CHAPTER

24 Statistical Pattern Recognition

Pattern recognition is a task that humans are able to achieve "at a glance" with little apparent effort. Much of pattern recognition is structural, being achieved essentially by analyzing shape. In contrast, statistical pattern recognition (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 algorithm—probably the most intuitive of all statistical pattern recognition techniques.
- Bayes' theory, which forms the ideal minimum error classification system.
- the relation linking the nearest neighbor method to Bayes' theory.
- the reason why the optimum number of features will always be finite.
- the receiver operating characteristic (ROC) curve, which allows an optimum balance between false positives and false negatives to be achieved.
- the distinction between supervised and unsupervised learning.
- the method of principal components analysis and its value.
- how artificial neural networks 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 pattern recognition methods and many other relevant techniques—as already indicated in the title to Part 4.

24.1 INTRODUCTION

The earlier chapters of this book have tackled the task of interpreting images on the basis that when suitable cues have been found, the positions of various objects and their identities will emerge in a natural and straightforward way. When the objects that appear in 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, 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 has already been seen in Part 3. Indeed, the very ambiguity involved in interpreting the 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 specific applications the problems tend to be much more restricted and simpler to solve; e.g., only widgets need to be inspected on a widget line, and only vehicles need to be scrutinized on a highway.

However, the fact that it is often expedient to start by searching for cues means that the vision system could lock on to erroneous interpretations. For example, HT-based methods lead to interpretations based on the most prominent peaks in parameter space, while the maximal clique approach supports interpretations based on the largest number of feature matches between parts of an image and the object model. Clearly, noise, background clutter, and other artifacts make it possible that an object will be seen (or its presence hypothesized) when none is there, whereas occlusions, shadows, etc. may cause an object to be missed altogether.

Hence, a rigorous interpretation strategy would ideally try to find all possible feature and object matches, and should have some means of distinguishing between them. An attractive idea is that of not being satisfied with an interpretation until the *whole* of any image has been understood. However, enough has been seen in previous chapters (e.g., Chapter 13) to demonstrate that the computational load of such a strategy will generally make it impracticable for real scenes. Yet there are more constrained types of situation in which the strategy is worth considering and 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 SPR.

The following sections study some of the important 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 three decades and has settled down sufficiently so that an overview chapter can serve a useful purpose. The reader is referred to several existing texts

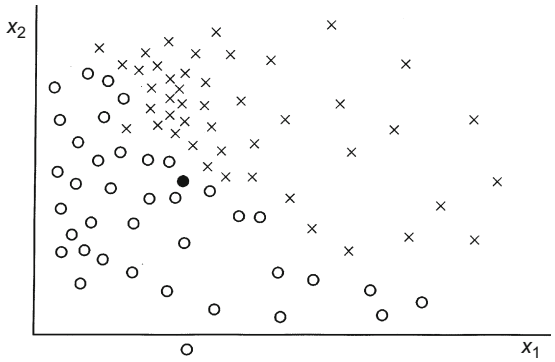

FIGURE 24.1

Principle of the nearest neighbor algorithm for a two-class problem. ○, class 1 training set patterns; ×, class 2 training set patterns; ●, test pattern.

for further details (Duda et al., 2001; Webb, 2002). We start by describing the nearest neighbor (NN) approach to SPR and then go on to consider Bayes' decision theory, which provides a more general model of the underlying process.

24.2 THE NEAREST NEIGHBOR ALGORITHM

The principle of the nearest neighbor algorithm is that of comparing input image[1] patterns against a number of paradigms and then classifying them according to the class of the paradigm that gives the closest match (Fig. 24.1). 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 arising when (a) training patterns of different classes are close together in Hamming distance (i.e., they differ in too few bits to be readily distinguishable) and (b) minor translations, rotations, or noise cause variations that inhibit accurate recognition. More generally, problem (b) 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 (a) implies that patterns of two different classes may in some cases be so close as to be

[1]Note that a number of the methods discussed in this chapter are very general and can be applied to the recognition of widely different datasets, including, e.g., speech and electrocardiograph waveforms.

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.

The example of Fig. 1.1 is rather trivial but nevertheless carries important lessons. Note that general images have many more pixels than that in Fig. 1.1 and also are not merely binary. However, since 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 Chapters 1 and 9). This gives (a) the numbers of nodes and limbs of various types, (b) the lengths and relative orientations of limbs, and perhaps (c) 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 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. 24.1. In the general case, the distance in feature space is no longer Hamming distance but some more general measure such as Mahalanobis distance (Duda and Hart, 1973). In fact, a problem arises since 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 it 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.

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 among these is that of pruning the training set by eliminating patterns that are not near the boundaries of class regions in feature space, since 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 analogs 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. 24.2). This may be achieved either

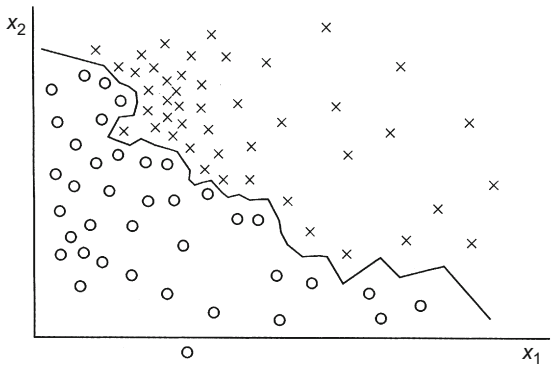

FIGURE 24.2

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.

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 approach is in many ways more attractive, since 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—since this underpins all the possibilities thrown up by the NN method and its derivatives.

24.3 BAYES' DECISION THEORY

The basis of Bayes' decision theory is examined in this section. 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:

$$P(C_i|x) = \frac{p(x|C_i)P(C_i)}{p(x)} \tag{24.1}$$

where

$$p(x) = \sum_j p(x|C_j)P(C_j) \tag{24.2}$$

Mathematically, the variables here are (a) the *a priori* probability of class C_i, $P(C_i)$; (b) the probability density for feature x, $p(x)$; (c) 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 (d) 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 as they move along a conveyor. As before, such a sequence of objects is called the 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 numbers of holes it possesses, and so on. 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. 24.3); 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 24.5.

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

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{p(\mathbf{x})} \tag{24.3}$$

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 $\mathbf{x}$:

$$p(\mathbf{x}) = \sum_j p(\mathbf{x}|C_j)P(C_j) \tag{24.4}$$

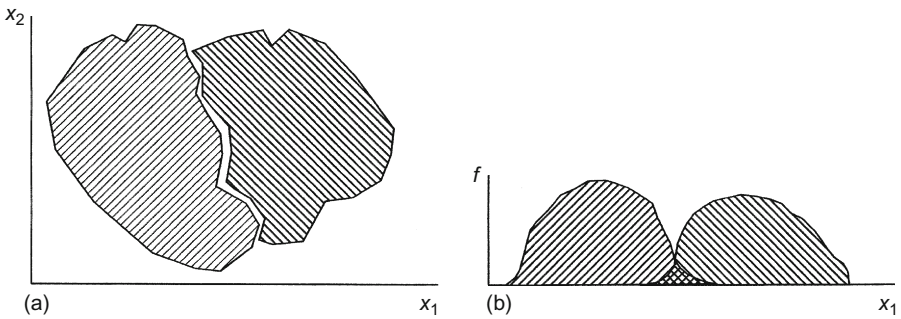

FIGURE 24.3

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.

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:

$$P(C_i|\mathbf{x}) > P(C_j|\mathbf{x}) \quad \text{for all } j \neq i \tag{24.5}$$

24.3.1 The Naive Bayes' Classifier

Many classification methods, including the nearest neighbor method 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 naive Bayes' classifier is able to achieve this in many applications—particularly 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. (24.4), and re-express it as appropriate for independent (uncorrelated) features $x_1, x_2, \ldots, x_N$:

$$p(\mathbf{x}|C_i)P(C_i) = p(x_1|C_i)p(x_2|C_i)\ldots p(x_N|C_i)\cdot P(C_i) = \prod_j p(x_j|C_i)\cdot P(C_i) \tag{24.6}$$

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 naive 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 naive Bayes' classifier can indeed be highly effective in practice.

24.4 RELATION OF THE NEAREST NEIGHBOR AND BAYES' APPROACHES

When Bayes' theory is applied to simple pattern recognition tasks, it is immediately clear that *a priori* probabilities are important in determining the final classification of any pattern, since 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. 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 of whether an adjustment needs to be made to the NN classifier to minimize the error rate. Since it is clearly important to have a categorical statement of the situation, Section 24.4.1 is devoted to providing such a statement together with necessary analysis.

24.4.1 Mathematical Statement of the Problem

This section 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 high 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:

$$D_k(\mathbf{x}) = \max_i D_i(\mathbf{x}) \tag{24.7}$$

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 non-normalized values of the $p(\mathbf{x}|C_i)$:

$$p(\mathbf{x}|C_i) = \frac{D_i(\mathbf{x})}{\int D_i(\mathbf{x})\mathrm{d}\mathbf{x}} \tag{24.8}$$

The standard Bayes' formulae (Eqs. (24.3) and (24.4)) 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 (24.9)−(24.13)). 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 above ideas for *indirect* inclusion of *a priori* probabilities may be expressed as follows:

$$P(C_i) = \frac{\int D_i(\mathbf{x})\mathrm{d}\mathbf{x}}{\sum_j \int D_j(\mathbf{x})\mathrm{d}\mathbf{x}} \tag{24.9}$$

Hence

$$P(C_i|\mathbf{x}) = \frac{D_i(\mathbf{x})}{\left(\sum_j \int D_j(\mathbf{x})\mathrm{d}\mathbf{x}\right) p(\mathbf{x})} \tag{24.10}$$

where

$$p(\mathbf{x}) = \frac{\sum_k D_k(\mathbf{x})}{\sum_j \int D_j(\mathbf{x})\mathrm{d}\mathbf{x}} \tag{24.11}$$

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

$$P(C_i|\mathbf{x}) = \frac{D_i(\mathbf{x})}{\sum_k D_k(\mathbf{x})} \tag{24.12}$$

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

$$D_i(\mathbf{x}) > D_j(\mathbf{x}) \quad \text{for all } j \neq i \tag{24.13}$$

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 an 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 an 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.

24.4.2 The Importance of the Nearest Neighbor Classifier

The NN classifier 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 among 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. (24.12) 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.

24.5 THE OPTIMUM NUMBER OF FEATURES

It has been stated in Section 24.3 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, since 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 pattern recognition (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 case which subset of features will minimize classification errors (see also Chittineni, 1980).

24.6 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:

$$R(C_i|\mathbf{x}) = \sum_j L(C_i|C_j)P(C_j|\mathbf{x}) \tag{24.14}$$

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:

$$R(C_i|\mathbf{x}) < R(C_j|\mathbf{x}) \quad \text{for all } j \neq i \tag{24.15}$$

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

$$L(C_i|C_j) = \begin{cases} 0 & \text{for } i = j \\ 1 & \text{for } i \neq j \end{cases} \tag{24.16}$$

then the result would turn out to be identical to the previous probability-based decision rule, relation (24.5). 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, e.g., 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 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.[2] 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. 24.4). 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.

[2]All error and reject rates are assumed to be calculated as proportions of the total number of test patterns to be classified.

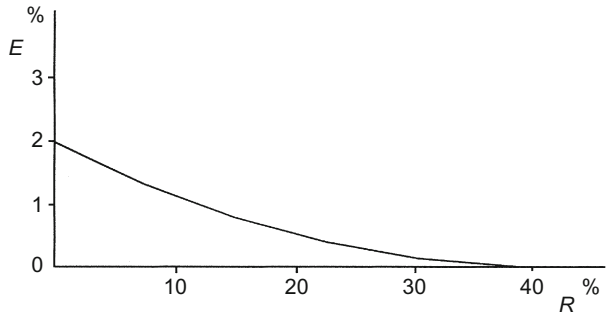

FIGURE 24.4

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%.

24.7 THE RECEIVER OPERATING CHARACTERISTIC

In the early sections of this chapter, there has been an implicit understanding that classification error rates have to be reduced as far as possible, although in the last section it was acknowledged that it is cost rather than error that is the practically important parameter. It was also found that a tradeoff between error rate and reject rate allows a further refinement to be made to the analysis.

Here we consider another refinement that is required in many practical cases where binary decisions have to be made. Radar provides a good illustration of this, showing that there are two basic types of misclassification: first, radar displays may indicate an aircraft or missile when none is present, in which case the error is called a false positive (or in popular parlance, a false alarm); second, they may indicate that no aircraft or missile is present when there actually is one, in which case the error is called a false negative. Similarly, in automated industrial inspection, when searching for deficient products, a false positive corresponds to finding one when none is present, whereas a false negative corresponds to missing a deficient product when one is present.

In fact, there are four relevant categories: (1) true positives (positives that are correctly classified), (2) true negatives (negatives that are correctly classified), (3) false positives (positives that are incorrectly classified), and (4) false negatives (negatives that are incorrectly classified). If many experiments are carried out to determine the proportions of these four categories in a given application, we can obtain the four probabilities of occurrence. Using an obvious notation, these will be related by the following formulae:

$$P_{\mathrm{TP}} + P_{\mathrm{FN}} = 1 \tag{24.17}$$

$$P_{TN} + P_{FP} = 1 \tag{24.18}$$

(In case the reader finds the combinations of probabilities in these formulae confusing, note that an object that is actually a faulty product will be either *correctly* detected as such or *incorrectly* categorized as acceptable, in which case it is a false negative.)

It will be apparent that the probability of error P_E is the sum:

$$P_E = P_{FP} + P_{FN} \tag{24.19}$$

In general, false positives and false negatives will have different costs. Thus, the loss function $L(C_1|C_2)$ will not be the same as the loss function $L(C_2|C_1)$. For example, missing an enemy missile or failing to find a glass splinter in baby food may be far more costly than the cost of a few false alarms (which in the case of food inspection merely means the rejection of a few good products). In fact, there are a good many applications where there is a prime need to reduce as far as possible the number of false negatives (the number of failures to detect the requisite targets).

But how far should we go in aiming to reduce the number of false negatives? This is an important question that should be answered by systematic analysis rather than by *ad hoc* means. The key to achieving this is to note that the proportions of false positives and false negatives will vary independently with the system setup parameters, although frequently only a single threshold parameter need be considered in any detail. In that case we can eliminate this parameter and determine how the numbers of false positives and false negatives depend on each other. The result is the receiver operating characteristic or "ROC" curve (Fig. 24.5).[3]

The ROC curve will often be approximately symmetrical, and, if expressed in terms of probabilities rather than numbers of items, will pass through the points (1, 0), (0, 1)—as shown in Fig. 24.5. It will generally be highly concave so it will pass well below the line $P_{FP} + P_{FN} = 1$, except at its two ends. The point closest to the origin will often be close to the line $P_{FP} = P_{FN}$. This means that if false positives and false negatives are assigned equal costs, the classifier can be optimized simply by minimizing P_E with the constraint $P_{FP} = P_{FN}$. Note, however, that in general the point closest to the origin is *not* the point that minimizes P_E: the point that minimizes total error is actually the point on the ROC curve where the gradient is -1 (Fig. 24.5).

Unfortunately, there is no general theory predicting the shape of the ROC curve. Furthermore, the number of samples in the training set may be limited (especially in inspection if rare contaminants are being sought), and then it may not prove possible to make an accurate assessment of the shape—especially in the extreme wings of the curve. In some cases this problem can be tackled by modeling, e.g., using exponential or other functions—as shown in Fig. 24.6, where the exponential functions lead to reasonably accurate descriptions. However, the

[3]While this text defines the ROC curve in terms of P_{FP} and P_{FN}, many other texts use alternative definitions based, e.g., on P_{TP} and P_{FP}, in which case the graph will appear inverted.

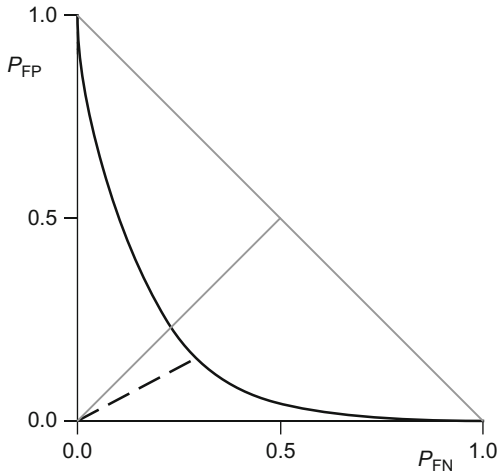

FIGURE 24.5

Idealized ROC curve. The gray line of gradient +1 indicates the position that *a priori* might be expected to lead to minimum error. In fact, the optimum working point is that indicated by the dotted line, where the gradient on the curve is −1. The gray line of gradient −1 indicates the limiting worst case scenario: all practical ROC curves will lie below this line.

underlying shape can hardly be exactly exponential, as this would suggest that the ROC curve tends to zero at infinity rather than at the points (1, 0), (0, 1). Also, there will in principle be a continuity problem at the join of two exponentials. Nevertheless, if the model is reasonably accurate over a good range of thresholds, the relative cost factors for false positives and false negatives can be adjusted appropriately, and an ideal working point determined systematically. Of course, there may be other considerations: e.g., it may not be permissible for the false negative rate to rise above a certain critical level. For examples of the use of the ROC analysis, see Keagy et al. (1995, 1996) and Davies et al. (2003c).

24.7.1 On the Variety of Performance Measures Relating to Error Rates

In signal detection theory (typified by the radar type of application), it is usual to work with error rates rather than the probabilities used in Section 24.7. Hence, we define the following:

$$\text{True positive rate: } tpr = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{24.20}$$

$$\text{True negative rate: } tnr = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{24.21}$$

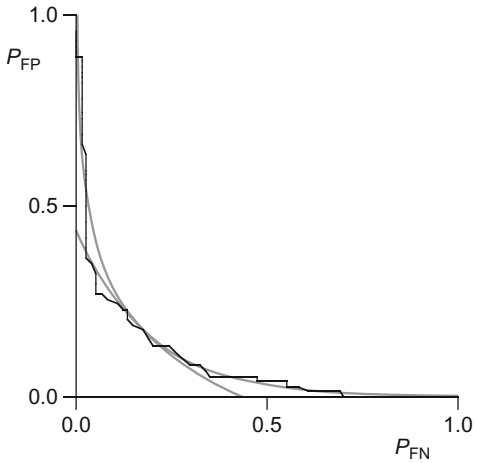

FIGURE 24.6

Fitting a ROC curve using exponential functions. Here the given ROC curve (see Davies et al., 2003c) has distinctive steps resulting from a limited set of data points. A pair of exponential curves fits the ROC curve quite well along the two axes, each having an obvious region that it models best. In this case, the crossover region is reasonably smooth, but there is no real theoretical reason for this. Furthermore, exponential functions will not pass through the limiting points (0, 1) and (1, 0).

$$\text{False positive rate: } fpr = \frac{FP}{N} = \frac{FP}{TN + FP} \tag{24.22}$$

$$\text{False negative rate: } fnr = \frac{FN}{P} = \frac{FN}{TP + FN} \tag{24.23}$$

where P and N are the actual numbers of objects in classes P and N, respectively. Following Eqs. (24.17) and (24.18), we have:

$$tpr + fnr = 1 \tag{24.24}$$

and

$$tnr + fpr = 1 \tag{24.25}$$

These two equations show the consistency of the four definitions presented above.

It is unfortunate and sometimes confusing that a plethora of names for these and related parameters have arisen in various fields of pattern recognition. Below we aim to make sense of these names as well as defining them:

Sensitivity	A parameter describing the success in finding a particular type of target. It is also a synonym for *hit-rate*. It is therefore equal to *tpr*.
Recall	A term used when describing the success in finding an item in a database. It is therefore also equal to *tpr*.

Specificity	A term that is important in medicine, and relates to the proportion of *well* patients who are accurately told after a test that they are not ill. It is therefore equal to $tnr = 1 - fpr$.
Discriminability	A term used when describing the success in differentiating a particular type of target from a similar type of target. It is therefore equal to $TP/(TP + FP)$.
Precision	A term describing the accuracy in picking out a particular type of target from any distractors, including noise and clutter. It is also equal to $TP/(TP + FP)$.
Accuracy	A term used to describe the overall success rate when distinguishing between foreground and background. We can deduce that it must be equal to $(TP + TN)/(P + N)$.
False alarm rate	Another synonym for *fpr*.
Positive predictive value	Another synonym for *precision*.
F-measure	A measure that is used as an overall performance indicator, combining the recall and precision measures (or alternatively the sensitivity and discriminability measures). Because it is the numbers of errors ($FP + FN$) that have to be combined (rather than the error rates themselves), the formula for *F-measure* may initially appear overcomplex: $\frac{2}{1/recall + 1/precision}$.

A more general formula for F-measure is the following:

$$F_\gamma\text{-}measure = \frac{1}{\frac{\gamma}{recall} + \frac{(1-\gamma)}{precision}} \tag{24.26}$$

where γ can be adjusted to give the most appropriate weighting between *recall* and *precision*: it is normally given a value close to 0.5.

Quite often, recall and precision are used together to form ROC-like graphs, although they are distinct from ROC curves which typically show *tpr* vs. *fpr*. (Note that, although $recall = tpr$, $precision \neq fpr$.)

Finally, a valuable performance indicator for binary classifiers is obtained by finding the area under the curve (AUC) for a *tpr* vs. *fpr* ROC curve. It is largest when the ROC curve lies close to the $tpr = 1$, $fpr = 0$ axes. Using this measure, the best classifier is the one that has the largest AUC.

24.8 MULTIPLE CLASSIFIERS

In recent years, there have been moves to make the classification process more reliable by application of multiple classifiers working in cooperation. The basic concept is much like that of three magistrates coming together to make a more reliable judgement than any can make alone. Each is expert in a variety of things, but not in everything, so putting their knowledge together in an appropriate way should permit more reliable judgements to be made. A similar concept applies to expert AI systems: multiple expert systems should be able to make up for each

other's shortcomings. In all these cases, some way should exist for getting the most out of the individual classifiers without confusion reigning.

Note that the idea is not just to take all the feature detectors that the classifiers use and to replace their output decision-making devices with a single more complex decision-making unit. Indeed, such a strategy could well run into the problem discussed in Section 24.5—of exceeding the optimum number of features: at best only a minor improvement would result from such a strategy, and at worst the system would be grossly failure-prone. On the contrary, the idea is to take the final classification of a number of complete but totally separate classifiers and to combine their outputs to obtain a substantially improved output. Furthermore, it could happen that the separate classifiers use totally different strategies to arrive at their decisions: one may be a nearest neighbor classifier; another may be a Bayes' classifier; and another may be a neural network classifier (see Sections 24.13–24.16). Likewise, one may employ structural pattern recognition, one might use SPR, and another might use syntactic pattern recognition. Each will be respectable in its own right: each will have its own strengths and weaknesses. Part of the idea is one of convenience: to make use of any soundly based classifier that is available, and to boost its effectiveness by using it in conjunction with other soundly based classifiers.

The next task is to see how to achieve this in practice. Perhaps the most obvious way forward is to get the individual classifiers to vote for the class of each input pattern. While this is a nice idea, it will often fail because the weaknesses of the individual classifiers may be worse than their strengths. Thus, the concept must be made more sophisticated.

Another strategy is again to allow the individual classifiers to vote, but this time to make them do so in an exclusive manner, so that as many classes as possible are eliminated for each input pattern. This is achievable with a simple intersection rule: a class is accepted as a possibility only if *all* the classifiers indicate that it is a possibility. The strategy is implemented by applying a threshold to each classifier in a special way, which will now be described.

A prerequisite for this strategy to work is that each classifier must not only give a class decision for each input pattern: it must also give the ranks of all possible classes for each pattern. In other words, it must give its first choice of class for any pattern, its second choice for that pattern, and so on. Then the classifier is labeled with the rank it assigned to the true class of that pattern. In fact, we apply each classifier to the whole training set and get a table of ranks (Table 24.1). Finally, we find the worst case (largest rank)[4] for each classifier, and take that as a threshold value that will be used in the final multiple classifier. When using this method for testing input patterns, only those classifiers that are not excluded by

[4]In everyday parlance, the worst case corresponds to the lowest rank, which is here the largest numerical rank; similarly, the highest rank is the smallest numerical rank. It is obviously necessary to be totally unambiguous about this nomenclature.

Table 24.1 Determining a Set of Classifiers for the Intersection Strategy

	Classifier Ranks				
	C_1	C_2	C_3	C_4	C_5
D_1	5	3	7	1	8
D_2	4	9	6	4	2
D_3	5	6	7	1	4
D_4	4	7	5	3	5
D_5	3	5	6	5	4
D_6	6	5	4	3	2
D_7	2	6	1	3	8
thr	6	9	7	5	8

In the upper section of this table, the original classifier ranks are shown for each input pattern; in the bottom line of the table, only the worst-case rank is retained. When later applying test patterns, this can be used as the threshold (marked 'thr') to determine which classifiers should be employed.

the threshold have their outputs intersected to give the final list of classes for the input pattern.

The above "intersection strategy" focuses on the worst-case behavior of the individual classifiers, and the result could be that a number of classifiers will hardly reduce the list of possible classes for the input patterns. This tendency can be tackled by an alternative "union strategy," which focusses on the specialisms of the individual classifiers: the aim is then to find a classifier that recognizes each particular pattern well. To achieve this, we look for the classifier with the smallest rank (classifier rank being defined exactly as already defined above for the intersection strategy) for each individual pattern (Table 24.2). Having found the smallest rank for the individual input patterns, we determine the largest of these ranks that arises for each classifier as we go right through all the input patterns. Applying this value as a threshold now determines whether the output of the classifier should be used to help determine the class of a pattern. Note that the threshold is determined in this way using the training set and is later used to decide which classifiers to apply to individual test patterns. Thus, for any pattern a restricted set of classifiers is identified that can best judge its class.

To clarify the operation of the union strategy, let us examine how well it will work on the training set. In fact, it is guaranteed to retain enough classifiers to ensure that the true class of any pattern is not excluded (although naturally, this is not guaranteed for any member of the test set). Hence, the aim of employing a classifier that recognizes each particular pattern well is definitely achieved.

Unfortunately, this guarantee is not obtained without cost. Specifically, if a member of the training set is actually an outlier, the guarantee will still apply, and the overall performance may be compromised. This problem can be tackled

Table 24.2 Determining a Set of Classifiers for the Union Strategy

	Classifier Ranks					Best Classifiers				
	C_1	C_2	C_3	C_4	C_5	C_1	C_2	C_3	C_4	C_5
D_1	5	3	7	1	8	0	0	0	1	0
D_2	4	9	6	4	2	0	0	0	0	2
D_3	5	6	7	1	4	0	0	0	1	0
D_4	4	7	5	3	5	0	0	0	3	0
D_5	3	5	6	5	4	3	0	0	0	0
D_6	6	5	4	3	2	0	0	0	0	2
D_7	2	6	1	3	8	0	0	1	0	0
			min−max threshold			3	0	1	3	2

In the left-hand section of this table, the original classifier ranks are shown for each input pattern: in the right-hand section of the table, only one rank is retained, namely that obtaining for the classifier that is best able to recognize that pattern. Note that to facilitate the next piece of analysis—finding the thresholds on the classifier ranks—all remaining places in the table are packed with zeros. A zero final threshold then indicates a classifier that is of no help in analyzing the input data.

in many ways, but a simple possibility is to eliminate excessively bad exemplars from the training set. Another way is to abandon the union strategy altogether and go for a more sophisticated voting strategy. Other approaches involve reordering the data to improve the rank of the correct class (Ho et al., 1994).

24.9 CLUSTER ANALYSIS

24.9.1 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 to 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 not unlikely that a number of errors would thereby be circumvented. Unfortunately, unsupervised learning involves a number of difficulties, as will be seen in Subsection 24.9.2.

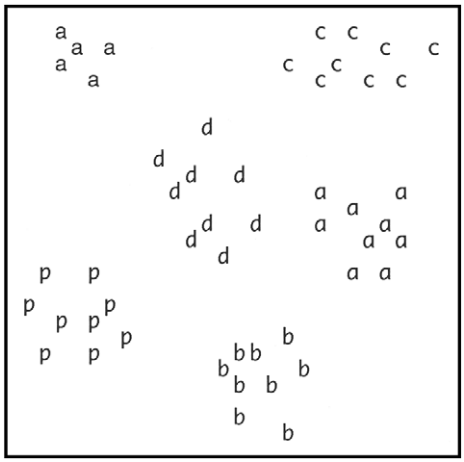

FIGURE 24.7

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 bent 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.

Before proceeding, we give two other reasons why unsupervised learning is useful. First, when the characteristics of objects vary with time—e.g., beans changing in size and color as the season develops—it will be necessary to track these characteristics within the classifier, and unsupervised learning provides an excellent 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, although 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. 24.7). In general, unsupervised learning is about the location of clusters in feature space.

24.9.2 Clustering Procedures

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 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 of 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, note 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 have to be provided to enable them to do so. While 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:

$$^{P}C_{2} = \tfrac{1}{2}P(P-1) \tag{24.27}$$

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,

Table 24.3 Basis of Forgy's Algorithm for Cluster Analysis

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

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:

$$^{k}C_{2} = \tfrac{1}{2}k(k-1) \tag{24.28}$$

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:

$$\sum_{q=1}^{p} {}^{p}C_{q} = \sum_{q=1}^{p} \frac{p!(p-q)!}{q!} \tag{24.29}$$

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, although termination can also be instituted if the quality of clustering ceases to improve. The overall algorithm, which was originally due to Forgy (1965), is given in Table 24.3.

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

Table 24.4 Basis of MacQueen's *k-means* Algorithm

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

Another disadvantage of iterative algorithms is that it may not be obvious when to get them to 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 24.4). 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, the main problem 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 that, 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.

24.10 PRINCIPAL COMPONENTS ANALYSIS

Closely related to cluster analysis is the concept of data representation. One powerful way of approaching this task is that of principal components analysis. This involves finding the mean of a cluster of points in feature space and then finding the principal axes of the cluster in the following way. First an axis is found which passes through the mean position and which gives the maximum variance when

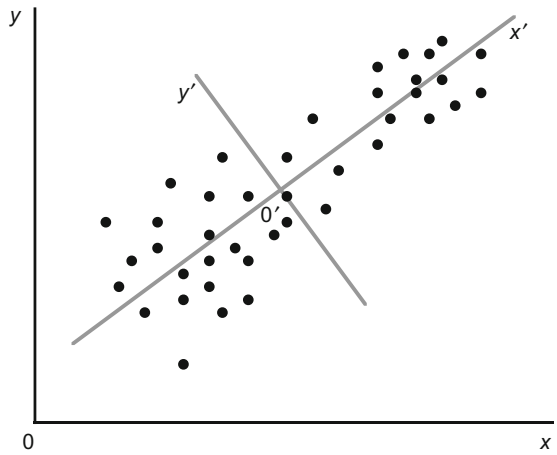

FIGURE 24.8

Illustration of principal components analysis. Here the dots represent patterns in feature space and are initially measured relative to the *x*- and *y*-axes. Then the sample mean is located at 0′, and the direction 0′*x*′ of the first principal component is found as the direction along which the variance is maximized. The direction 0′*y*′ of the second principal component is normal to 0′*x*′; in a higher dimensional space it would be found as the direction normal to 0′*x*′ along which the variance is maximized.

the data is projected onto it. Then a second such axis is found which maximizes variance in a direction normal to the first. This process is carried out until a total of N principal axes have been found for an N-dimensional feature space. The process is illustrated in Fig. 24.8. In fact, the process is entirely mathematical and need not be undertaken in the strict sequence indicated above. It merely involves finding a set of orthogonal axes that diagonalizes the covariance matrix.

The covariance matrix for the input population is defined as:

$$C = \mathrm{E}\{(\mathbf{x}_{(p)} - \mathbf{m})(\mathbf{x}_{(p)} - \mathbf{m})^{\mathrm{T}}\} \tag{24.30}$$

where $\mathbf{x}_{(p)}$ is the location of the pth data point, and $\mathbf{m}$ is the mean of the P data points; $\mathrm{E}\{\ldots\}$ indicates expectation value for the underlying population. We can estimate C from the following equations:

$$C = \frac{1}{P}\sum_{p=1}^{P} \mathbf{x}_{(p)}\mathbf{x}_{(p)}^{\mathrm{T}} - \mathbf{m}\mathbf{m}^{\mathrm{T}} \tag{24.31}$$

$$\mathbf{m} = \frac{1}{P}\sum_{p=1}^{P} \mathbf{x}_{(p)} \tag{24.32}$$

Since C is real and symmetric, it is possible to diagonalize it using a suitable orthogonal transformation matrix A, obtaining a set of N orthonormal eigenvectors $\mathbf{u}_i$ with eigenvalues λ_i given by:

$$C\mathbf{u}_i = \lambda_i \mathbf{u}_i \quad (i = 1, 2, \ldots, N) \tag{24.33}$$

The vectors $\mathbf{u}_i$ are derived from the original vectors $\mathbf{x}_i$ by:

$$\mathbf{u}_i = A(\mathbf{x}_i - \mathbf{m}) \tag{24.34}$$

and the inverse transformation needed to recover the original data vectors is:

$$\mathbf{x}_i = \mathbf{m} + A^{\mathrm{T}}\mathbf{u}_i \tag{24.35}$$

Here we have recalled that, for an orthogonal matrix:

$$A^{-1} = A^{\mathrm{T}} \tag{24.36}$$

In fact it may be shown that A is the matrix whose rows are formed from the eigenvectors of C, and that the diagonalized covariance matrix C' is given by:

$$C' = ACA^{\mathrm{T}} \tag{24.37}$$

so that:

$$C' = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_N \end{bmatrix} \tag{24.38}$$

Note that in an orthogonal transformation, the trace of a matrix remains unchanged. Thus, the trace of the input data is given by:

$$\text{trace } C = \text{trace } C' = \sum_{i=1}^{N} \lambda_i = \sum_{i=1}^{N} s_i^2 \tag{24.39}$$

where we have interpreted the λ_i as the variances of the data in the directions of the principal component axes (note that for a real symmetric matrix, the eigenvalues are all real and positive).

In what follows, we shall assume that the eigenvalues have been placed in an ordered sequence, starting with the largest. In that case, λ_1 represents the most significant characteristic of the set of data points, with the later eigenvalues representing successively less significant characteristics. We could even go so far as to say that, in some sense, λ_1 represents the most interesting characteristic of the data, while λ_N would be largely devoid of "interest." More practically, if we ignored λ_N, we might not lose much useful information, and indeed the last few eigenvalues would frequently represent characteristics that are not statistically significant and are essentially noise. For these reasons, principal components analysis is commonly used for reduction in the dimensionality of the feature space from N to some lower value N'. In some applications, this would be taken as leading to a useful amount

of data compression. In other applications, it would be taken as providing a reduction in the enormous redundancy present in the input data.

We can quantify these results by writing the variance of the data in the reduced dimensionality space as:

$$\text{trace}(C')_{\text{reduced}} = \sum_{i=1}^{N'} \lambda_i = \sum_{i=1}^{N'} s_i^2 \tag{24.40}$$

Not only is it now clear why this leads to reduced variance in the data, but also we can see that the mean square error obtained by making the inverse transformation (Eq. (24.35)) will be:

$$\overline{e^2} = \sum_{i=1}^{N} s_i^2 - \sum_{i=1}^{N'} s_i^2 = \sum_{i=N'+1}^{N} s_i^2 \tag{24.41}$$

One application in which principal components analysis has become especially important is the analysis of multispectral images, e.g., from earth-orbiting satellites. Typically, there will be six separate input channels (e.g., three color and three infra-red), each providing an image of the same ground region. If these images are 512×512 pixels in size, there will be about a quarter of a million data points and these will have to be inserted into a six-dimensional feature space. After finding the mean and covariance matrix for these data points, the latter is diagonalized and a total of six principal component images can be formed. Commonly, only two or three of these will contain immediately useful information, and the rest can be ignored. (For example, the first three of the six principal component images may well possess 95% of the variance of the input images.) Ideally, the first few principal component images in such a case will highlight such areas as fields, roads, and rivers, and this will be precisely the data that is required for map-making or other purposes. In general, the vital pattern recognition tasks can be aided and considerable savings in storage can be achieved on the incoming image data by attending to just the first few principal components.

Finally, it is as well to note that principal components analysis really provides a particular form of data representation. In itself it does not deal with pattern classification, and methods that are required to be useful for the latter type of task must possess useful discrimination. Thus, selection of features simply because they possess the highest variability does not mean that they will necessarily perform well in pattern classifiers. Another important factor that is relevant to the whole study of data analysis in feature space is the scales of the various features. Often, these will be an extremely variegated set, including length, weight, color, numbers of holes, and so on. Clearly, such a set of features will have no special comparability and are unlikely even to be measurable in the same units. This means that placing them in the same feature space and assuming that the scales on the various axes should have the same weighting factors must be invalid. One way of tackling this problem is to normalize the individual features to some

standard scale given by measuring their variances. Such a procedure will naturally radically change the results of principal components calculations and further mitigates against principal components methodology being used thoughtlessly. On the other hand, there are some occasions when different features can be compatible, and where principal components analysis can be performed without such worries: one such situation is where all the features are pixel intensities in the same window (this case is discussed in Section 8.5).

24.11 THE RELEVANCE OF PROBABILITY IN IMAGE ANALYSIS

Having seen the success of Bayes' theory in pointing to apparently absolute answers in the interpretation of certain types of image, it is attractive to consider complex scenes in terms of the probabilities of various interpretations and the likelihood of a particular interpretation being the correct one. Given a sufficiently large number of such scenes and their interpretations, it seems that it ought to be possible to use them to train a suitable classifier. However, practical interpretation in real time is quite another matter. Next, note that the eye−brain system does not appear to operate in a manner corresponding to the algorithms we have studied. Instead, it appears to pay attention to various parts of an image in a nonpredetermined sequence of "fixations" of the eye, interrogating various parts of the scene in turn and using the newly acquired information to work out where the next piece of relevant information is to come from. Clearly, it is employing a process of *sequential pattern recognition*, which saves effort overall by progressively building up a store of knowledge about relevant parts of the scene and at the same time forming and testing hypotheses about its structure.

The above process can be considered as one of modifying and updating the *a priori* probabilities as analysis progresses. This is an inherently powerful process, since the eye is thereby not tied to "average" *a priori* probabilities for *all* scenes but is able to use information in a particular scene to improve on the average *a priori* probabilities. However, it will be difficult to estimate at all accurately the *a priori* probabilities for sequences of real, complex scenes. So while this is a tempting approach, its realization will be fraught with difficulty.

Nevertheless, the concept of probability is useful when it can validly be applied. This certainly covers cases where a restricted range of images can arise, such that the consequent image description contains relatively few bits of information—*viz.* those forming the various pattern class names. In addition, structured images containing several parts that can separately be recognized and then coupled together can also be dealt with under the SPR (probabilistic) formalism. Overall, there are limits to its application, but it can still be used in conjunction with structural, syntactic, and other forms of pattern recognition in the design of more powerful recognition systems.

24.12 ANOTHER LOOK AT STATISTICAL PATTERN RECOGNITION: THE SUPPORT VECTOR MACHINE

The support vector machine (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. 24.9(a). The idea is to find the pair of parallel hyperplanes that leads to the maximum separation between two classes of feature so as to provide the greatest protection against errors. In Fig. 24.9(a), 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. 24.9 (a), the planes are fully defined by three support vectors, although 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; since however many data points exist in a feature space, the maximum number of vectors used to describe it is $N+1$.

For comparison, Fig. 24.9(b) shows the situation that would exist if the nearest neighbor method 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.

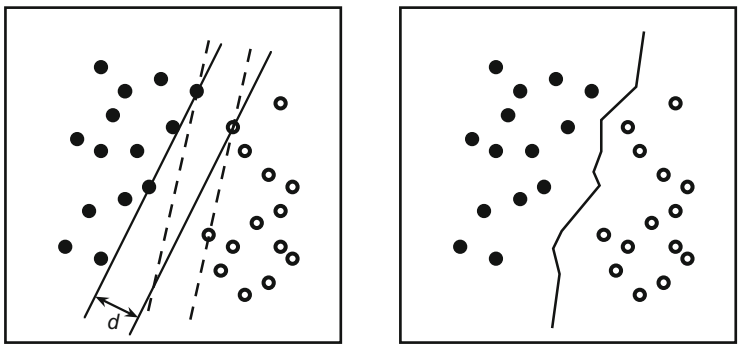

FIGURE 24.9

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.

To overcome this problem, it is possible to transform the training and test data to a feature space of higher dimension where the data does 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 plus 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 basic 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 the original papers by Vapnik, including Vapnik (1998), the specialized text by Cristianini and Shawe-Taylor (2000), or other texts on SPR, such as Webb (2002).

24.13 ARTIFICIAL NEURAL NETWORKS

The concept of an artificial neural network that could be useful for pattern recognition 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 continuous following for many years, it is probably no exaggeration to say that it is Rosenblatt's "perceptron" (1958, 1962), which has had the greatest 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. 24.10). The mathematics is simplified by writing $-\theta$ as w_0, and taking it to correspond to an input x_0 that is maintained at a constant value of unity. The output of the linear part of the classifier is then written in the form:

$$d = \sum_{i=1}^{N} w_i x_i - \theta = \sum_{i=1}^{N} w_i x_i + w_0 = \sum_{i=0}^{N} w_i x_i \tag{24.42}$$

and the final output of the classifier is given by:

$$y = f(d) = f\left(\sum_{i=0}^{N} w_i x_i\right) \tag{24.43}$$

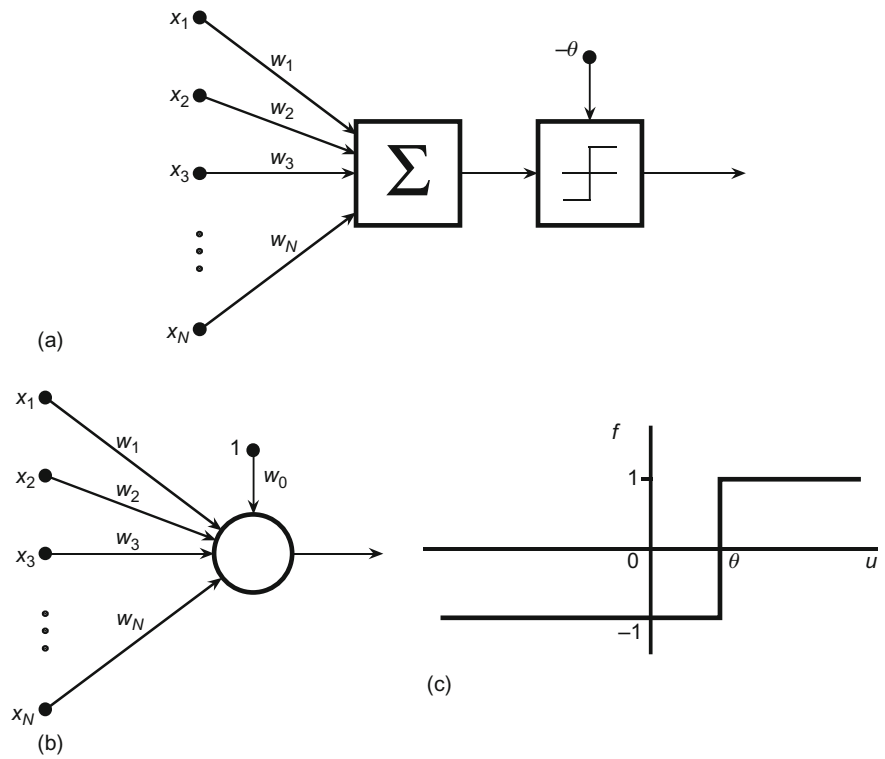

FIGURE 24.10

Simple perceptron. (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. (b) gives a convenient shorthand notation for the perceptron; and (c) shows the activation function of the threshold unit.

This type of neuron can be trained using a variety of procedures, such as the *fixed increment rule* given in Table 24.5. (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:

$$w_i(k+1) = w_i(k) \quad y(k) = \omega(k) \tag{24.44}$$

$$w_i(k+1) = w_i(k) + \eta[\omega(k) - y(k)]x_i(k) \quad y(k) \neq \omega(k) \tag{24.45}$$

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

Table 24.5 Perceptron *Fixed Increment* Algorithm

```
initialize weights with small random numbers;
select suitable value of learning rate coefficient η in the range 0 – 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;
```

times, *and* if the feature vectors are linearly separable, then the algorithm will converge to a correct error-free solution.

Unfortunately, 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, which 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:

$$w_i(k+1) = w_i(k) + \eta\delta x_i(k) = w_i(k) + \eta[\omega(k) - d(k)]x_i(k) \tag{24.46}$$

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. Figure 24.11 clarifies the situation by appealing to a 2-D case. Figure 24.11(a) shows separable data, which is 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. 24.11(b) 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. 24.11(a) 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 that minimizes the probability of error. Clearly, the Widrow–Hoff rule goes some way to solving this problem.

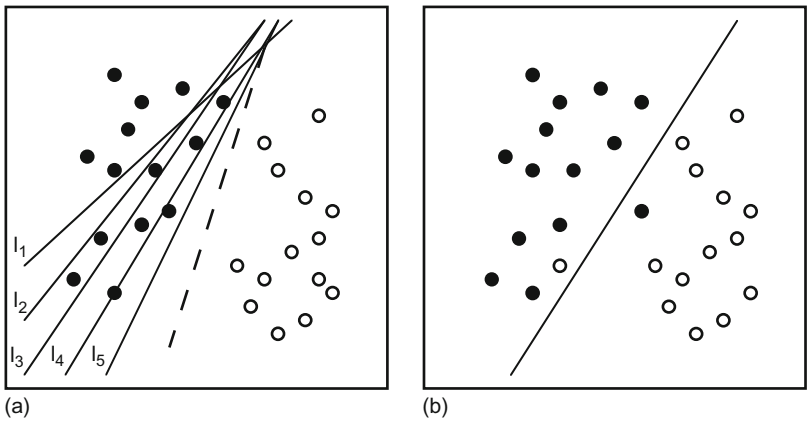

FIGURE 24.11

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, although it is only capable of dichotomizing feature data, a suitably trained array of simple perceptrons—the "single-layer perceptron" of Fig. 24.12—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 approach by other means. In particular, multilayer perceptron (MLP) networks (see Fig. 24.13)—which would emulate the neural networks in the brain—seem poised to provide a solution since they should be able to recode the outputs of the first layer of simple perceptrons.

Rosenblatt himself proposed such networks, but was unable to propose 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.

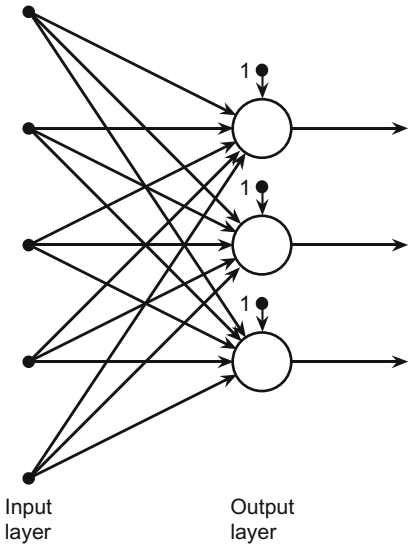

FIGURE 24.12

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.

24.14 THE BACK-PROPAGATION ALGORITHM

The problem of training an MLP can be simply stated: a general layer of an 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). In addition, it is not evident what the optimal MLP architecture should be. While 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."[5]

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

[5]This is 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.

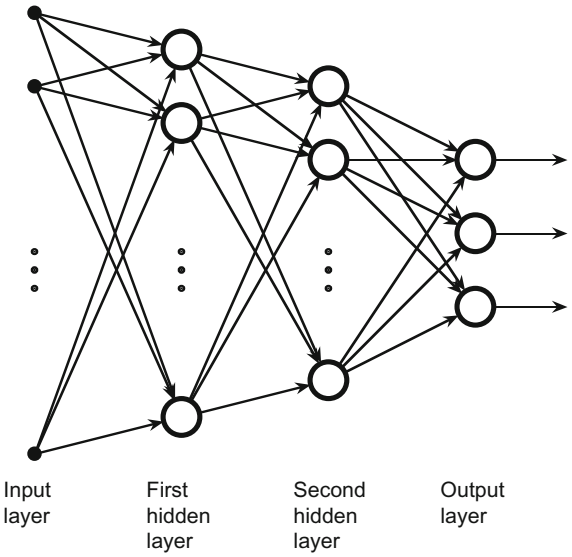

FIGURE 24.13

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 perform 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.

consider removing the thresholding functions from the lower layers of MLP networks to make them easier to train. Unfortunately, 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 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. 24.14), is effective, and indeed is almost certainly the most widely used of the available functions.[6] Once these softer activation functions were used, it became possible for each layer of the MLP to

[6]We do not here make a marked distinction between symmetrical activation functions and alternatives that are related to them by shifts of axes, although 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, although it ranges from 0 and 1 as v goes from $-\infty$ to ∞.

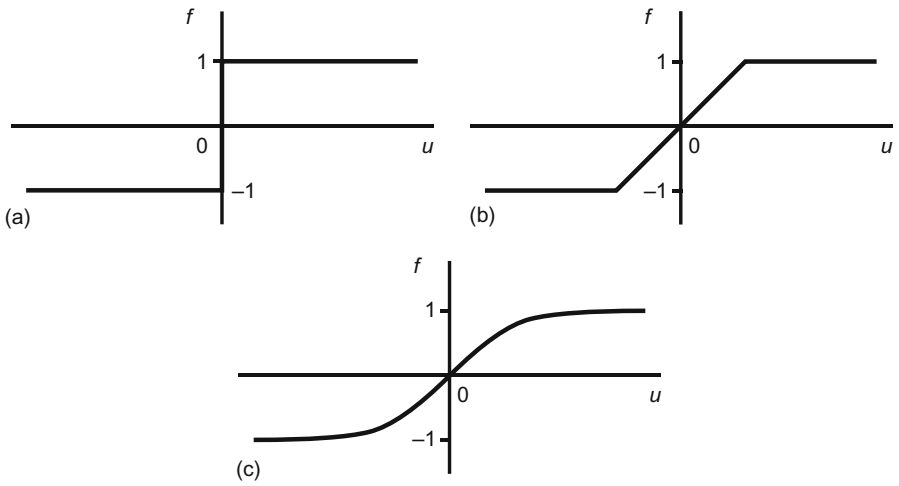

FIGURE 24.14

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

"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 which 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 24.6). 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 $\delta_j = y_j(1 - y_j)(\Sigma_m \delta_m w_{jm})$, 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 downward one layer at a time.
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.

Table 24.6 The Back-Propagation Algorithm

```
initialize weights with small random numbers;
select suitable value of learning rate coefficient η in the range 0 – 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)
                δ_j = y_j(1 − y_j)(t_j − y_j);
            else δ_j = y_j(1 − y_j)(Σ_m δ_m w_jm);
            adjust weights i of node j according to w_ij = w_ij + ηδ_j y_i;
        }
} until changes are reduced to some predetermined level;
```

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.
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 is almost universally 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 $k-1$:

$$w_{ij}(k+1) = w_{ij}(k) + \eta\delta_j y_i + \alpha[w_{ij}(k) - w_{ij}(k-1)] \tag{24.47}$$

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

24.15 MLP ARCHITECTURES

The preceding sections gave the motivation for designing an 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

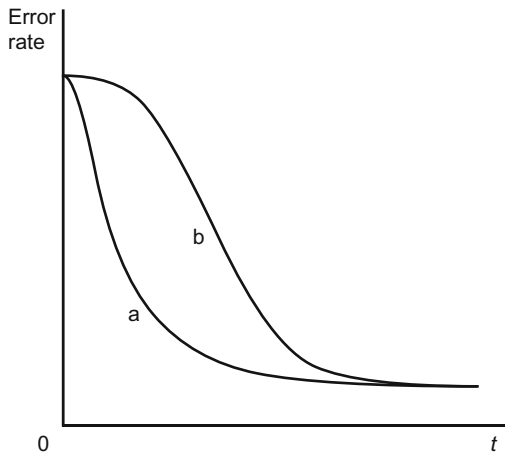

FIGURE 24.15

Learning curve for the multilayer perceptron. Here curve (a) shows the learning curve for a single-layer perceptron, and curve (b) shows that for a multilayer perceptron. Note that the multilayer perceptron takes considerable time to get going, since 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, although this situation would seldom occur in practice.

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

Although the back-propagation algorithm can train MLPs of 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. 24.15). 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 are especially welcome.

24.16 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,

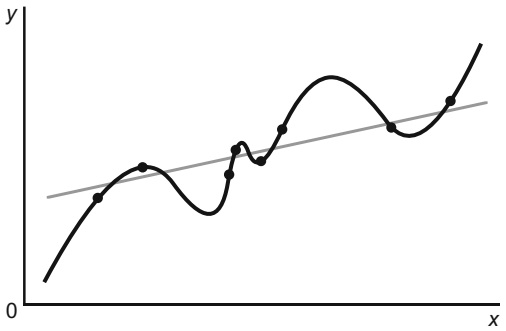

FIGURE 24.16

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

although clearly not to the extent of being able to respond correctly to types of data different from that 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 has 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. Figure 24.16 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 most 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.[7] This is not difficult, since we merely need to test the system periodically during training to ensure that the point of

[7]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 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.

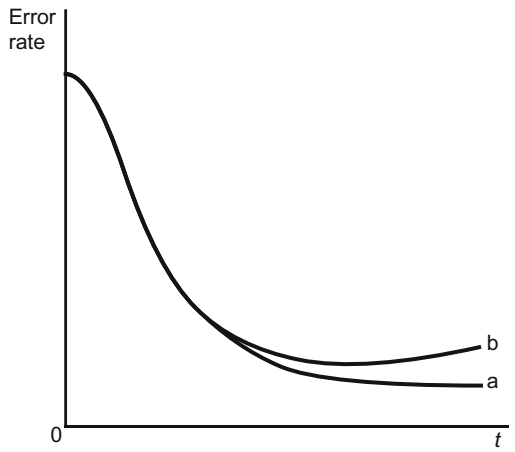

FIGURE 24.17

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. Curve (a) tends to go on improving even when overfitting is occurring. However, this situation is detected when curve (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 curve (b).

overadaptation has not been reached. Figure 24.17 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 while performance on the training data appears to go on improving. This is the point where serious overfitting is occurring, and the training process should be curtailed. The aim, then, is to make the whole 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 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 an 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.

24.17 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 several factors: (a) expediency and in particular the need for speed of interpretation; (b) the fact that algorithms are designed 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; and (c) tacit recognition of the situation outlined in Section 24.11, that probabilistic methods have limited applicability. In practice, it is at the stage of strong image structure and contextual analysis that probabilistic interpretations really come into their own.

Nonetheless, SPR is extremely valuable within its own range of utility. This includes identifying objects on conveyors and making value judgements 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 and insignificant perturbation: much the same comments apply to the subject of principal components analysis which has had an increasing following in many areas of machine vision (see Section 24.10); nor should it go unnoticed that these topics link in strongly with ANNs, which are often able to play a powerful role.

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 nearest neighbor algorithm, ROCs, PCA, 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 possibility of overfitting.

24.18 BIBLIOGRAPHICAL AND HISTORICAL NOTES

Although the subject of SPR tends not to be at the center of attention in image analysis work,[8] 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. Most of the relevant work on this topic was 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, e.g., 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 pattern recognition, 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, 1988f); see Section 24.4.

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

[8]Note, however, that it is vital to the analysis of multispectral data from satellite imagery.

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 cited therein.

While the field of pattern recognition 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). The reader can also appeal to the review article by Jain et al. (2000), which outlines new areas that appeared in the previous decade.

The multiple classifier approach is a relatively recent development, and is well reviewed by Duin (2002). Ho et al. (1994) dates from when the topic was rather younger, and lists an interesting set of options as seen at that point—some of these being covered in Section 24.8.

"Bagging" and "boosting" are further variants on the multiple classifier theme: they were developed by Breiman (1996) and Freund and Schapire (1996). Bagging (short for "bootstrap aggregating") means sampling the training set, with replacement, n times, generating b bootstrap sets to train b subclassifiers, and assigning any test pattern to the class most often predicted by the subclassifiers. The method is particularly useful for unstable situations (such as when classification trees are used), but is almost valueless when stable classification algorithms are used (such as the nearest neighbor algorithm). Boosting is useful for aiding the performance of weak classifiers. In contrast with bagging, which is a parallel procedure, boosting is a sequential deterministic procedure. It operates by assigning different weights to different training set patterns according to their intrinsic (estimated) accuracy. For further progress with these techniques, see Rätsch et al. (2002), Fischer and Buhmann (2003), and Lockton and Fitzgibbon (2002). Finally, Beiden et al. (2003) discuss a variety of factors involved in the training and testing of competing classifiers; in addition, much of the discussion relates to multivariate ROC analysis.

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 digress to outline something of 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 back-propagation 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 back-propagation 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), among others (Fasel, 2002; Garcia and Delakis, 2002). For further general information on ANNs, see the book by Bishop (2006).

24.18.1 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, "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 semi-supervised 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.

Li and Zhang (2004) describe how a new boosting algorithm "FloatBoost" has been applied to produce the first real-time multiview face detection system reported. The method uses a backtrack mechanism after each iteration of AdaBoost learning to minimize the error rate directly; it also uses a novel statistical model for learning the best weak classifiers and a stagewise approximation to the posterior probability, thereby requiring fewer weak classifiers than AdaBoost. Gao et al. (2010) report on a modified version of AdaBoost to resolve the key problems of how to *select* the most discriminative weak learners and how to optimally *combine* the selected weak learners. Experiments confirm the utility of the algorithm including the capability to solve these two key problems; both synthetic and real scene data (car and non-car patterns) are used for the tests. Fumera et al. (2008) present a theoretical analysis of bagging as a linear combination of classifiers, thereby giving an analytical model of bagging misclassification probability as a function of ensemble size.

Youn and Jeong (2009) describe a class-dependent feature scaling method employing a naive Bayes' classifier for text data mining, including functions such as text categorization and search. While the reasons why the naive 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 naive Bayes' classifier maps well to text categorization. Rish (2001) provides an empirical study of naive Bayes, containing much useful information.

Decision trees provide a convenient fast-operating method of pattern recognition, and the methodology has developed quite rapidly in recent years. Chandra et al. (2010) describe a new node splitting procedure called the distinct class based splitting measure (DCSM) for decision tree construction. Node splitting measures are important as they help to produce compact decision trees with improved generalization abilities. Chandra et al. have shown that DCSM is well-behaved and produces decision trees that are more compact and provide better classification accuracy than trees constructed using other common node splitting measures. The DCSM measure also helps with pruning (which produces compact trees with better classification accuracy). Köktas et al. (2010) describe a multi-classifier for grading knee osteoarthritis using gait analysis. It employs a decision tree with MLPs at the leaves. In fact, three different MLPs (different "experts") with binary classifications are employed at different leaves of the tree. They showed that, for this type of data, this produced better results than a single multi-class classifier. Rodríguez et al. (2010) describe tests made on a large number of datasets using ensemble methods to generate more accurate classifiers. They show that, for multiclass problems, ensembles of decision trees ("forests") can be successfully combined with ensembles of nested dichotomies. The direct approach, using ensembles of nested dichotomies with a forest method as the base classifier, can be improved using ensemble methods with a nested dichotomy of decision trees as the base classifier.

Fawcett (2006) produced an excellent, largely tutorial summary of ROC analysis in which many descriptors employing true and false positives and negatives are used; a valuable feature of the paper is the unification of a subject in which many apparently different descriptors appear with different names according to the varying backgrounds of the workers. In particular, the recently much more widely used terms "precision" and "recall" are related to "sensitivity," "specificity," "accuracy," and others (for definitions and further discussion of these performance measures, see Section 24.7.1). In addition, measures such as "F-measure" are defined, and problems and pitfalls of using ROC graphs are pointed out. Ooms et al. (2010) underline the value of Fawcett's summary, but show that the ROC concept is limited and is not an optimal measure for *sorting* as distinct from cases where *misclassification costs* are the main concern. They propose a sorting optimization curve (SOC) to cope with sorting problems and help identify the

best choice of operating point in that case. In contrast with the ROC curve, which plots *fpr* vs. *fnr* or *tpr* vs. *fpr*, the SOC curve plots yield rate (Y) vs. relative quality improvement rate (Q), where $Y = (TP + FP)/(P + N)$; this formula arises because no distinction is made between true and false positives when selling a product. Quality Q is defined in terms of the precision $Pr = TP/(TP + FP)$, *viz.* $Q = f(Pr)$, and uses whatever function f is needed to achieve this when sorting a particular commodity (such as apples). Typically, optimization involves moving up the Y vs. Q curve until reaching the lowest level of quality that is acceptable or legal.

Assessing the quality of the ROC curve has acquired some importance in the past decade, and the AUC (area under the curve) measure has been the main performance indicator for this (Fawcett, 2006). For example, Hu et al. (2008) have used it to advantage for optimal evaluation and selection of features.

24.19 PROBLEMS

1. Show that if the cost function of Eq. (24.16) is chosen, the decision rule (24.15) can be expressed in the form of relation (24.5).
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. Why is the point on a ROC curve closest to the origin *not* the point that minimizes total error? Prove that the point that minimizes the total error on a ROC curve is actually the point where the gradient is -1 (see Section 24.7).
4. Consider the four quantities TP, TN, FP, FN defined in Section 24.7. Arrange them in order of size for situations where positives are rare and recognition errors are likely to be low. If the rates *tpr*, *tnr*, *fpr*, *fnr* are also arranged in order of size, will the order be the same as for TP, TN, FP, FN?
5. Compare the shapes of ROC curves and precision−recall curves for identical classifiers. What mathematical relations link their shapes? Determine whether the ROC curves for two classifiers will cross each other the same number of times as the precision−recall curves.
6. Prove that Eq. (24.26) provides a mathematically sound way of combining precision and recall into a single measure.