11
Classification

11.1 The purpose of automated classification

The identification of cells in a histological slide as healthy or abnormal, the separation of 'high-quality' fruit from inferior specimens in a fruit-packing plant and the categorization of remote-sensing images are just a few simple examples of *classification* tasks. The first two examples are fairly typical of *binary classification*. In these cases, the purpose of the classification is to assign a given cell or piece of fruit to either of just two possible classes. In the first example above, the two classes might be 'healthy' and 'abnormal'; in the second example, these might be 'top quality' (expensive) or 'market grade' (cheap). The third example typically permits a larger number of classes to be assigned ('forest', 'urban', 'cultivated', 'unknown', etc.).

Autonomous classification is a broad and widely studied field that more properly belongs within the discipline of *pattern recognition* than in image processing. However, these fields are closely related and, indeed, classification is so important in the broader context and applications of image processing that some basic discussion of classification seems essential. In this chapter we will limit ourselves to some essential ideas and a discussion of some of the best-known techniques.

Classification takes place in virtually all aspects of life and its basic role as a necessary first step before selection or other basic forms of decision-making hardly needs elaboration. In the context of image processing, the goal of classification is to identify characteristic features, patterns or structures within an image and use these to assign them (or indeed the image itself) to a particular class. As far as images and visual stimuli go, human observers will often perform certain classification tasks very accurately. Why then should we attempt to build automatic classification systems? The answer is that the specific demands or nature of a classification task and the sheer volume of data that need to be processed often make automated classification the only realistic and cost-effective approach to addressing a problem. Nonetheless, expert human input into the design and training of automated classifiers is invariably essential in two key areas:

(1) *Task specification* What exactly do we want the classifier to achieve? The designer of a classification system will need to decide what *classes are going to be considered* and *what*

Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab Chris Solomon and Toby Breckon
 ISBN: 978-0-470-84472-4

variables or parameters are going to be important in achieving the classification.[1] For example, a simple classifier designed to make a preliminary medical diagnosis based on image analysis of histological slides may only aim to classify cells as 'abnormal' or 'normal'. If the classifier produces an 'abnormal' result, a medical expert is typically called upon to investigate further. On the other hand, it may be that there is sufficient information in the shape, density, size and colour of cells in typical slides to attempt a more ambitious classification system. Such a system might assign the patient to one of a number of categories, such as 'normal', 'anaemic', 'type A viral infection' and so on.

(2) *Class labelling* The process of training an automated classifier can often require 'manual labelling' in the initial stage, a process in which an expert human user assigns examples to specific classes based on selected and salient properties. This forms part of the process in generating so-called *supervised* classifiers.

11.2 Supervised and unsupervised classification

Classification techniques can be grouped into two main types: *supervised* and *unsupervised*. Supervised classification relies on having example pattern or feature vectors which have already been assigned to a defined class. Using a sample of such feature vectors as our training data, we design a classification system with the intention and hope that *new examples* of feature vectors which were not used in the design will subsequently be classified accurately. In supervised classification then, the aim is to use training examples to design a classifier which *generalizes well* to new examples. By contrast, unsupervised classification does not rely on possession of existing examples from a known pattern class. The examples are not labelled and we seek to identify groups directly within the overall body of data and features which enables us to distinguish one group from another. Clustering techniques are an example of unsupervised classification which we will briefly discuss later in the chapter.

11.3 Classification: a simple example

Consider the following illustrative problem. Images are taken at a food processing plant in which three types of object occur: pine-nuts, lentils and pumpkin seeds. We wish to design a classifier that will enable us to identify the three types of object accurately. A typical image frame is shown in Figure 11.1.

Let us assume that we can process these images to obtain reliable measurements on the two quantities of circularity and line-fit error.[2] This step (often the most difficult) is called *feature extraction*. For each training example of a pine-nut, lentil or pumpkin seed we thus obtain a 2-D *feature vector* which we can plot as a point in a 2-D *feature space*. In Figure 11.2,

[1] In a small number of specialist applications, it is possible even to attempt to identify identify appropriate patterns and classes automatically; but generally, the expert input of the designer at this stage is absolutely vital.
[2] The details of these measures and how they are obtained from the image do not concern us here.

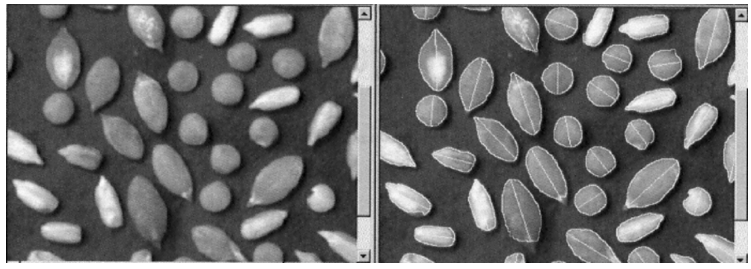


Figure 11.1 Three types of object exist in this image: pine-nuts, lentils and pumpkin seeds. The image on the right has been processed to extract two: circularity and line-fit error

training examples of pine-nuts, lentils and pumpkin seeds are respectively identified by squares, triangles and circles.

We can see by direct inspection of Figure 11.2 that the three classes form more or less distinct clusters in the feature space. This indicates that these two chosen features are broadly adequate to discriminate between the different classes (i.e. to achieve satisfactory classification). By contrast, note from Figure 11.2 what happens if we consider the use of just one of these features in isolation. In this case, the feature space reduces to a single dimension given by the orthogonal projection of the points either onto the vertical (line-fit error) or horizontal (circularity) axis. In either case, there is considerable overlap or confusion of classes, indicating that misclassification occurs in a certain fraction of cases.

Now, the real aim of any automatic classification system is to *generalize* to new examples, performing accurate classification on objects or structures whose class is *a priori* unknown.

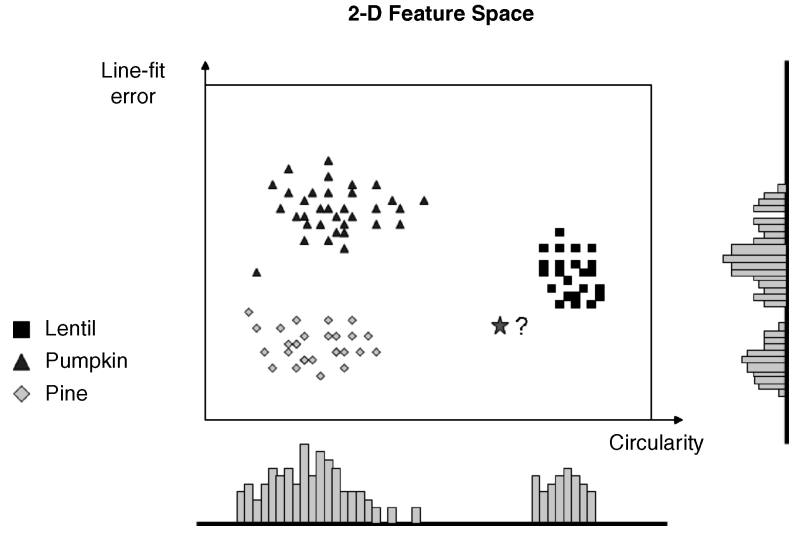


Figure 11.2 A simple 2-D feature space for discriminating between certain objects. The 1-D bar plots in the horizontal x and vertical y directions show the projection of the data onto the respective axes

Consider, then, a previously unseen object having the feature vector indicated in Figure 11.2 by a star. How should we assign this object to its appropriate class? As this object has features which are closer to the lentils, common sense would suggest that it is more likely to be a lentil, but where should we draw the boundary between classes and how sure can we be about our decision? Most of the classification methods we will explore in the remainder of this chapter are concerned with just such questions and finding rigorous mathematical criteria to answer them. First, however, we move from our simple example to a more general overview of the steps in the design of classifiers.

11.4 Design of classification systems

Figure 11.3 provides a summary flow chart for classifier design. We elaborate below on the main steps indicated in Figure 11.3.

(1) *Class definition* Clearly, the definition of the classes is *problem specific*. For example, an automated image-processing system that analysed mammograms might ultimately aim to classify the images into just two categories of interest: normal and abnormal. This would be a *binary classifier* or, as it is sometimes called, a dichotomizer. On the other hand, a more ambitious system might attempt a more detailed diagnosis, classifying the scans into several different classes according to the preliminary diagnosis and the degree of confidence held.

(2) *Data exploration* In this step, a designer will explore the data to identify possible attributes which will allow discrimination between the classes. There is no fixed or best way to approach this step, but it will generally rely on a degree of intuition and common sense. The relevant attributes can relate to absolutely any property of an image or image region that might be helpful in discriminating one class from another.

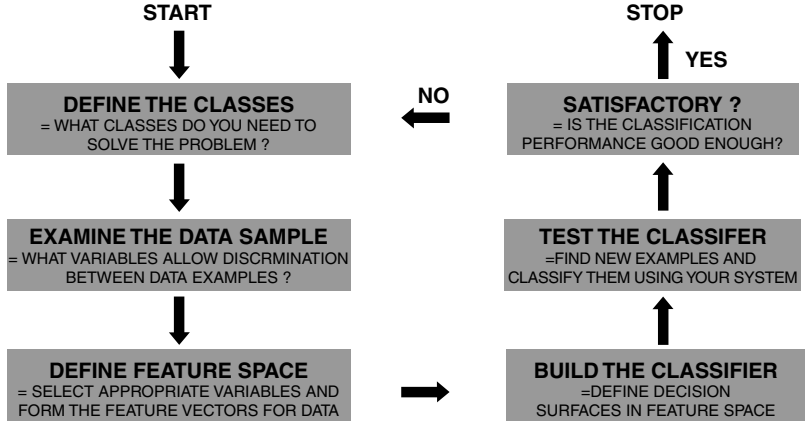


Figure 11.3 Flow diagram representing the main steps in classifier design

Most common measures or attributes will be broadly based on *intensity*, *colour*, *shape*, *texture* or some mixture thereof.

(3) *Feature selection and extraction* Selection of the discriminating features defines the feature space. This, of course, implicitly assumes that we have established reliable image-processing procedures to perform the necessary feature extraction.[3] In general, it is crucial to select features that possess two key properties. The first is that the feature set be as *compact* as possible and the second that they should possess what, for want of a more exact word, we will call *discriminatory power*. A compact set of features is basically a small set and it is of paramount importance, as larger numbers of selected features require an increasingly large number of training samples to train the classifier effectively. Second, it obviously makes sense to select (i) attributes whose distribution over the defined classes is as widely separated as possible and (ii) that the selected set of attributes should be, as closely as possible, statistically independent of each other. A set of attributes possessing these latter two properties would have maximum discriminatory power.

A simple but concrete illustration of this idea might be the problem of trying to discriminate between images of a lion and a leopard. Defining feature vectors which give some measure of the boundary (e.g. a truncated radial Fourier series) or some gross shape measure like form factor or elongation[4] constitute one possible approach. However, it is clearly not a very good one, since lions and leopards are both big cats of similar physique. A better measure in this case will be one based on *colour*, since the distribution in r–g chromatic space is simple to measure and is quite sufficient to discriminate between them.

(4) *Build the classifier using training data* The training stage first requires us to find a sample of examples which we can reliably assign to each of our selected classes. Let us assume that we have selected N features which we anticipate will be sufficient to achieve the task of discrimination and, hence, classification. For each training example we thus record the measurements on the N features selected as the elements of an N-dimensional feature vector $\mathbf{x} = [x_1, x_2, \cdots, x_N]$. All the training examples thus provide feature vectors which may be considered to occupy a specific location in an (abstract) N-dimensional feature space (the jth element x_j specifying the length along the jth axis of the space). If the feature selection has been performed judiciously, then the feature vectors for each class of interest will form more or less distinct clusters or groups of points in the feature space.

(5) *Test the classifier* The classifier performance should be assessed on a *new sample* of feature vectors to see how well it can generalize to new examples. If the performance is unsatisfactory (what constitutes an unsatisfactory performance is obviously application dependent), the designer will return to the drawing board, considering whether

[3] Something that is often very challenging but which, in the interest of staying focused, we will gloss over in this chapter.

[4] The formal definitions of these particular measures are given in Chapter 9.

Table 11.1 Some common pattern classification terms with summary descriptions

Term	Description
Feature vector (pattern vector)	An N-dimensional vector, each element of which specifies some measurement on the object
Feature space	The abstract (N-dimensional) mathematical space spanned by the feature vectors used in the classification problem
Training data	A collection of feature vectors used to build a classifier
Test data	A collection of feature vectors used to test the performance of a classifier
Pattern class	A generic term which encapsulates a group of pattern or feature vectors which share a common statistical or conceptual origin
Discriminant function	A function whose value will determine assignment to one class or another; typically, a positive evaluation assigns to one class and negative to another

the selected classes are adequate, whether the feature selection needs to be reconsidered or whether the classification algorithm itself needs to be revised. The entire procedure would then be repeated until a satisfactory performance was achieved.

Table 11.1 summarizes some of the main terms used in the field of pattern classification.

11.5 Simple classifiers: prototypes and minimum distance criteria

A prototype is an instance of something serving as a typical example or standard for other things of the same category. One of the simplest and most common methods of defining a prototype for a particular class is to take a number of training examples for that class and calculate their average. Figure 11.4 shows a set of 8 male faces and 8 female faces and their calculated prototypes. In this particular example, the original images contain many tens of thousands of pixels. By a procedure based on PCA, however, it is possible to represent these faces as lower dimensional, feature vectors $\{\mathbf{x}_i\}$ whose components provide a compact description of both the shape and texture of the face and from which the original images can be reconstructed with high accuracy. The prototypes $\mathbf{M}$ and $\mathbf{F}$ of the two classes are simply calculated as

$$\mathbf{M} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i^{\mathrm{M}} \qquad \text{and} \qquad \mathbf{F} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i^{\mathrm{F}} \tag{11.1}$$

where $\{\mathbf{x}_i^{\mathrm{M}}\}$ correspond to the male examples and $\{\mathbf{x}_i^{\mathrm{F}}\}$ the female examples and there are N examples of each. The reconstructed prototypes shown in Figure 11.4 are given by applying Equation (11.1) and then reconstructing the original images from the feature vectors.

Minimum distance classifiers are conceptually simple and generally easy to calculate. The first step is to calculate the mean feature vector or *class prototype* for each class. When

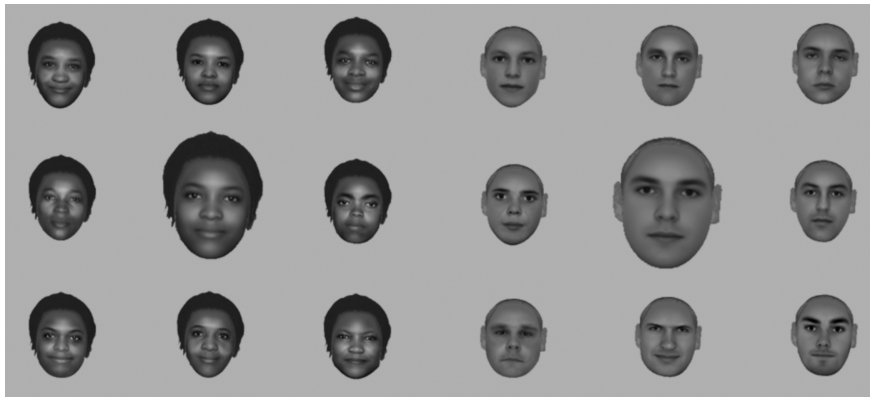


Figure 11.4 Facial prototypes: Left: 8 female faces with the prototype at the centre. Right: 8 male faces with the calculated prototype at the centre. The prototypes are calculated by averaging the shape coordinates of each example and then averaging the RGB colour values of each face after warping to the average shape

presented with a previously unseen feature vector, the *minimum distance classifier will simply assign the new example to that class whose prototype lies closest to it*. We note immediately that the concept of 'distance' in a feature space is a flexible one and admits of a number of other valid definitions apart from the familiar Euclidean measure. However, the principle of a minimum distance classifier is the same irrespective of the precise metric employed. Table 11.2 gives several such metrics which are sometimes used.

11.6 Linear discriminant functions

To apply a minimum distance classifier by calculating the Euclidean distance in the feature space from a given point to the prototype of each class involves a quadratic function of distance and is a straightforward enough procedure. However, it is

Table 11.2 Some common distance measures or *metrics* between two *N*-dimensional vectors **x** and **y**

Metric	Definition
Euclidean distance (L_2 norm)	$L(\mathbf{x},\mathbf{y}) = \left[\sum_{i=1}^{N}(x_i - y_i)^2\right]^{1/2}$
City-block* (L_1 norm)	$L(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N}\lvert x_i - y_i\rvert$
Minkowski metric (L_k norm)	$L_k(\mathbf{x},\mathbf{y}) = \left[\sum_{i=1}^{N}\lvert x_i - y_i\rvert^k\right]^{1/k}$
Mahalanobis distance	$L(\mathbf{x},\mathbf{y}) = [(\mathbf{x}-\mathbf{y})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathbf{y})]^{1/2}$, where $\boldsymbol{\Sigma} = \langle(\mathbf{x}-\vec{\mu}_x)(\mathbf{y}-\vec{\mu}_y)^{\mathrm{T}}\rangle$

*Also known as the Manhattan Distance.

possible to show that application of the minimum distance criterion reduces to a formulation that is of greater utility when we extend our discussion to more complex criteria. It is thus useful to introduce at this point the concept of a (*linear*) *discriminant function*.

For simplicity, consider constructing a line in a 2-D feature space which orthogonally bisects the line connecting a given pair of class prototypes (let us say from the ith prototype possessing coordinates (x_1^i, x_2^i) to the jth prototype (x_1^j, x_2^j)) and passes through the exact midpoint between them. This line divides the feature space into two regions, one of which is closer to the ith prototype, the other to the jth prototype and any point lying on the line is equidistant from the two prototypes. This line is the simplest example of a linear discriminant function. It is easy to show that this function can be expressed in the general form

$$g_{ij}(\mathbf{x}) = g_{ij}(x_1, x_2) = ax_1 + bx_2 + c = 0 \tag{11.2}$$

If we evaluate $g_{ij}(\mathbf{x})$ at an arbitrary position within the feature space $\mathbf{x} = (x_1, x_2)$, then a positive value indicates that the point lies closer to the jth prototype, whereas a negative value indicates greater proximity to the ith prototype.

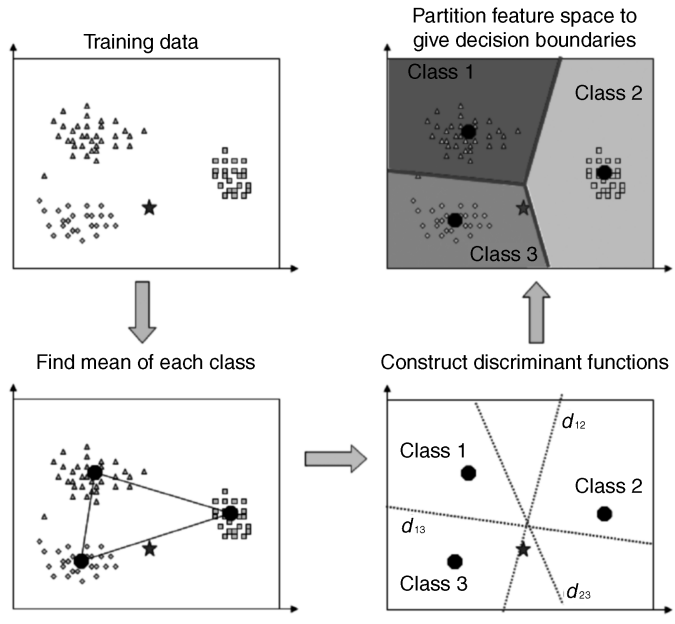
In principle, we may calculate a linear discriminant of this kind for every pair of classes. Figure 11.5 depicts the data in the simple three-class problem described in Section 11.3; therefore, we have $\binom{3}{2} = 3$ linear discriminant functions. By calculating linear discriminants for all pairs of prototypes and evaluating each discriminant function, it is possible in principle to classify a feature vector simply by observing the signs of the discriminants and applying basic logic (see table in Figure 11.5).

However, it is normal practice in N-class problems to define discriminant functions (whether linear or otherwise) in the following way:

$$\begin{array}{ll} \text{if} & g_j(\mathbf{x}) > g_k(\mathbf{x}) \quad \text{for all } j \neq k \\ \text{then} & \text{assign } \mathbf{x} \text{ to class } j \end{array} \tag{11.3}$$

It is evident, then, that the discriminant functions for each class consist of *segments of such linear functions* and combine to define *decision boundaries* which partition the feature space into distinct regions. Figure 11.5 (top right) shows the *decision boundaries* in the 2-D feature space of our example.

Consider male and female faces; these differ in visual appearance, and human observers are generally very adept at distinguishing between them. Could this be achieved through an automatic classifier? In Example 11.1, we use a 2-D feature vector based on the sixth and seventh component values extracted through a PCA on suitably normalized frontal images of the human face. Ten of these subjects are male and ten female and the region of the face subjected to the PCA is shown in Figure 11.6. It is apparent that use of just these two components does a reasonably good job and a simple (Euclidean) minimum distance classifier results in just two misclassifications.



d_{12}	d_{13}	d_{23}	Assigned class
+ve	+ve	+ve	3
+ve	+ve	+ve	2
+ve	-ve	-ve	2
+ve	-ve	+ve	Not possible
-ve	+ve	+ve	3
-ve	+ve	-ve	Not possible
-ve	-ve	-ve	1
-ve	-ve	+ve	1

Figure 11.5 Illustration of the minimum distance classifier, the construction of linear discriminant functions and the resulting decision boundaries

Example 11.1

Matlab Example 11.1	**What is happening?**
load PCA_data.mat;	%Load PCA coefficients for males %and females
f6=female(6,:)./1000; f7=female(7,:)./1000;	%extract female coefficients on %6 and 7
m6=male(6,:)./1000; m7=male(7,:)./1000;	%extract male coefficients on 6 and 7
F_proto=mean([f6' f7']); M_proto=mean([m6' m7']);	%calculate Male and female %prototypes
dely=M_proto(2)-F_proto(2); delx=M_proto(1)-F_proto(1);	%Difference between prototypes
mid_pt=[F_proto(1)+delx./2 F_proto(2)+dely./2];	%Midpoint between prototypes

```
grad=-1./(dely./delx);                          %Gradient of linear discriminant
a=1./(mid_pt(2)-grad.*mid_pt(1));               %Coefficients of linear discriminant
b=grad./(mid_pt(2)-grad.*mid_pt(1));

figure;
plot(f6,f7,'r*'); hold on; plot(m6,m7,'kd');    %Plot data
x=linspace(-5,5,100); y=(1+b.*x)./a;            %Plot linear discriminant
plot(x,y,'b-');
f_indx=find(a.*f7-b.*f6-1>0)                    %Find index of misclassified females
m_indx=find(a.*m7-b.*m6-1                       %Find index of misclassified males
```

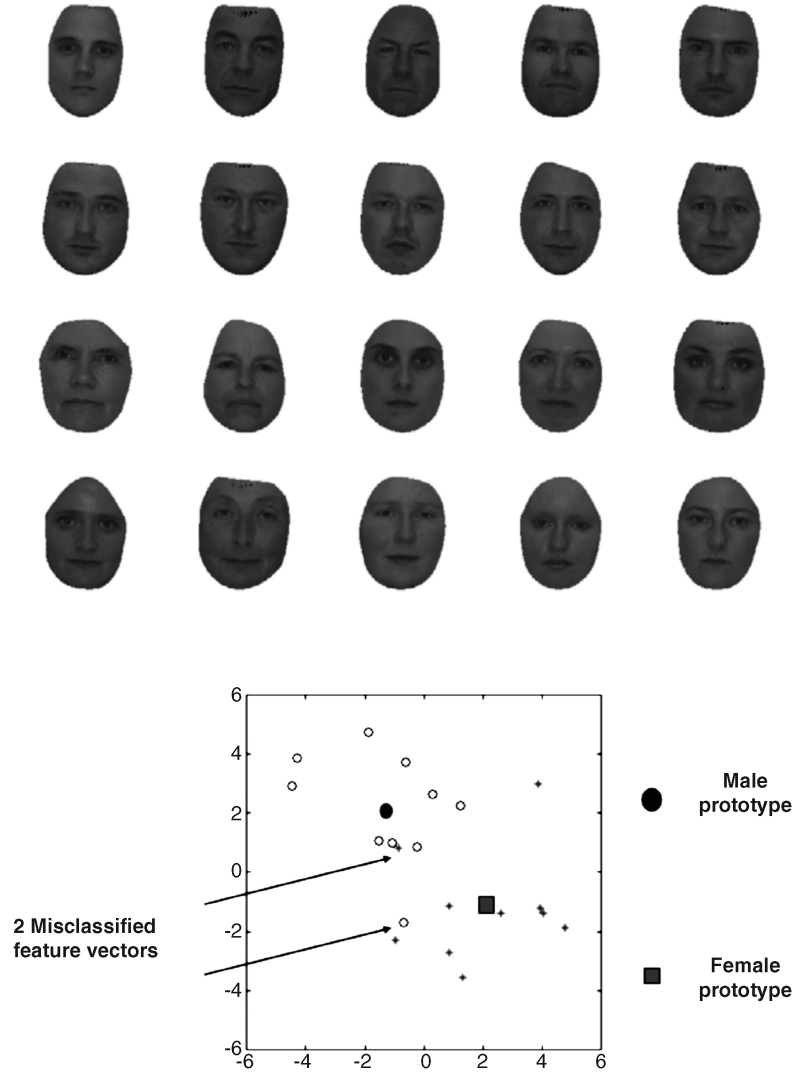


Figure 11.6 Thumbnail images of human faces and their respective positions in a 2-D feature space based on the 6th and 7th principal components. Male faces are plotted as circles and female faces as asterisks. A minimum distance classifier results in two misclassifications

Linear discriminants are important not just for simple minimum-distance classifiers, but find much wider application. For this reason, the next section extends the concept to a general N-dimensional feature space and looks at some of their key properties.

11.7 Linear discriminant functions in *N* dimensions

Although a number of interesting classification problems are two-class problems, the majority require the definition of multiple classes. The widespread popularity of linear discriminants in classification problems originates from their simplicity, easy computation and the fact that they extend simply and naturally to higher dimensional feature spaces.

The general form of the N-dimensional, linear discriminant $g(\mathbf{x})$ is

$$g(\mathbf{x}) = g(x_1, x_2 \cdots x_N) = \sum_{k=0}^{N} w_k x_k = w_0 + w_1 x_1 + w_2 x_2 + \cdots w_N x_N = 0$$

or expressed in compact, vector form

$$g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0 = 0 \tag{11.4}$$

For a 2-D feature space, the discriminant describes the equation of a straight line and divides a plane into two partitions. For a 3-D feature space, the discriminant describes the equation of *a plane* and divides a volume into two partitions. In the most general N-dimensional case, the discriminant describes the equation of *a hyperplane* and divides the N-dimensional volume into two partitions.

We note two important properties. Let us consider two arbitrary but distinct vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ both lying on the hyperplane. Since $g(\mathbf{x}) = 0$ for any $\mathbf{x}$ that lies on the hyperplane, it follows that

$$g(\mathbf{x}_1) - g(\mathbf{x}_2) = 0 \quad \Rightarrow \quad \mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 0 \tag{11.5}$$

Since the arbitrary vector $(\mathbf{x}_1 - \mathbf{x}_2)$ lies *within* the plane,[5] we conclude that the vector $\mathbf{w}$ is perpendicular to any vector lying in the hyperplane.

The discriminant function $g(\mathbf{x})$ also gives an algebraic measure of the shortest (i.e. perpendicular) distance s of the point $\mathbf{x}$ to the dividing hyperplane. It is possible to show that this is:

$$s = \frac{g(\mathbf{x})}{\|\mathbf{w}\|} \tag{11.6}$$

[5] More precisely, the vector $(\mathbf{x}_1 - \mathbf{x}_2)$ lies within the subspace defined by the hyperplane.

11.8 Extension of the minimum distance classifier and the Mahalanobis distance

The basic (Euclidean) minimum distance classifier certainly has the virtue of simplicity and is adequate for some applications. However, a little consideration will reveal that it has clear limitations. A classifier that only considers the mean of a distribution of feature vectors (i.e. prototypes) but takes no account of the *statistical distribution of feature vectors about that mean* cannot generally be optimal. Example 11.2 and the resulting Figure 11.7 illustrate this point graphically. In this problem, we simulate two classes of object lying within an arbitrary 2-D feature space. Class 1 (depicted by blue points) and class 2 (depicted by red points) are both normally distributed but have different means and covariances. Clearly, the Euclidean distance of many feature vectors originating from class 1 to the prototype of the second group is less than the distance to the first prototype. They are nonetheless *more likely to have originated from the first distribution*. It is readily apparent that a considerable number of misclassifications result even on the training data (the points additionally circled in green).

Apart from the Euclidean definition, there are a number of possible definitions of distance or *metric* within a feature space. A sensible adjustment we can, then, is to *reconsider our definition of distance in the feature space* to give a probabilistic measure. Referring to Table 11.2, we draw attention here to the *Mahalanobis distance*. For a feature vector $\mathbf{x} = [x_1, x_2 \cdots, x_N]$ belonging to a class with mean vector μ and covariance Σ, this

Example 11.2

Matlab Example 11.2 **What is happening?**

```
clear; NOPTS=200;
mu1 = [1 -1]; cov1 = [.9 .4; .4 .3];                %Define mean and covariance
d1 = mvnrnd(mu1, cov1, NOPTS);                      %Training data for class 1
plot(d1(:,1),d1(:,2),'b.'); hold on;                %Plot class 1 training data
proto1=mean(d1); plot(proto1(1),proto1(2),'ks');    %Calculate & plot prototype 1

[mu2 = [-1 1]; cov2 = [2 0; 0 2];                   %Define mean and covariance
d2 = mvnrnd(mu2, cov2, NOPTS);                      %Training data for class 1
plot(d2(:,1),d2(:,2),'r.');                         %Plot class 2 training data
proto2=mean(d2); plot(proto2(1),                    %Calculate & plot prototype 2
proto2(2),'ks');
axis([-6 4 -5 5]); axis square;

for i=1:NOPTS
  if norm(d2(i,:)-proto2,2) > norm(d2(i,:)-proto1,2)   %Misclassified points
      plot(d2(i,1),d2(i,2),'go');                      %Plot in green circle
    end
end
```

Comments

Matlab functions ***mvnrnd, norm***

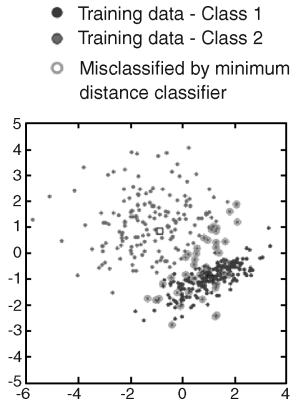


Figure 11.7 Feature vectors which are statistically more likely to have originated from one class may lie closer to the prototype of another class. This is a major weakness of a simple Euclidean minimum distance classifier *(See colour plate section for colour version)*

is defined as

$$D_{\text{mh}} = (\mathbf{x}-\boldsymbol{\mu})^{\text{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) \tag{11.7}$$

and the covariance matrix $\boldsymbol{\Sigma}$ in Equation (11.7) is defined by:

$$\boldsymbol{\Sigma} = \langle(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^{\text{T}}\rangle = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1N} \\ \sigma_{21} & \sigma_2^2 & & \sigma_{2N} \\ \vdots & & \ddots & \vdots \\ \sigma_{N1} & \sigma_{N2} & \cdots & \sigma_N^2 \end{bmatrix} \tag{11.8}$$

with $\sigma_{ij} = \langle(x_i-\mu_i)(x_j-\mu_j)\rangle$ and $\langle\ \rangle$ denotes the expectation or ensemble average over the sample.

The Mahalanobis distance effectively scales the absolute distances in the feature space by their corresponding standard deviations and, thus, provides an intuitively more 'probabilistic' measure of distance. In the following section, we develop the basic ideas behind classifiers that attempt to make full use of any *information we might possess about the statistical distribution of the feature vectors* within the space. This is the basic idea behind *Bayesian classifiers*.

11.9 Bayesian classification: definitions

The Bayesian approach to optimizing classifier performance is probabilistic in nature. From an operational perspective, a good classifier should obviously maximize the number of correct classifications and minimize the number of incorrect classifications over a definitive

number of trials. A little thought, however, will reveal that these are to some degree conflicting requirements[6] and cannot be used as a design principle. Bayesian classifiers are actually designed to assign each feature vector to the *most probable class*. It is probably worth saying at the outset that the Bayesian approach corresponds strictly to a rather idealized situation (in which all relevant probabilities are known) and these conditions are not often met in practice. Despite this, it forms an elegant framework, serving as both an adequate approximation to many problems and as an excellent point of reference for practical classifiers.

We begin by outlining some general definitions and notation. We will assume the following:

- The patterns in whose classification we are interested may be represented as N-dimensional feature vectors $\mathbf{x} = [x_1, x_2, \cdots, x_N]$ whose components $x_1, x_2, \cdots, x_N$ are the measurement parameters which describe the feature space.[7]

- There are a total of C pattern classes, $\omega_1, \omega_2 \cdots \omega_C$ to which a given feature vector $\mathbf{x}$ may be assigned.

- The probability of drawing a feature vector from each of the classes is known and described by the probabilities $p(\omega_1), p(\omega_2), \cdots, p(\omega_C)$. These are known as the *priors*.

- There exists a set of M such examples of feature vectors $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_M\}$ whose corresponding pattern classes are known. These examples form the *design* or *training* set.

11.10 The Bayes decision rule

Let the conditional probability $p(\omega_j|\mathbf{x})$ represent the probability density of belonging to class ω_j given the occurrence of the feature vector $\mathbf{x}$. Note that, in general, $p(\omega_j|\mathbf{x})$ should be interpreted as a probability density (rather than a strict probability), since the feature vector $\mathbf{x}$ is a continuous variable and, thus, $p(\omega_j|\mathbf{x})$ is a function conditional upon $\mathbf{x}$. The Bayes decision rule is an intuitive rule based on probabilities:

$$\begin{aligned} &\text{if} \quad p(\omega_j|\mathbf{x}) > p(\omega_k|\mathbf{x}) \quad \text{for all } k \neq j \\ &\text{then} \quad \text{assign } \mathbf{x} \text{ to } \omega_j \end{aligned} \tag{11.9}$$

[6] For example, if I wish to ensure that patterns from class A are always correctly classified then I can ensure this most easily by *always* assigning to class A whatever the pattern. The consequences for patterns belonging to other classes are obvious.

[7] Naturally, the measurement parameters should correspond directly or indirectly to those specific features thought to be important for classification.

In other words, we simply assign $\mathbf{x}$ to the most probable class. This decision rule, however, begs the question of how to find or estimate the probabilities $p(\omega_j|\mathbf{x})$ in Equation (11.9). To resolve this, we invoke Bayes' theorem, which for conditional probabilities extends naturally to the notion of probability densities and says

$$p(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)p(\omega_j)}{p(\mathbf{x})} \tag{11.10}$$

On substitution into Equation (11.9), our decision rule then becomes

$$\begin{aligned} &\text{if} \quad p(\mathbf{x}|\omega_j)p(\omega_j) > p(\mathbf{x}|\omega_k)p(\omega_k) \quad \text{for all } k \neq j \\ &\text{then} \quad \text{assign } \mathbf{x} \text{ to } \omega_j \end{aligned} \tag{11.11}$$

It is important to interpret the terms occurring in Equations (11.10) and (11.11) clearly:

- $p(\mathbf{x}|\omega_j)$ is known as the *class-conditional probability density function*. The term $p(\mathbf{x}|\omega_j)$ may be interpreted as 'the likelihood of the vector $\mathbf{x}$ occurring when the feature is known to belong to class ω_j.

- The functions $p(\omega_j)$ are known as the *prior probabilities*. Their values are chosen to reflect the fact that not all classes are equally likely to occur. Clearly, the values chosen for the $p(\omega_j)$ should reflect one's knowledge of the situation under which classification is being attempted.

- $p(\mathbf{x})$ is sometimes known as the 'evidence' and can be expressed using the partition theorem or 'law of total probability' as $p(\mathbf{x}) = \sum_j p(\mathbf{x}|\omega_j)p(\omega_j)$. Note that $p(\mathbf{x})$ is independent of the class label ω_j and does not enter into the decision rule described by Equation (11.11).

The key thing to note here is that application of the Bayes decision rule given by Equation (11.11) requires *knowledge of all the class-conditional, probability density functions and the priors*.

It is rare that we will know the functional forms and parameters of these density functions exactly. The approach typically taken in practice is to assume (or estimate) suitable functional forms for the distributions $p(\mathbf{x}|\omega_j)$ and $p(\omega_j)$ for all values of j and estimate their parameters from our training data. We then attempt to apply the rule given by Equation (11.11).

In certain instances, the form of the distribution may actually be known or confidently predicted. By far the most common is the multivariate normal (MVN) distribution. The main reason for this is its analytical tractability and simple properties (other distributions in N dimensions are *much* harder to deal with). However, real-life situations in which the MVN distribution is a fairly good approximation are fortunately quite common.

Before we deal with the specific cases of Bayesian classifiers in which the underlying distributions can be well approximated by the MVN, we will, therefore, dedicate the next section to reviewing its properties.

11.11 The multivariate normal density

The one-dimensional Gaussian or normal density for the random variable x is given by

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{11.12}$$

where μ is the mean of the distribution and σ is the standard deviation. The 1-D normal density is thus defined by just two parameters. For multivariate data $x_1, x_2 \cdots x_N$, represented as a column vector $\mathbf{x} = [x_1, x_2 \cdots x_N]^{\mathrm{T}}$, the normal distribution has the more general form

$$p(\mathbf{x}) = \frac{1}{(2\pi)N/2|\mathbf{C}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\vec{\mu}_x)^{\mathrm{T}}\mathbf{C}_x^{-1}(\mathbf{x}-\vec{\mu}_x)\right] \tag{11.13}$$

where $\vec{\mu}_x = E(\mathbf{x})$ is the expectation or average value of $\mathbf{x}$ and $\mathbf{C}_x = E[(\mathbf{x}-\vec{\mu}_x)(\mathbf{x}-\vec{\mu}_x)^{\mathrm{T}}]$ is the covariance matrix.

The covariance matrix contains the covariance between every pair of variables in the vector $\mathbf{x}$. Thus, the general, *off-diagonal* element is given by

$$\mathbf{C}_x(i,j) = E[(x_i-\mu_i)(x_j-\mu_j)] \qquad i \neq j \tag{11.14}$$

where $\mu_k = E(x_k)$ and the diagonal elements contain the variances

$$\mathbf{C}_x(i,i) = E[(x_i-\mu_i)^2] \qquad i = 1, 2, \cdots, N \tag{11.15}$$

By definition, the covariance matrix is a symmetric matrix and there are $N(N+1)/2$ free parameters. Combining these with the N free parameters for the average vector $\vec{\mu}_x = E(\mathbf{x})$, a total of $N(N+3)/2$ parameters are required to completely specify the MVN density.

We note in passing an important feature of the MVN density. The quadratic form which appears in the exponential is actually the (squared) Mahalanobis distance which we mentioned in Section 11.8:

$$L^2 = (\mathbf{x}-\vec{\mu}_x)^{\mathrm{T}}\mathbf{C}_x^{-1}(\mathbf{x}-\vec{\mu}_x) \tag{11.16}$$

Setting this term to a constant value, we obtain contours of constant probability density. It is possible to show that this quadratic form defines an hyper-ellipsoid, which reduces to an ellipsoid in three dimensions and to an ellipse in two dimensions and, thus, determines the orientation and spread of the distribution of points within the feature space.

11.12 Bayesian classifiers for multivariate normal distributions

For an arbitrary class ω_j with features $\mathbf{x}$ distributed as an MVN distribution having mean $E(\mathbf{x}) = \vec{\mu}_i$ and covariance matrix $\mathbf{C}_j$, the class-conditional density is given by

$$p(\mathbf{x}|\omega_j) = \frac{1}{(2\pi)N/2|\mathbf{C}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\vec{\mu}_i)^{\mathrm{T}}\mathbf{C}_j^{-1}(\mathbf{x}-\vec{\mu}_i)\right] \tag{11.17}$$

The Bayes decision rule states that

$$\begin{array}{ll} \text{if} & p(\mathbf{x}|\omega_j)p(\omega_j) > p(\mathbf{x}|\omega_k)p(\omega_k) \quad \text{for all } k \neq j \\ \text{then} & \text{assign } \mathbf{x} \text{ to } \omega_j \end{array} \tag{11.18}$$

Taking logarithms on both sides of the Bayes decision rule and noting that log is a monotonically increasing function, we obtain the equivalent rule

$$\begin{array}{ll} \text{if} & g_j(\mathbf{x}) = \log p(\mathbf{x}|\omega_j) + \log p(\omega_j) > g_k(\mathbf{x}) = \log p(\mathbf{x}|\omega_k) + \log p(\omega_k) \quad \text{for all } k \neq j \\ \text{then} & \text{assign } \mathbf{x} \textit{ to } \omega_j \end{array} \tag{11.19}$$

Substituting the normal form for the class-conditional density function, it is easy to show that the discriminant functions take the form

$$\begin{aligned} g_j(\mathbf{x}) &= \log p(\mathbf{x}|\omega_j) + \log p(\omega_j) \\ &= -\frac{1}{2}(\mathbf{x}-\vec{\mu}_x)^{\mathrm{T}}\mathbf{C}_j^{-1}(\mathbf{x}-\vec{\mu}_x) - \frac{N}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{C}_j| + \log p(\omega_j) \end{aligned} \tag{11.20}$$

These discriminant functions take simpler forms when the covariance matrix $\mathbf{C}_j$ possesses certain properties. First note, however, that the term $(N/2)\log 2\pi$ is the same for all classes. Since our task is to compare discriminants of the form given by Equation (11.20), it can thus be removed from Equation (11.20).

Case I: $\mathbf{C}_i = \sigma^2\mathbf{I}$ If we have a situation in which the features in $\mathbf{x}$ are statistically independent and all have the same variance, then the discriminant for MVN densities reduces to the form

$$\begin{aligned} g_j(\mathbf{x}) &= \log p(\mathbf{x}|\omega_j) + \log p(\omega_j) \\ &= -\frac{\|\mathbf{x}-\vec{\mu}_x\|^2}{2\sigma^2} + \log p(\omega_j) \end{aligned} \tag{11.21}$$

Expanding the term $\|\mathbf{x}-\vec{\mu}_x\|^2 = (\mathbf{x}-\vec{\mu}_x)^{\mathrm{T}}(\mathbf{x}-\vec{\mu}_x)$ and noting that $\mathbf{x}^{\mathrm{T}}\mathbf{x}$ is the same for all classes and can thus be ignored, we thereby obtain a *linear discriminant* of the standard form

$$g_j(\mathbf{x}) = \mathbf{w}_j^{\mathrm{T}}\mathbf{x} + v_j \tag{11.22}$$

where

$$\mathbf{w}_j = \frac{\vec{\mu}_j}{\sigma^2} \quad \text{and} \quad v_j = -\frac{1}{2\sigma^2}\vec{\mu}_j^{\mathrm{T}}\vec{\mu}_j + \log p(\omega_j)$$

A classifier that uses linear discriminant functions is called a linear machine. The decision surfaces for a linear machine are thus determined by the intersection of hyperplanes as described by Equation (11.22). Note from Equation (11.22) that, in the case that all classes have the same a priori likelihood of occurring (i.e. $p(\omega_j)$ is the same for all classes), the expression *reduces to the minimum distance criterion* discussed in earlier sections.

Case II: $\mathbf{C}_j = \mathbf{C}$ When the covariance matrices may be assumed to be the same for all classes (i.e. $\mathbf{C}_j = \mathbf{C}$ for all j), we again substitute Equation (11.20) into Equation (11.19), simplify and obtain a linear discriminant:

$$g_j(\mathbf{x}) = \mathbf{w}_j^{\mathrm{T}}\mathbf{x} + v_j \tag{11.23}$$

where this time

$$\mathbf{w}_j = \mathbf{C}^{-1}\vec{\mu}_j \quad \text{and} \quad v_j = -\frac{1}{2}\vec{\mu}_j^{\mathrm{T}}\mathbf{C}\vec{\mu}_j + \log p(\omega_j).$$

In Example 11.3 and the resulting Figure 11.8, we compare the results of these classifiers on some test data for the two-class problem we presented in Section 11.8.

Example 11.3

Matlab Example 11.3	**What is happening?**
NOPTS=200;	
mu1 = [1 -1]; cov1 = [.9 .4; .4 .3];	%Define mean and covariance
d1 = mvnrnd(mu1, cov1, NOPTS);	%Training data for class 1
subplot(2,2,1), plot(d1(:,1),d1(:,2),'b.'); hold on;	%Plot class 1 training data
proto1=mean(d1); plot(proto1(1),proto1(2),'ks');	%Calculate and plot prototype 1
mu2 = [-1 1]; cov2 = [2 0; 0 2];	%Define mean and covariance
d2 = mvnrnd(mu2, cov2, NOPTS);	%Training data for class 1
plot(d2(:,1),d2(:,2),'r.');	%Plot class 2 training data
proto2=mean(d2); plot(proto2(1),proto2(2),'ks');	%Calculate and plot prototype 2
axis([-6 4 -5 5]); axis square;	
title('TRAINING DATA')	
group=[ones(NOPTS,1); 2.*ones(NOPTS,1)];	%vector to specify actual classes of %training points
%for i=1:NOPTS	
% if norm(d2(i,:)-proto2,2) > norm(d2(i,:)-proto1,2)	%Find misclassified points
% plot(d2(i,1),d2(i,2),'go');	%Plot on top in green circles
% end	
%end	
	%%%%%%%%%%%%%%%%%%
	%CLASS = CLASSIFY(SAMPLE, TRAINING,GROUP,TYPE)

```
                                                    %generate TEST DATA

N1=50; N2=100;                                      %Number of test points from classes
testgroup=[ones(N1,1); 2.*ones(N2,1)];              %vector to specify actual class of
                                                    %test points
sample=[mvnrnd(mu1, cov1, N1);
mvnrnd(mu2, cov2, N2)];                             %generate test data points

subplot(2,2,2),
plot(proto1(1),proto1(2),'ks');
axis([-6 4 -5 5]); axis square; hold on;
plot(proto2(1),proto2(2),'ks');
plot(sample(1:N1,1),sample(1:N1,2),'b*');
plot(sample(N1+1:N1+N2,1),sample(N1+1:N1+N2,2),'r*');
title('TEST DATA');

[class,err]=classify(sample,[d1;d2],group,
   'DiagLinear',[0.9 0.1]);                         %Classify using diagonal
                                                    %covariance estimate
subplot(2,2,3),
plot(proto1(1),proto1(2),'ks');
axis([-6 4 -5 5]); axis square; hold on;
plot(proto2(1),proto2(2),'ks');
plot(sample(1:N1,1),sample(1:N1,2),'b*');
plot(sample(N1+1:N1+N2,1),sample(N1+1:N1+N2,2),'r*');

for i=1:size(class,1)
   if class(i)~=testgroup(i)
      plot(sample(i,1),sample(i,2),'go');
   end
end

[class,err]=classify(sample,[d1;d2],group,'DiagLinear',[N1./(N1+N2) N2./(N1+N2)]);
   %Classify using diagonal covariance estimate
subplot(2,2,4),
plot(proto1(1),proto1(2),'ks');
axis([-6 4 -5 5]); axis square; hold on;
plot(proto2(1),proto2(2),'ks');
plot(sample(1:N1,1),sample(1:N1,2),'b*');
plot(sample(N1+1:N1+N2,1),sample(N1+1:N1+N2,2),'r*');

for i=1:size(class,1)
   if class(i)~=testgroup(i)
      plot(sample(i,1),sample(i,2),'go');
   end
end
```

Comments

Matlab functions ***mvnrnd, norm:*** mvnrnd is a Matlab function that generates arrays of normally distributed random variables

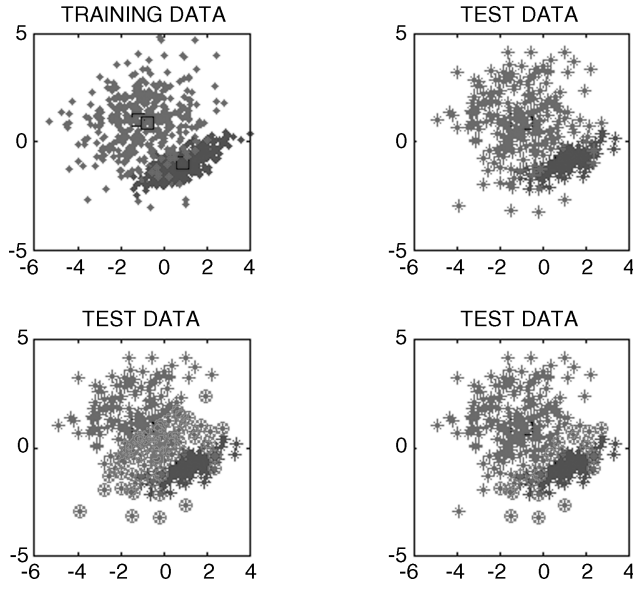


Figure 11.8 A comparison of Bayesian classifiers under different assumptions about the covariance matrix and prior distributions. The class-conditional densities of the training data are both normal. (a) Training data: 200 samples from each class (class 1: blue; class 2: red). (b) Test data: 50 originate from class 1, 100 originate from class 2 (indicated in red). (c) Classification using a Bayesian classifier with a diagonal estimate of the covariance and an erroneous prior distribution of $p(\omega_1) = 0.9$ and $p(\omega_2) = 0.1$. Misclassified test points are circled in green. (d) Classification using a Bayesian classifier with a diagonal estimate of the covariance and correct prior distribution of $p(\omega_1) = 1/3$ and $p(\omega_2) = 2/3$. Misclassified test points are circled in green *(See colour plate section for colour version)*

11.12.1 The Fisher linear discriminant

For simplicity, let us initially consider the case of a two-class problem in a feature space where the training data are described by feature vectors $\{\mathbf{x}_i\}$. The basic aim of the Fisher linear discriminant (FLD) is to find that axis from among all the possible axes we could choose within the feature space such that projection of the training data onto it will ensure that the classes are *maximally separated*. The projection of the data $\{\mathbf{x}_i\}$ is given by some linear combination $\mathbf{w}^T\mathbf{x}$ with $\mathbf{w}$ to be determined. This basic idea is demonstrated in Figure 11.9 for a 2-D feature space. Note, however, that the concept extends naturally to three dimensions and higher, since we can always project data of an arbitrary dimensionality orthogonally onto a line.

The formal calculation of the FLD relies on maximizing the criterion function:

$$J(\mathbf{w}) = \frac{|m_A - m_B|^2}{s_A^2 + s_B^2}$$

where m_A and m_B are the means of the projected data and s_A^2 and s_B^2 are the sample variances of the projected data. Note how this criterion tries to maximize the separation of the projected means but scales them by the total variance of the data.

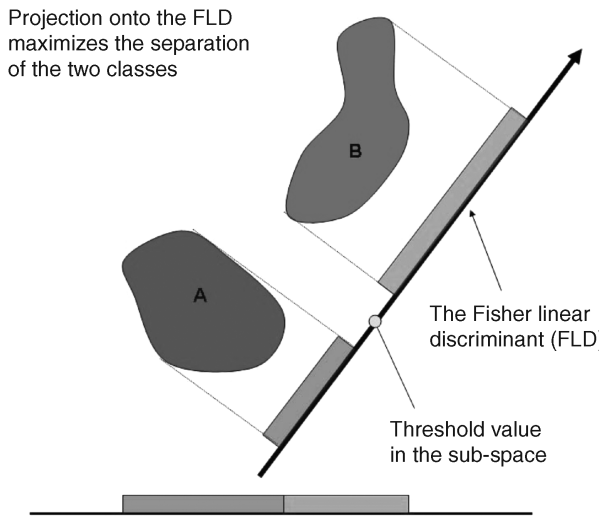


Figure 11.9 The FLD defines a direction within the feature space that maximizes the separation of the projected feature vectors in each class. This method can achieve a substantial reduction in the dimensionality of the problem (from N dimensions to 1). However, there is, of course, no guarantee that the data will be separable when projected onto a single direction

It is possible to show[8] that the solution is

$$\mathbf{w} = \mathbf{S}_w^{-1}[\mathbf{M}_{\mathrm{A}}^x - \mathbf{M}_{\mathrm{B}}^x]$$

where $\mathbf{M}_{\mathrm{A}}^x$ and $\mathbf{M}_{\mathrm{B}}^x$ are the vectors giving the means of the original (unprojected) data and the total scatter matrix $\mathbf{S}_w$ is given by the sum of the individual scatter matrices $\mathbf{S}_w = \mathbf{S}_{\mathrm{A}} + \mathbf{S}_{\mathrm{B}}$ which, in general, are of the form

$$\mathbf{S}_K = \sum_{\substack{\text{all } \mathbf{x} \\ \text{in class } K}} (\mathbf{x} - \mathbf{M}_K^x)(\mathbf{x} - \mathbf{M}_K^x)^{\mathrm{T}}$$

Once the optimal $\mathbf{w}$ has been found, the classification is then simply a matter of deciding the threshold value along the axis (subspace) defined by the discriminant.

The FLD for the two-class problem reduces the N-dimensional feature space to a single dimension. Whilst such an approach naturally sacrifices the best possible performance for a classifier, this can be acceptable if the advantages of working in just one dimension compensate.

11.12.2 Risk and cost functions

It is self-evident that a basic aim of classification is to assign the maximum number of feature vectors to their correct class whilst making as few misclassifications as possible.

[8] See, for example, Duda *et al.* (2001), *Pattern Classification*, John Wiley & Sons, Inc., pp. 117–124.

However, we should recognize that real-life situations are invariably more subtle than this. Depending on the specific situation, *the relative importance of misclassifications can vary significantly*. For instance, a significant amount of research has been devoted to the development of image processing software for automatic breast-screening. Regular screening of the female population was introduced because of the much improved prognosis which generally results if breast cancer can be detected in its early stages. The huge workload involved in manual inspection of these mammograms has motivated the search for reliable, automated methods of diagnosis. The purpose of the automated analysis is essentially to classify to the classes of either 'normal' (in which case no further action is taken) or 'abnormal' (which generally results in further more careful inspection by a radiologist). Abnormal scans are not always the result of abnormal pathology, but because the consequences of classifying what is actually an abnormal mammogram as 'normal' (i.e. missing a pathological condition) can be very serious, the classification tends towards a 'play-safe strategy'. In other words, we associate a *greater cost* or *risk* with a misclassification in one direction. The cost of a *false-negative* event (in which an abnormal scan is assigned to the normal class) is considerably greater than the other kind of misclassification (the *false positive* event, in which a normal scan is assigned to the abnormal class). A reverse philosophy operates in courts of law, hence the old saying 'Better one hundred guilty men go free than one innocent man be convicted'.

By contrast, some financial organizations are beginning to employ sophisticated software to assess the normality of their customers spending patterns, the idea being that unusual and/or excessive spending is a likely indicator of fraudulent use of a card. The prevailing philosophy tends towards the idea that it is better to accept a certain modest amount of fraudulent use than to generate many false-alarms in which the legitimate owner of the card is refused the use of their card.

It is outside the scope of this text to discuss risk or cost function analysis in image classification problems. The theory has been extensively developed and the interested reader can refer to the literature.[9]

11.13 Ensemble classifiers

Ensemble classifiers are classification systems that combine a set of component classifiers in the attempt to achieve an overall performance that is better than any of the individual, components. The principle of consulting several 'experts' before making a final decision is familiar to us in our everyday lives. Before we decide to purchase a particular model of car, undergo a medical procedure or employ a particular job candidate, we typically solicit opinions from a number of sources. Combining the information from our various sources, and weighting them according to our innate level of confidence in each, we generally feel more confident about the final decision we reach.

Automated ensemble classifiers try to achieve a similar goal according to strict mathematical (or at least sound, empirical) principles. A detailed discussion of ensemble classifiers is beyond the scope of this book and we will only attempt here to describe one such ensemble

[9] A good starting place is Pattern classification, Duda et al, Wiley 2001

classifier, namely the AdaBoost method, which is rapidly emerging as the most important. Central to this method is the requirement that the component classifiers be as *diverse* as possible. It stands to reason that if we had a single classifier that had perfect generalization to new examples, then we would have no need at all to consider ensemble systems. It is thus implicit that the component classifiers in an ensemble system are imperfect and make misclassifications. Intuitively, the hope would be that each imperfect component misclassifies different examples and that the combination of the components would reduce the total error. The notion of diversity encapsulates the idea that each classifier be as unique as possible.

11.13.1 Combining weak classifiers: the AdaBoost method

A so-called 'weak classifier' is basically a classifier that does not perform well. Indeed, a weak classifier may perform only slightly better than chance.

Consider a set of n feature vectors to constitute our training sample. A subset of these n vectors is first randomly selected to form the training sample for the first component classifier. The probability of random selection for each vector is determined by its associated weight. After the first component classifier has been trained on the data, we increase the weights of those vectors which have been incorrectly classified but decrease the weights of those vectors which have been correctly classified. In this way, we increase the probability that the next component classifier will be trained on more 'troublesome/difficult' feature vectors. The process is then repeated for the second (and subsequent) component classifiers, adjusting the weights associated with each feature vector. Thus, in general, on the kth iteration we train the kth component classifier.

Provided each component classifier performs better than chance (i.e. has an error of less than 0.5 on a two-class problem), it is possible to show that AdaBoost can achieve an arbitrarily small misclassification error. Moreover, there is no restriction placed on the nature of the component classifiers used; thus, the AdaBoost method is generic in nature. This powerful result is not, however, an instant panacea. The choice of component classifiers which satisfy the 'better than chance' requirement cannot be guaranteed beforehand. Even so, AdaBoost performs well in many real problems and has been described as 'the best off-the-shelf classifier in the world'.

11.14 Unsupervised learning: k-means clustering

Unsupervised classifiers do not have the benefit of class-labelled examples to train on. Rather, they simply explore the data and search for naturally occurring patterns or clusters within it. Once these clusters have been found, we may then construct decision boundaries to classify unseen data using broadly similar methods to those used for supervised classifiers. One of the simplest unsupervised learning algorithms is the k-means algorithm.

This procedure, summarized in Figure 11.10 (the figure is generated using the Matlab® code in Example 11.4), outlines a conceptually simple way to partition a data set into a specified number of clusters k. The algorithm aims to iteratively minimize a simple

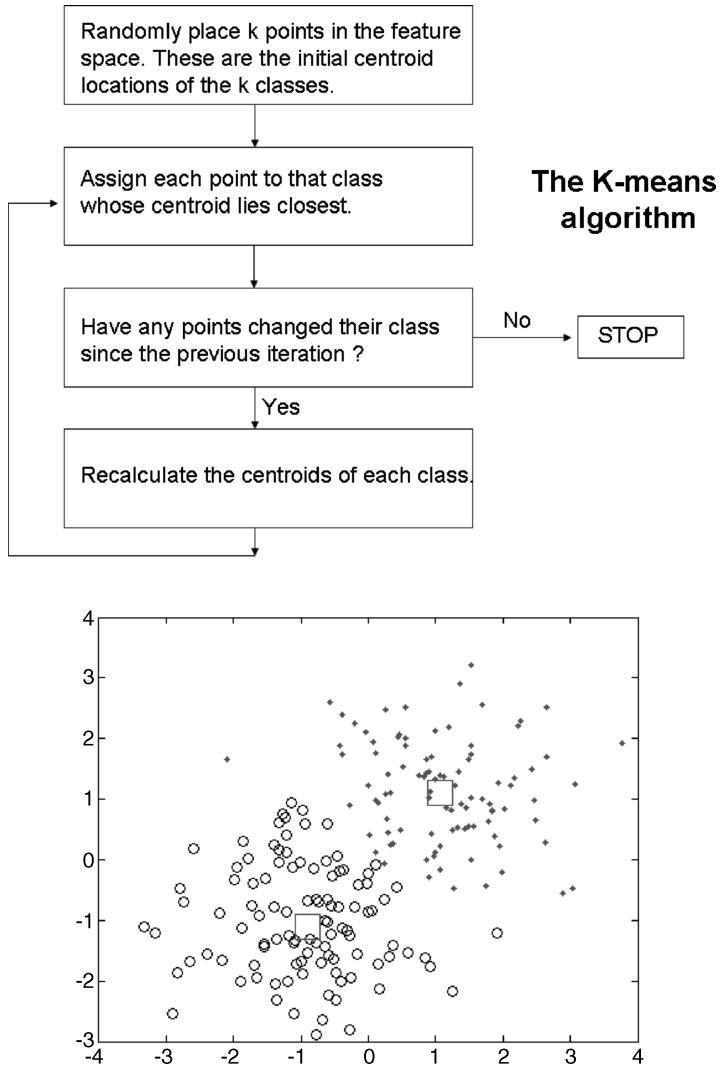


Figure 11.10 The k means algorithm assigns the data to k classes such that the sum of the squared distances from each point to the centroid of the assigned class is as small as possible. This example is for k=2 classes (see example 11.4)

squared error objective function of the form

$$J = \sum_{j=1}^{k} \sum_{\substack{\text{all } i \\ \text{in class } j}} |\mathbf{x}_i^j - \mathbf{c}_j|^2,$$

where $\mathbf{c}_k$ denotes the coordinate vector of the jth cluster and $\{\mathbf{x}_i^j\}$ are the points assigned to the jth cluster. Minimizing J equivalently means reaching that configuration at which switching any point to a cluster other than its currently assigned one will only *increase* the

Example 11.4

Matlab Example 11.4	**%What is happening?**
X = [randn(100,2)+ones(100,2); randn(100,2)-ones(100,2)];	%Generate random data array
opts = statset('Display','final');	%Specify statistics options structure
[cidx, ctrs] = kmeans(X, 2, 'Distance','city','Replicates',5, 'Options',opts);	%Run 5 repeats, city-block metric
plot(X(cidx==1,1),X(cidx==1,2),'r.');	%Plot 1st class points ...
hold on; plot(X(cidx==2,1),X(cidx==2,2),'ko');	%Plot 2nd class points ...
plot(ctrs(:,1),ctrs(:,2),'rs','MarkerSize',18);	%plot class centroids as squares

objective function. It is possible to show that the procedure outlined in Figure 11.10 generally achieves this goal, although it is somewhat sensitive to the initial choice of centroid locations. Typically, the *k*-means algorithm is run multiple times with different starting configurations to reduce this effect.

Arguably, the central weakness of the *k*-means algorithm is the need to fix the number of clusters '*a priori*'. Clearly, increasing the number of clusters will guarantee that the minimum value of the objective function reduces monotonically (ultimately, the objective function will have value zero when there as many clusters as points), so that the algorithm is not inherently sensitive to 'natural clusters' in the data. This last problem is troublesome, since we often have no way of knowing how many clusters exist.

Unfortunately, there is no general theoretical solution to find the optimal number of clusters for any given data set. A simple approach is to compare the results of multiple runs with different values of *k* and choose the best one according to a given criterion; but we need to exercise care, because increasing *k* increases the risk of overfitting.

For further examples and exercises see http://www.fundipbook.com.