLP {
            obtain feature vector x and target output value t;
            compute MLP output y;
            if (node is in output layer)
                  δj = yj(1 − yj)(tj − yj);
            else δj = yj(1 − yj)(Σmδmwjm);
            adjust weights i of node j according to wij = wij + ηδjyi;
        }
} until changes are reduced to some predetermined level;
```

7. Choice of value for the learning rate coefficient η will be a balance between achieving a high rate of learning and avoidance of overshoot: normally a value of around 0.8 is selected.

When there are many hidden nodes, convergence of the weights can be very slow, and indeed, this is one disadvantage of MLP networks. Many attempts have been made to speed convergence, and a method that has been very widely used is to add a "momentum" term to the weight update formula, it being assumed that weights will change in a similar manner during iteration k to the change during iteration $k - 1$:

$$w_{ij}(k+1) = w_{ij}(k) + \eta\delta_j y_i + \alpha[w_{ij}(k) - w_{ij}(k-1)] \tag{13.25}$$

where α is the momentum factor. This technique is primarily intended to prevent networks from becoming stuck at local minima of the energy surface.

13.12 MULTILAYER PERCEPTRON ARCHITECTURES

The preceding sections gave the motivation for designing a MLP and for finding a suitable training procedure, and then outlined a general MLP architecture and the widely used back-propagation training algorithm. However, having a general solution is only one part of the answer. The next question is how best to adapt the general architecture to specific types of problem. We shall not give a full answer to this question here. However, Lippmann attempted to answer this problem in 1987. He showed that a two-layer (single hidden layer) MLP can implement arbitrary convex decision boundaries and indicated that a three-layer (two-hidden layer) network is required to implement more complex decision boundaries. It was subsequently found that it should never be necessary to exceed two hidden layers, as a three-layer network can tackle quite general situations if sufficient neurons are used (Cybenko, 1988). Subsequently, Cybenko (1989) and Hornik *et al.* (1989)

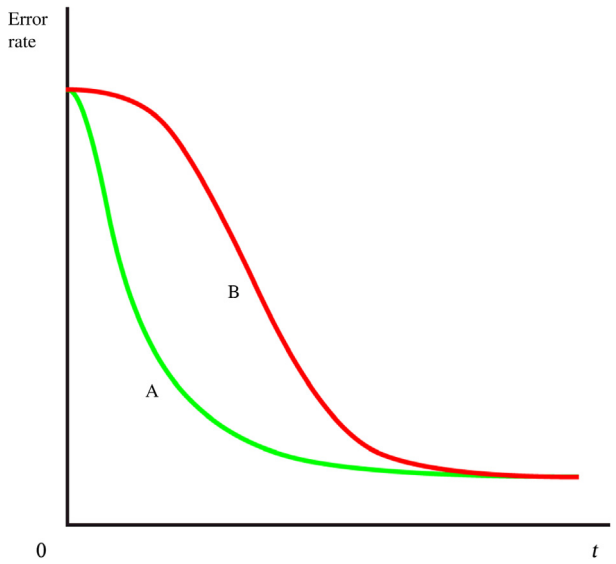

FIGURE 13.12

Learning curve for the multilayer perceptron. Here, (A) shows the learning curve for a single-layer perceptron, and (B) shows that for a multilayer perceptron. Note that the multilayer perceptron takes considerable time to get going, as initially each layer receives relatively little useful training information from the other layers. Note also that the lower part of the diagram has been idealized to the case of identical asymptotic error rates—though this situation would seldom occur in practice.

showed that a two-layer MLP can approximate any continuous function, though nevertheless there may sometimes be advantages in using more than two layers.

Although the back-propagation algorithm can train MLPs containing any number of layers, in practice, training one layer "through" several others introduces an element of uncertainty that is commonly reflected in increased training times (see Fig. 13.12). Thus, there is some advantage to be gained from using a minimal number of layers of neurons. In this context, the above findings on the necessary numbers of hidden layers have proved to be valuable.

13.13 OVERFITTING TO THE TRAINING DATA

When training MLPs and many other types of ANN, there is a problem of overfitting the network to the training data. One of the fundamental aims of SPR is for the learning machine to be able to generalize from the particular set of data it is trained on to other types of data it might meet during testing. In particular, the machine should be able to cope with noise, distortions, and fuzziness in the data,

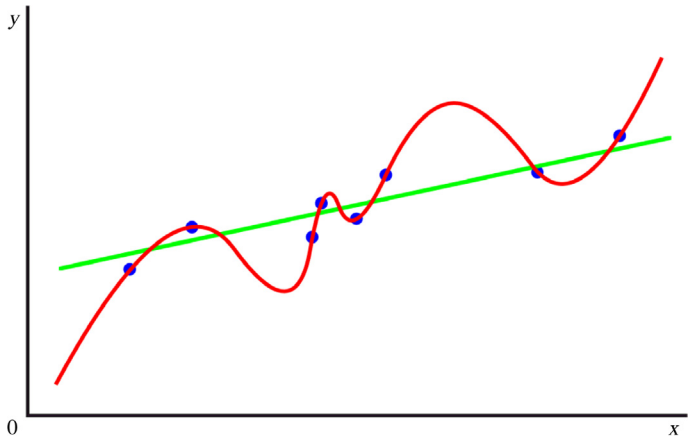

FIGURE 13.13

Overfitting of data. In this graph, the data points are rather too well fitted by the red curve, which matches every nuance exactly. Unless there are strong theoretic reasons why the red curve should be used, the green line will give a higher confidence level.

though clearly not to the extent of being able to respond correctly to types of data different from those on which it has been trained. The main points to be made here are (1) that the machine should learn to respond to the underlying population from which the training data have been drawn, and (2) that it must not be so well adapted to the specific training data that it responds less well to other data from the same population. Fig. 13.13 shows in a 2-D case both a fairly ideal degree of fit, and a situation where every nuance of the set of data has been fitted, thereby achieving a degree of overfit.

Typically, overfitting can arise if the learning machine has more adjustable parameters than are strictly necessary for modeling the training data: with too few parameters such a situation should not arise. However, if the learning machine has enough parameters to ensure that relevant details of the underlying population are fitted, there may be overmodeling of part of the training set; thus, the overall recognition performance will deteriorate. Ultimately, the reason for this is that recognition is a delicate balance between capability to discriminate and capability to generalize, and it is unlikely that any complex learning machine will get the balance right for all the features it has to take account of.

Be this as it may, we clearly need to have some means of preventing overadaptation to the training data. One way of achieving this is to curtail the training process before overadaptation can occur. (It is often stated that this procedure aims to prevent overtraining. However, the term "overtraining" is ambiguous. On the one hand, it can mean recycling through the *same* set of training data until eventually the learning machine is overadapted to it. On the other hand, it can mean using more and more *totally new* data—a procedure that cannot produce

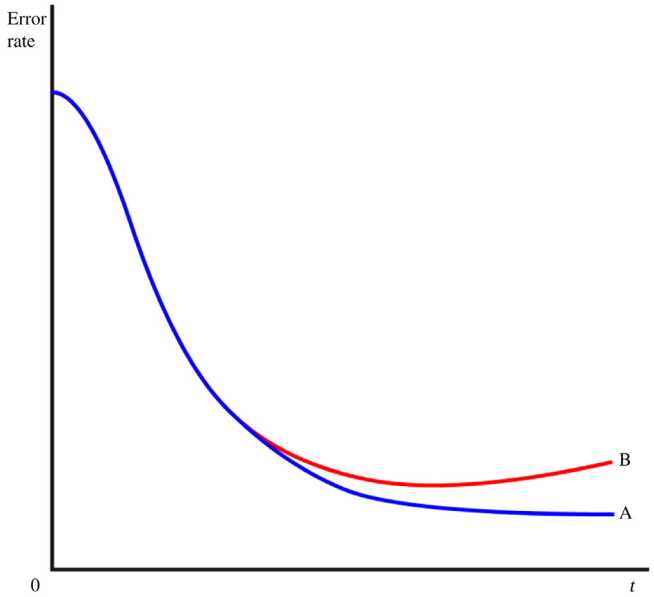

FIGURE 13.14

Cross-validation tests. This diagram shows the learning curve for a multilayer perceptron (A) when tested on the training data and (B) when tested on a special validation set. Part (A) tends to go on improving even when overfitting is occurring. However, this situation is detected when (B) starts deteriorating. To offset the effects of noise (not shown on curves (A) and (B)), it is usual to allow 5%–10% deterioration relative to the minimum in (B).

overadaptation to the data, and on the contrary is almost certain to improve performance. In view of this ambiguity, it seems better not to use the term.) In fact, curtailing the training process is not difficult to manage: we merely need to test the system periodically during training to ensure that the point of overadaptation has not been reached. Fig. 13.14 shows what happens when testing is carried out simultaneously on a separate dataset: at first, performance on the test data closely matches that on the training data, being slightly superior for the latter because a small degree of overadaptation is already occurring. But after a time, performance starts deteriorating on the test data, whereas that on the training data appears to go on improving. This is the point where serious overfitting is occurring, and the training process needs to be curtailed. The aim, then, is to make the overall training process far more rigorous by splitting the original training set into two parts—the first being retained as a normal training set, and the second being called the *validation set*. Note that the latter is actually part of the training set in the sense that it is not part of the eventual test set.

The process of checking the degree of training by use of a validation set is called *cross-validation* and is vitally important to proper use of an ANN.

The training algorithm should include cross-validation as a fully integrated part of the whole training schedule: it should not be regarded as an optional extra.

It is useful to speculate how overadaptation could occur when the training procedure is completely determined by the back propagation (or other) provably correct algorithm. In fact, there are specific mechanisms by which overadaptation can occur. For example, when the training data do not control particular weights sufficiently closely, some could drift to large positive or negative values, while retaining a sufficient degree of cancellation so that no problems appear to arise with the training data; yet, when test or validation data are employed, the problems become all too clear. The fact that the form of the sigmoid function will permit some nodes to become "saturated out" does not help the situation, as it inactivates parameters and hides certain aspects of the incoming data. Yet, it is intrinsic to the MLP architecture and the way it is trained that some nodes are *intended* to be saturated out in order to ignore irrelevant features of the training set. The problem is whether inactivation is inadvertent or designed. The answer probably lies in the quality of the training set and how well it covers the available or potential feature space.

Finally, let us suppose a MLP is being set up, and it is initially unknown how many hidden layers will be required or how many nodes there will have to be in each layer; it will also be unknown how many training set patterns will be required or how many training iterations will be needed—or what values of the momentum or learning parameters will be appropriate. A quite substantial number of tests will be required to decide all the relevant parameters. There is therefore a definite risk that the final system will be overadapted not only to the training set but also to the validation set. In such circumstances, what we need is a second validation set that can be used after the whole network has been finalized and final training is being undertaken.

13.14 CONCLUDING REMARKS

The methods of this chapter make it rather surprising that so much of image processing and analysis is possible without any reference to a priori probabilities. It seems likely that this situation is due to the fact that algorithms are designed and implemented by humans who have knowledge of the types of input data and thereby incorporate a priori probabilities implicitly, e.g., via the application of suitable threshold values. Nonetheless, SPR is extremely valuable within its own range of utility. This includes identifying objects on conveyors and making value judgments of their quality, reading labels and codes, verifying signatures, checking fingerprints, and so on. Indeed, the number of distinct applications of SPR is huge, and it forms an essential counterpart to the other methods described in this book.

This chapter has concentrated mainly on the supervised learning approach to SPR. However, unsupervised learning is also vitally important, particularly when training on huge numbers of samples (e.g., in a factory environment) is involved. The section on this topic should therefore not be ignored as a minor, insignificant perturbation.

In summary, the only methods of this chapter that transcend SPR and include probabilistic analysis are NN-based methods (though only when suitably trained) and of course Bayes theory. In Chapter 14, Machine Learning: Probabilistic Methods, it will be seen how new methods can be developed that incorporate probability as an intrinsic part of their design.

Vision is largely a recognition process with both structural and statistical aspects. This chapter has reviewed SPR, emphasizing fundamental classification error limits, and has shown the part played by Bayes' theory, the NN algorithm and ANNs. Note that the last of these is subject to the same limitations as other SPR methods, particularly with regard to adequacy of training and the possibly of overfitting. Further methods incorporating probability as part of their formulation are left to later chapters, notably Chapter 14, Machine Learning: Probabilistic Methods.

13.15 BIBLIOGRAPHICAL AND HISTORICAL NOTES

Although the subject of SPR tends not to be at the center of attention in image analysis work, it provides an important background—especially in the area of automated visual inspection where decisions continually have to be made on the adequacy of products. (Note, however, that SPR is vital to the analysis of multispectral data from satellite imagery: see for example, Landgrebe (1981).) Most of the relevant works on this topic were already in place by the early 1970s, including the work of Hughes (1968) and Ullmann (1969) relating to the optimum number of features to be used in a classifier. At that stage, a number of important volumes appeared—see for example, Duda and Hart (1973) and Ullmann (1973), and these were followed a little later by Devijver and Kittler (1982).

In fact, the use of SPR for image interpretation dates from the 1950s. For example, in 1959, Bledsoe and Browning developed the n-tuple method of PR, which turned out (Ullmann, 1973) to be a form of NN classifier; however, it has been useful in leading to a range of simple hardware machines based on RAM (n-tuple) lookups (see e.g., Aleksander *et al.*, 1984), thereby demonstrating the importance of marrying algorithms and readily implementable architectures.

Many of the most important developments in this area have probably been those comparing the detailed performance of one classifier with another, particularly with respect to cutting down the amount of storage and computational effort. Papers in these categories include those by Hart (1968) and Devijver and Kittler (1980). Oddly, there appeared to be no overt mention in the literature of how

a priori probabilities should be used with the NN algorithm, until the author's paper on this topic (Davies, 1988e): see Section 13.4.

On the unsupervised approach to SPR, Forgy's (1965) method for clustering data was soon followed by the famous ISODATA approach of Ball and Hall (1966), and then by MacQueen's (1967) *K-means* algorithm. Much related work ensued, and this was summarized by Jain and Dubes (1988), which became a classic text. However, cluster analysis is an exacting process and various workers have felt the need to push the subject further forward: e.g., Postaire and Touzani (1989) required more accurate cluster boundaries; Jolion and Rosenfeld (1989) wanted better detection of clusters in noise; Chauduri (1994) needed to cope with time-varying data; and Juan and Vidal (1994) required faster *K-means* clustering. Note that all this work can be described as conventional and did not involve the use of robust statistics per se. However, elimination of outliers is central to the problem of reliable cluster analysis: for a discussion of this aspect of the problem, see Appendix A and the references listed therein.

Although the field of PR has moved forward substantially since 1990, there are fortunately several quite recent texts that cover the subject relatively painlessly (Duda *et al.*, 2001; Webb, 2002; Theodoridis and Koutroumbas, 2009).

SVMs also came into prominence over the 1990s and have found an increasing number of applications: the concept was invented by Vapnik and the historical perspective is covered in Vapnik (1998). Cristianini and Shawe-Taylor (2000) provide a student-orientated text on the subject.

Next, we turn our attention to the history of ANNs. After a promising start in the 1950s and 1960s, they fell into disrepute (or at least, disregard) following the pronouncements of Minsky and Papert in 1969; they picked up again in the early 1980s, were subjected to an explosion in interest after the announcement of the backpropagation algorithm by Rumelhart *et al.* in 1986, and in the mid-1990s settled into the role of normal tools for vision and other applications. Note that the backpropagation algorithm was invented several times (Werbos, 1974; Parker, 1985) before its relevance was finally recognized. In parallel with these MLP developments, Oja (1982) developed his Hebbian principal components network. Useful early references on ANNs include the volumes by Haykin (1999) and Bishop (1995), and papers on their application to segmentation and object location, such as Toulson and Boyce (1992) and Vaillant *et al.* (1994); for work on contextual image labeling, see Mackeown *et al.* (1994).

After the euphoria of the early 1990s, during which papers on ANNs applied to vision were ubiquitous, it was seen that the main value of ANNs lay in their unified approach to feature extraction and selection (even if this necessarily carries the disadvantage that the statistics are hidden from the user), and their intrinsic capability for finding moderately nonlinear solutions with relative ease. Later papers include the ANN face detection work of Rowley *et al.* (1998), amongst others (Fasel, 2002; Garcia and Delakis, 2002). For further general information on ANNs, see the book by Bishop (2006).

13.15.1 MORE RECENT DEVELOPMENTS

Returning to mainstream SPR, Jain (2010) presented a review of the subject of clustering, entitled "Data clustering: 50 years beyond *K-means*". He noted that "In spite of the fact that *K-means* was proposed over 50 years ago and thousands of clustering algorithms have been published since then, *K-means* is still widely used"—thereby reflecting the difficulty of designing a general purpose clustering algorithm and the ill-posed nature of the problem; emerging and useful research directions include semisupervised clustering and ensemble clustering. The review presents the main challenges and issues facing the subject as of 2010: above all is the plea for a suite of benchmark data with ground truth to test and evaluate clustering methods.

Youn and Jeong (2009) describe a class-dependent feature-scaling method employing a naïve Bayes' classifier for text data mining, including functions such as text categorization and search. Although the reasons why the naïve Bayes independence assumption works well in many cases have not been well explained or understood until recently, this paper confirms that it is often a good choice for text analysis because the amount of data used is large (e.g., the number of features is about 100,000 for protein sequence data). In particular, the simplicity and the effectiveness of the naïve Bayes' classifier maps well to text categorization. Rish (2001) provides an empirical study of naïve Bayes, containing much useful information.

13.16 PROBLEMS

1. Show that if the cost function of Eq. (13.16) is chosen, then the decision rule (13.15) can be expressed in the form of relation (13.5).
2. Show that in a simple two-class system, introducing a reject classification to reduce the number of errors by R in fact requires $2R$ test patterns to be rejected, assuming that R is small. What is likely to happen as R increases?
3. In a 1-D feature space, show that the feature value at which the a posteriori probability curves cross over corresponds to a decision boundary giving the minimum possible classification error.