8
Supervised Classification Techniques

The purpose of this chapter is to present the algorithms used for the supervised classification of single sensor remote sensing image data.

When data from a variety of sensors or sources (such as found in the integrated spatial data base of a Geographical Information System) requires analysis, more sophisticated tools may be required. These are the subject of Chap. 12 which deals with the topic of Multisource Classification.

8.1
Steps in Supervised Classification

Supervised classification is the procedure most often used for quantitative analysis of remote sensing image data. It rests upon using suitable algorithms to label the pixels in an image as representing particular ground cover types, or classes. A variety of algorithms is available for this, ranging from those based upon probability distribution models for the classes of interest to those in which the multispectral space is partitioned into class-specific regions using optimally located surfaces. Irrespective of the particular method chosen, the essential practical steps usually include:

1. Decide the set of ground cover types into which the image is to be segmented. These are the information classes and could, for example, be water, urban regions, croplands, rangelands, etc.
2. Choose representative or prototype pixels from each of the desired set of classes. These pixels are said to form *training data.* Training sets for each class can be established using site visits, maps, air photographs or even photointerpretation of a colour composite product formed from the image data. Often the training pixels for a given class will lie in a common region enclosed by a border. That region is then often called a *training field.*
3. Use the training data to estimate the parameters of the particular classifier algorithm to be used; these parameters will be the properties of the probability model

used or will be equations that define partitions in the multispectral space. The set of parameters for a given class is sometimes called the *signature* of that class.
4. Using the trained classifier, label or classify every pixel in the image into one of the desired ground cover types (information classes). Here the whole image segment of interest is typically classified. Whereas training in Step 2 may have required the user to identify perhaps 1% of the image pixels by other means, the computer will label the rest by classification.
5. Produce tabular summaries or thematic (class) maps which summarise the results of the classification.
6. Assess the accuracy of the final product using a labelled testing data set.

In practice it might be necessary to decide, on the basis of the results obtained at Step 6, to refine the training process in order to improve classification accuracy. Sometimes it might even be desirable to classify just the training samples themselves to ensure that the signatures generated at Step 3 are adequate.

It is our objective now to consider the range of algorithms that could be used in 3 and 4. In so doing it will be assumed that the information classes each consists of only one spectral class, so that the two names will be used synonomously. (See Chap. 3 for a discussion of the two class types.) By making this assumption, problems with establishing sub-classes will not distract from the algorithm development to be given. Handling sub-classes is taken care of explicitly in Chaps. 9 and 11.

In the following sections it is assumed that the reader is familiar at least with the sections on quantitative analysis contained in Chap. 3. This relates particularly to definitions and terminology.

8.2 Maximum Likelihood Classification

Maximum likelihood classification is the most common supervised classification method used with remote sensing image data. This is developed in the following in a statistically acceptable manner; it can be derived however in a more general and rigorous manner and this is presented for completeness in Appendix F. The present approach is sufficient though for most remote sensing exercises.

8.2.1 Bayes' Classification

Let the spectral classes for an image be represented by

$$\omega_i, i = 1, \ldots M$$

where M is the total number of classes. In trying to determine the class or category to which a pixel vector $\boldsymbol{x}$ belongs it is strictly the conditional probabilities

$$p(\omega_i|\boldsymbol{x}), i = 1, \ldots M$$

that are of interest. The measurement vector $\boldsymbol{x}$ is a column of brightness values for the pixel. It describes the pixel as a point in multispectral space with co-ordinates defined by the brightnesses, as shown in the simple two-dimensional example of Fig. 3.5. The probability $p(\omega_i|\boldsymbol{x})$ gives the likelihood that the correct class is ω_i for a pixel at position $\boldsymbol{x}$. Classification is performed according to

$$\boldsymbol{x} \in \omega_i, \quad \text{if} \quad p(\omega_i|\boldsymbol{x}) > p(\omega_j|\boldsymbol{x}) \quad \text{for all} \quad j \neq i \tag{8.1}$$

i.e., the pixel at $\boldsymbol{x}$ belongs to class ω_i if $p(\omega_i|\boldsymbol{x})$ is the largest. This intuitive *decision rule* is a special case of a more general rule in which the decisions can be biased according to different degrees of significance being attached to different incorrect classifications. The general approach is called Bayes' classification and is the subject of the treatment in Appendix F.

8.2.2 The Maximum Likelihood Decision Rule

Despite its simplicity, the $p(\omega_i|\boldsymbol{x})$ in (8.1) are unknown. Suppose however that sufficient training data is available for each ground cover type. This can be used to estimate a probability distribution for a cover type that describes the chance of finding a pixel from class ω_i, say, at the position $\boldsymbol{x}$. Later the form of this distribution function will be made more specific. For the moment however it will be retained in general terms and represented by the symbol $p(\boldsymbol{x}|\omega_i)$. There will be as many $p(\boldsymbol{x}|\omega_i)$ as there are ground cover classes. In other words, for a pixel at a position $\boldsymbol{x}$ in multispectral space a set of probabilities can be computed that give the relative likelihoods that the pixel belongs to each available class.

The desired $p(\omega_i|\boldsymbol{x})$ in (8.1) and the available $p(\boldsymbol{x}|\omega_i)$ - estimated from training data – are related by Bayes' theorem (Freund, 1992):

$$p(\omega_i|\boldsymbol{x}) = p(\boldsymbol{x}|\omega_i)\ p(\omega_i)/p(\boldsymbol{x}) \tag{8.2}$$

where $p(\omega_i)$ is the probability that class ω_i occurs in the image. If, for example, 15% of the pixels of an image happen to belong to class ω_i then $p(\omega_i) = 0.15$; $p(\boldsymbol{x})$ in (8.2) is the probability of finding a pixel from *any* class at location $\boldsymbol{x}$. It is of interest to note in passing that

$$p(\boldsymbol{x}) = \sum_{i=1}^{M} p(\boldsymbol{x}|\omega_i)\ p(\omega_i),$$

although $p(\boldsymbol{x})$ itself is not important in the following. The $p(\omega_i)$ are called *a priori* or prior probabilities, since they are the probabilities with which class membership of a pixel could be guessed before classification. By comparison the $p(\omega_i|\boldsymbol{x})$ are posterior probabilities. Using (8.2) it can be seen that the classification rule of (8.1) is:

$$\boldsymbol{x} \in \omega_i \quad \text{if} \quad p(\boldsymbol{x}|\omega_i)p(\omega_i) > p(\boldsymbol{x}|\omega_j)p(\omega_j) \quad \text{for all} \quad j \neq i \tag{8.3}$$

where $p(\boldsymbol{x})$ has been removed as a common factor. The rule of (8.3) is more acceptable than that of (8.1) since the $p(\boldsymbol{x}|\omega_i)$ are known from training data, and it

is conceivable that the $p(\omega_i)$ are also known or can be estimated from the analyst's knowledge of the image. Mathematical convenience results if in (8.3) the definition

$$\begin{aligned} g_i(\boldsymbol{x}) &= \ln \{p(\boldsymbol{x}|\omega_i)\ p(\omega_i)\} \\ &= \ln\ p(\boldsymbol{x}|\omega_i) + \ln\ p(\omega_i) \end{aligned} \tag{8.4}$$

is used, where ln is the natural logarithm, so that (8.3) is restated as

$$\boldsymbol{x} \in \omega_i \quad \text{if} \quad g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \quad \text{for all} \quad j \neq i \tag{8.5}$$

This is, with one modification to follow, the decision rule used in maximum likelihood classification; the $g_i(\boldsymbol{x})$ are referred to as *discriminant functions.*

8.2.3 Multivariate Normal Class Models

At this stage it is assumed that the probability distributions for the classes are of the form of multivariate normal models. This is an assumption, rather than a demonstrable property of natural spectral or information classes; however it leads to mathematical simplifications in the following. Moreover it is one distribution for which properties of the multivariate form are well-known.

In (8.4) therefore, it is now assumed for N bands that (see Appendix E)

$$p(\boldsymbol{x}|\omega_i) = (2\pi)^{-N/2}\,|\Sigma_i|^{-1/2} \exp\{-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^t\ \Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i)\} \tag{8.6}$$

where $\boldsymbol{m}_i$ and Σ_i are the mean vector and covariance matrix of the data in class ω_i. The resulting term $-N/2\ \ln\ (2\pi)$ is common to all $g_i(\boldsymbol{x})$ and does not aid discrimination. Consequently it is ignored and the final form of the discriminant function for maximum likelihood classification, based upon the assumption of normal statistics, is:

$$g_i(\boldsymbol{x}) = \ln\ p(\omega_i) - \tfrac{1}{2}\ln\,|\Sigma_i| - \tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^t\ \Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i) \tag{8.7}$$

Often the analyst has no useful information about the $p(\omega_i)$, in which case a situation of equal prior probabilities is assumed; as a result $\ln\ p(\omega_i)$ can be removed from (8.7) since it is then the same for all i. In that case the 1/2 common factor can also be removed leaving, as the discriminant function:

$$g_i(\boldsymbol{x}) = -\ln|\Sigma_i| - (\boldsymbol{x}-\boldsymbol{m}_i)^t\ \Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i) \tag{8.8}$$

Implementation of the maximum likelihood decision rule involves using either (8.7) or (8.8) in (8.5). There is a further consideration however concerned with whether any of the available labels or classes is appropriate. This relates to the use of thresholds as discussed in Sect. 8.2.5 following.

8.2.4 Decision Surfaces

As a means for assessing the capabilities of the maximum likelihood decision rule it is of value to determine the essential shapes of the surfaces that separate one class

from another in the multispectral domain. These surfaces, albeit implicit, can be devised in the following manner.

Spectral classes are defined by those regions in multispectral space where their discriminant functions are the largest. Clearly these regions are separated by surfaces where the discriminant functions for adjoining spectral classes are equal. The ith and jth spectral classes are separated therefore by the surface

$$g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = 0.$$

This is referred to as a *decision surface* since, if all the surfaces separating spectral classes are known, decisions about the class membership of a pixel can be made on the basis of its position relative to the complete set of surfaces.

The construction $(\boldsymbol{x} - \boldsymbol{m}_i)^t \ \Sigma_i^{-1}(\boldsymbol{x} - \boldsymbol{m}_i)$ in (8.7) and (8.8) is a quadratic function of $\boldsymbol{x}$. Consequently the decision surfaces implemented by maximum likelihood classification are quadratic and thus take the form of parabolas, circles and ellipses. Some indication of this can be seen in Fig. 3.8.

8.2.5 Thresholds

It is implicit in the foregoing development that pixels at every point in multispectral space will be classified into one of the available classes ω_i, irrespective of how small the actual probabilities of class membership are. This is illustrated for one dimensional data in Fig. 8.1a. Poor classification can result as indicated. Such situations can arise if spectral classes (between 1 and 2 or beyond 3) have been overlooked or, if knowing other classes existed, enough training data was not available to estimate the parameters of their distributions with any degree of accuracy (see Sect. 8.2.6 following). In situations such as these it is sensible to apply thresholds to the decision process in the manner depicted in Fig. 8.1b. Pixels which have probabilities for *all* classes below the threshold are not classified.

In practice, thresholds are applied to the discriminant functions and not the probability distributions, since the latter are never actually computed. With the incorporation of a threshold therefore, the decision rule of (8.5) becomes

$$\boldsymbol{x} \in \omega_i \quad \text{if} \quad g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \quad \text{for all} \quad j \neq i \tag{8.9a}$$

$$\text{and} \quad g_i(\boldsymbol{x}) > T_i \tag{8.9b}$$

where T_i is the threshold seen to be significant for spectral class ω_i. It is now necessary to consider how T_i can be estimated. From (8.7) and (8.9b) a classification is acceptable if

$$\ln\, p(\omega_i) - \tfrac{1}{2} \ln |\Sigma_i| - \tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{m}_i)^t \ \Sigma_i^{-1}(\boldsymbol{x} - \boldsymbol{m}_i) > T_i$$

i.e.

$$(\boldsymbol{x} - \boldsymbol{m}_i)^t \ \Sigma_i^{-1}(\boldsymbol{x} - \boldsymbol{m}_i) < -2T_i - \ln |\Sigma_i| + 2 \ln\, p(\omega_i) \tag{8.10}$$

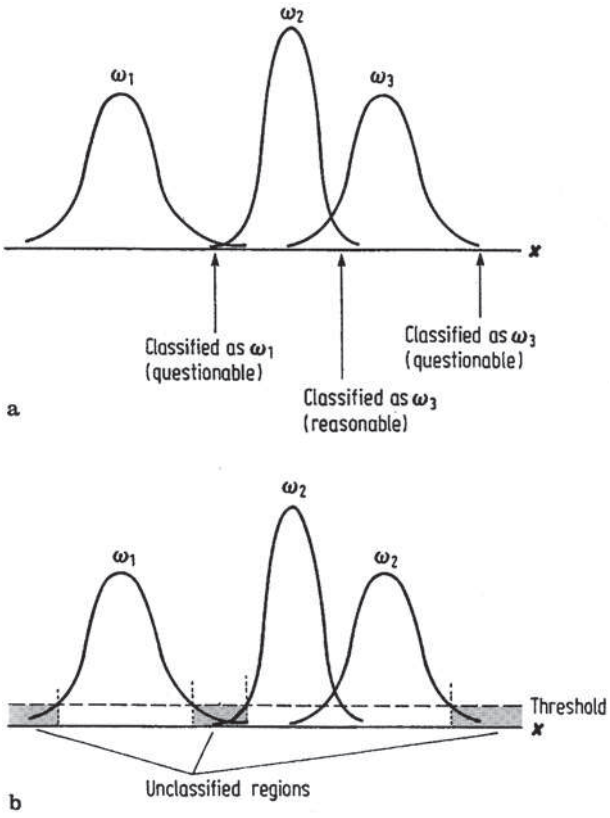


Fig. 8.1. a Illustration of poor classification for patterns lying near the tails of the distribution functions of all spectral classes; **b** Use of a threshold to remove poor classification

The left hand side of (8.10) has a χ^2 distribution with N degrees of freedom, if $\boldsymbol{x}$ is (assumed to be) distributed normally (Swain and Davis, 1978). N is the dimensionality of the multispectral space. As a result χ^2 tables can be consulted to determine that value of $(\boldsymbol{x} - \boldsymbol{m}_i)^t \ \Sigma_i^{-1}(\boldsymbol{x} - \boldsymbol{m}_i)$ below which a desired percentage of pixels will exist (noting that larger values of that quadratic form correspond to pixels lying further out in the tails of the normal probability distribution). This is depicted in Fig. 8.2.

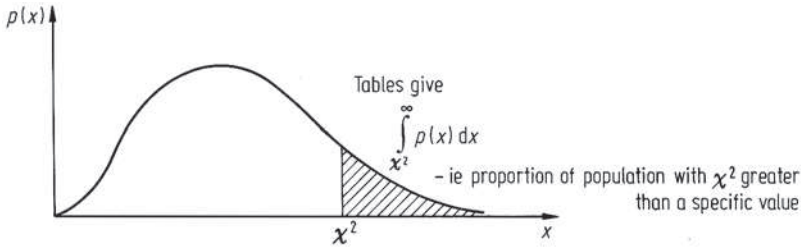


Fig. 8.2. Use of the χ^2 distribution for obtaining classifier thresholds

As an example of how this is used consider the need to choose a threshold such that 95% of all pixels in a class will be classified (i.e. such that the 5% least likely pixels for each spectral class will be rejected). χ^2 tables show that 95% of all pixels have χ^2 values (in Fig. 8.2) less than 9.488. Thus, from (8.10)

$$T_i = -4.744 - \frac{1}{2}\ln|\Sigma_i| + \ln\, p(\omega_i)$$

which thus can be calculated from a knowledge of the prior probability and covariance matrix of the ith spectral class.

8.2.6 Number of Training Pixels Required for Each Class

Sufficient training pixels for each spectral class must be available to allow reasonable estimates to be obtained of the elements of the class conditional mean vector and covariance matrix. For an N dimensional multispectral space the covariance matrix is symmetric of size $N \times N$. It has, therefore, $\frac{1}{2}N(N+1)$ distinct elements that need to be estimated from the training data. To avoid the matrix being singular at least $N(N+1)$ independent *samples* is needed. Fortunately, each N dimensional pixel vector in fact contains N samples (one in each waveband); thus the minimum number of independent training *pixels* required is $(N+1)$. Because of the difficulty in assuring independence of the pixels, usually many more than this minimum number is selected. Swain and Davis (1978) recommend as a practical minimum that $10N$ training pixels per spectral class be used, with as many as $100N$ per class if possible. For data with low dimensionality (say up to 5 or 6 bands) those numbers can usually be achieved, but for hyperspectral data sets finding enough training pixels per class is extremely difficult. Section 13.5 considers this problem in some detail.

8.2.7 A Simple Illustration

As an example of the use of maximum likelihood classification, the segment of Landsat multispectral scanner image shown in Fig. 8.3 is chosen. This is a 256×276 pixel array of image data in which four broad ground cover types are evident. These are water, fire burn, vegetation and "developed" land (urban). Suppose we want to produce a thematic map of these four cover types in order to enable the area and extent of the fire burn to be evaluated.

The first step is to choose training data. For such a broad classification, suitable sets of training pixels for each of the four classes are easily identified visually in the image data. Figure 8.3 also shows the locations of four training fields used for this purpose. Sometimes, to obtain a good estimate of class statistics it may be necessary to choose several training fields for the one cover type, located in different regions of the image.

The four-band signatures for each of the four classes, as obtained from the training fields, are given in Table 8.1. The mean vectors can be seen to agree generally

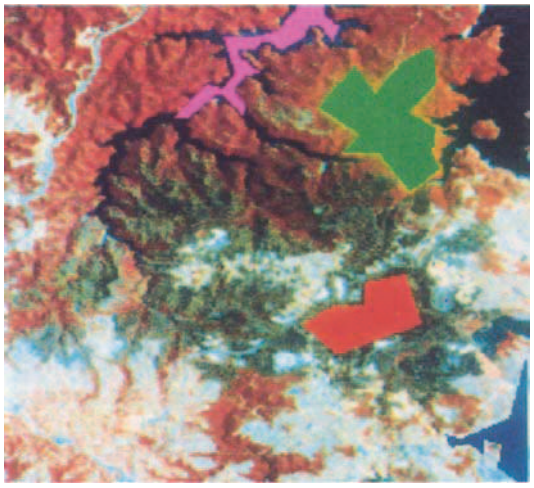


Fig. 8.3. Image segment to be classified, consisting of a mixture of natural vegetation, waterways, urban development and vegetation damaged by fire. Four training regions are identified in solid colour. These are water (violet), vegetation (green), fire burn (red) and urban (dark blue in the bottom right hand corner). Pixels from these were used to generate the signatures in Table 8.1

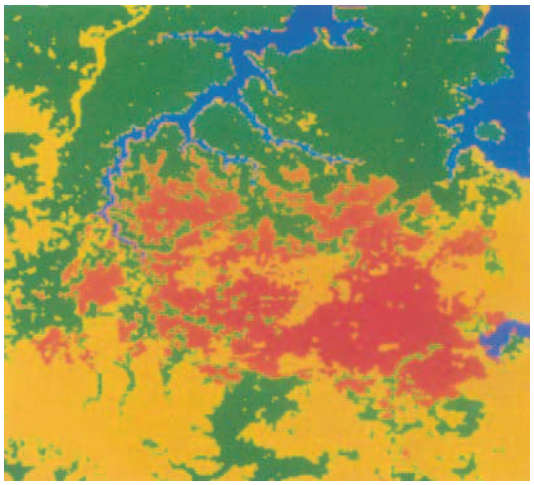


Fig. 8.4. Thematic map produced by maximum likelihood classification. Blue represents water, red is fire damaged vegetation, green is natural vegetation and yellow is urban development

with known spectral reflectance characteristics of the cover types. Also the class variances (diagonal elements in the covariance matrices) are small for water as might be expected but on the large side for the developed/urban class, indicative of its heterogeneous nature.

Using these signatures in a maximum likelihood algorithm to classify the four bands of the image in Fig. 8.3, the thematic map shown in Fig. 8.4 is obtained. The four classes, by area, are given in Table 8.2. Note that there are no unclassified pixels, since a threshold was not used in the labelling process. The area estimates are obtained by multiplying the number of pixels per class by the effective area of a pixel. In the case of the Landsat 2 multispectral scanner the pixel size was 0.4424 hectares.

Table 8.1. Class signatures generated from the training areas in Fig. 8.3. Numbers are on a scale of 0 to 255 (8 bit)

Class	Mean vector	Covariance matrix			
Water	44.27	14.36	9.55	4.49	1.19
	28.82	9.55	10.51	3.71	1.11
	22.77	4.49	3.71	6.95	4.05
	13.89	1.19	1.11	4.05	7.65
Fire burn	42.85	9.38	10.51	12.30	11.00
	35.02	10.51	20.29	22.10	20.62
	35.96	12.30	22.10	32.68	27.78
	29.04	11.00	20.62	27.78	30.23
Vegetation	40.46	5.56	3.91	2.04	1.43
	30.92	3.91	7.46	1.96	0.56
	57.50	2.04	1.96	19.75	19.71
	57.68	1.43	0.56	19.71	29.27
Developed (urban)	63.14	43.58	46.42	7.99	−14.86
	60.44	46.42	60.57	17.38	−9.09
	81.84	7.99	17.38	67.41	67.57
	72.25	−14.86	−9.09	67.57	94.27

Table 8.2. Tabular summary of the thematic map of Fig. 8.4

Class	No. of pixels	Area (ha)
Water	4830	2137
Fireburn	14182	6274
Vegetation	28853	12765
Developed (urban)	22791	10083

8.3 Minimum Distance Classification

8.3.1 The Case of Limited Training Data

The effectiveness of maximum likelihood classification depends upon reasonably accurate estimation of the mean vector $\boldsymbol{m}$ and the covariance matrix Σ for each spectral class. This in turn is dependent upon having a sufficient number of training pixels for each of those classes. In cases where this is not so, inaccurate estimates of the elements of Σ result, leading to poor classification. When the number of training samples per class is limited it can be more effective to resort to a classifier that does not make use of covariance information but instead depends only upon the mean positions of the spectral classes, noting that for a given number of samples these can be more accurately estimated than covariances. The so-called minimum distance classifier, or more precisely, minimum distance to class means classifier, is such an

approach. With this classifier, training data is used only to determine class means; classification is then performed by placing a pixel in the class of the nearest mean.

The minimum distance algorithm is also attractive since it is a faster technique than maximum likelihood classification, as will be seen in Sect. 8.5. However because it does not use covariance data it is not as flexible as the latter. In maximum likelihood classification each class is modelled by a multivariate normal class model that can account for spreads of data in particular spectral directions. Since covariance data is not used in the minimum distance technique class models are symmetric in the spectral domain. Elongated classes therefore will not be well modelled. Instead several spectral classes may need to be used with this algorithm where one might be suitable for maximum likelihood classification. This point is developed further in the case studies of Chap. 11.

8.3.2 The Discriminant Function

The discriminant function for the minimum distance classifier is developed as follows.

Suppose $\boldsymbol{m}_i, i = 1, \ldots M$ are the means of the M classes determined from training data, and $\boldsymbol{x}$ is the position of the pixel to be classified. Compute the set of squared Euclidean distances of the unknown pixel to each of the class means, defined in vector form as

$$\begin{aligned} \mathrm{d}(\boldsymbol{x}, \boldsymbol{m}_i)^2 &= (\boldsymbol{x} - \boldsymbol{m}_i)^t (\boldsymbol{x} - \boldsymbol{m}_i) \\ &= (\boldsymbol{x} - \boldsymbol{m}_i) \cdot (\boldsymbol{x} - \boldsymbol{m}_i), i = 1, \ldots M \end{aligned}$$

Expanding the product gives

$$\mathrm{d}(\boldsymbol{x}, \boldsymbol{m}_i)^2 = \boldsymbol{x} \cdot \boldsymbol{x} - 2\boldsymbol{m}_i \cdot \boldsymbol{x} + \boldsymbol{m}_i \cdot \boldsymbol{m}_i .$$

Classification is performed on the basis of

$$\boldsymbol{x} \in \omega_i \quad \text{if} \quad d(\boldsymbol{x}, \boldsymbol{m}_i)^2 < d(\boldsymbol{x}, \boldsymbol{m}_j)^2 \quad \text{for all} \quad j \neq i$$

Note that $\boldsymbol{x} \cdot \boldsymbol{x}$ is common to all $\mathrm{d}(\boldsymbol{x}, \boldsymbol{m}_j)^2$ and thus can be removed. Moreover, rather than classifying according to the smallest of the remaining expressions, the signs can be reversed and classification performed on the basis of

$$\boldsymbol{x} \in \omega_i \quad \text{if} \quad g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \quad \text{for all} \quad j \neq i \tag{8.11a}$$

where

$$g_i(\boldsymbol{x}) = 2\boldsymbol{m}_i \cdot \boldsymbol{x} - \boldsymbol{m}_i \cdot \boldsymbol{m}_i, \quad \text{etc.} \tag{8.11b}$$

Equation (8.11b) defines the discriminant function for the minimum distance classifier. In contrast to the maximum likelihood approach the decision surfaces for this classifier, separating the distinct spectral class regions in multispectral space, are linear, as seen in Sect. 8.3.4 following. The higher order decision surface possible with maximum likelihood classification renders it more powerful for partitioning multispectral space than the linear surfaces for the minimum distance approach.

Nevertheless, as noted earlier, minimum distance classification is of value when the number of training samples is limited and, in such a case, can lead to better accuracies than the maximum likelihood procedure.

Minimum distance classification can be performed also using distance measures other than Euclidean (Wacker and Landgrebe, 1972); notwithstanding this, algorithms based upon Euclidean distance definitions are those generally implemented in software packages for remote sensing image analysis, such as Multispec (http://dynamo.ecn.purdue.edu/∼biehl/MultiSpec), ENVI (http://www.rsinc.com) and ERMapper (http://www.ermapper.com).

8.3.3 Degeneration of Maximum Likelihood to Minimum Distance Classification

The major difference between the minimum distance and maximum likelihood classifiers lies in the use, by the latter, of the sample covariance information. Whereas the minimum distance classifier labels a pixel as belonging to a particular class on the basis only of its distance from the relevant mean, irrespective of its direction from that mean, the maximum likelihood classifier modulates its decision with direction, based upon the information in the covariance matrix. Furthermore the entry $-\frac{1}{2}\ln|\Sigma_i|$ in its discriminant function shows explicitly that patterns have to be closer to some means than others to have equivalent likelihoods of class membership. As a result substantially superior performance is expected of the maximum likelihood classifier, in general. The following situation however warrants consideration since then there is no advantage in maximum likelihood procedures. It could occur in practice when class covariance is dominated by systematic noise rather than by natural spectral spreads of the individual spectral classes.

Consider the covariance matrices of all classes to be diagonal and equal, and the variances in each component to be identical, so that

$$\Sigma_i = \sigma^2 I \quad \text{for all} \quad i.$$

Under these circumstances the discriminant function for the maximum likelihood classifier, from (8.7) becomes

$$g_i(\boldsymbol{x}) = \frac{1}{2}\ln \sigma^{2N} - \frac{1}{2\sigma^2}(\boldsymbol{x}-\boldsymbol{m}_i)^t(\boldsymbol{x}-\boldsymbol{m}_i) + \ln p(\omega_i)$$

The $\ln \sigma^{2N}$ term is now common to all classes and can be ignored, as can the $\boldsymbol{x}\cdot\boldsymbol{x}$ term that results from the scalar product, leaving

$$g_i(\boldsymbol{x}) = \frac{1}{2\sigma^2}\{2\boldsymbol{m}_i\cdot\boldsymbol{x} - \boldsymbol{m}_i\cdot\boldsymbol{m}_i\} + \ln p(\omega_i)$$

If the $\ln p(\omega_i)$ are ignored, on the basis of equal prior probabilities, then the $1/2\sigma^2$ factor can be removed giving

$$g_i(\boldsymbol{x}) = 2\boldsymbol{m}_i\cdot\boldsymbol{x} - \boldsymbol{m}_i\cdot\boldsymbol{m}_i$$

which is the discriminant function for the minimum distance classifier. Thus minimum distance and maximum likelihood classification are equivalent for identical and symmetric spectral class distributions.

8.3.4 Decision Surfaces

The implicit surfaces in multispectral space separating adjacent classes are defined by the respective discriminant functions being equal. Thus the surface between the ith and jth spectral classes is given by

$$g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = 0$$

Substituting from (8.11b) gives

$$2(\boldsymbol{m}_i - \boldsymbol{m}_j) \cdot \boldsymbol{x} - (\boldsymbol{m}_i \cdot \boldsymbol{m}_i - \boldsymbol{m}_j \cdot \boldsymbol{m}_j) = 0$$

This defines a linear surface – often called a hyperplane in more than three dimensions. In contrast therefore to maximum likelihood classification in which the decision surfaces are quadratic and therefore more flexible, the decision surfaces for minimum distance classification are linear and more restricted.

8.3.5 Thresholds

Thresholds can be applied to minimum distance classification by ensuring that not only is a pixel closest to a candidate class but also that it is within a prescribed distance of that class. Such a technique is used regularly. Often the distance threshold is specified according to a number of standard deviations from a class mean.

8.4 Parallelepiped Classification

The parallelepiped classifier is a very simple supervised classifier that is, in principle, trained by inspecting histograms of the individual spectral components of the available training data. Suppose, for example, that the histograms of one particular spectral class for two dimensional data are as shown in Fig. 8.5. Then the upper and lower significant bounds on the histograms are identified and used to describe the brightness value range for each band for that class. Together, the range in all bands describes a multidimensional box or parallelepiped. If, on classification, pixels are found to lie in such a parallelepiped they are labelled as belonging to that class. A two-dimensional pattern space might therefore be segmented as shown in Fig. 8.6.

While the parallelepiped method is, in principle, a particularly simple classifier to train and use, it has several drawbacks. One is that there can be considerable gaps between the parallelepipeds; pixels in those regions will not be classified. By

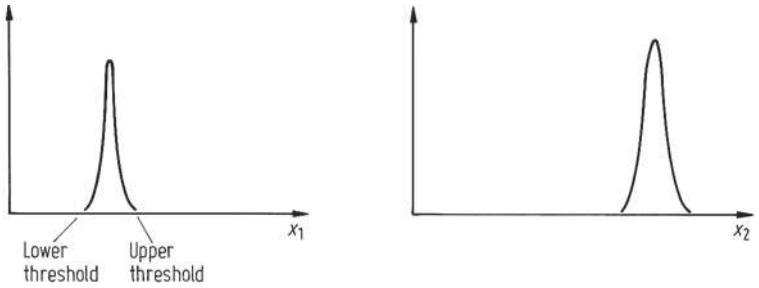


Fig. 8.5. Histograms for the components of a two-dimensional set of training data corresponding to a single spectral class. The upper and lower bounds are identified as the edges of a two-dimensional parallelepiped

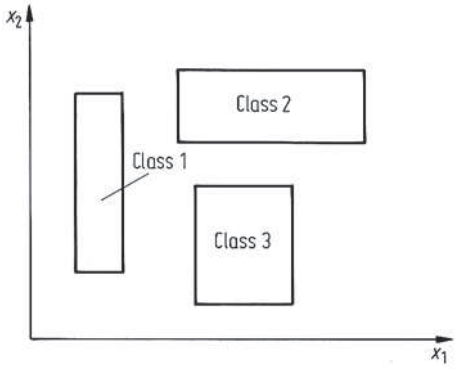


Fig. 8.6. An example of a set of two-dimensional parallelepipeds

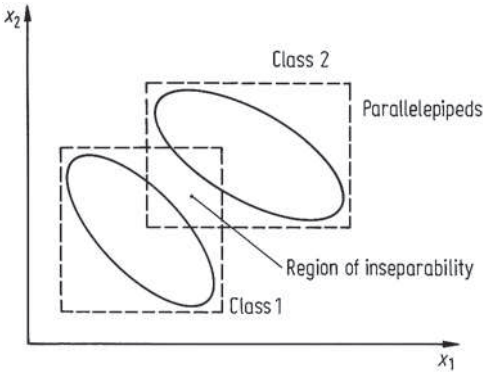


Fig. 8.7. Parallelepiped classification of correlated data showing regions of inseparability

comparison the minimum distance and maximum likelihood classifiers will label all pixels in an image, unless thresholding methods are used. Another limitation is that prior probabilities of class membership are not taken account of; nor are they for minimum distance classification. Finally, for correlated data there can be overlap of the parallelepipeds since their sides are parallel to the spectral axes. Consequently there is some data that cannot be separated, as illustrated in Fig. 8.7.

8.5 Classification Time Comparison of the Classifiers

Of the three classifiers commonly used with remote sensing image data the parallelepiped procedure is the fastest in classification since only comparisons of the spectral components of a pixel with the spectral dimensions of the parallelepipeds are required.

For the minimum distance classifier the discriminant function in (8.11b) requires evaluation for each pixel. In practice $2\boldsymbol{m}_i$ and $\boldsymbol{m}_i \cdot \boldsymbol{m}_i$ would be calculated beforehand, leaving N multiplications and N additions to check the potential membership of a pixel to one class, where N is the number of components in $\boldsymbol{x}$.

By comparison, evaluation of the discriminant function for maximum likelihood classification in (8.7) requires $N^2 + N$ multiplications and $N^2 + 2N + 1$ additions, to check one pixel against one class, given that

$$-\frac{1}{2} \ln |\Sigma_i| + \ln\, p(\omega_i)$$

would have been calculated beforehand. Ignoring additions by comparison to multiplications, the maximum likelihood classifier takes $N + 1$ times as long as the minimum distance classifier to perform a classification. It is also significant to note that classification time, and thus cost, increases quadratically with number of spectral components for the maximum likelihood classifier but only linearly for minimum distance and parallelepiped classification. This has particular relevance to feature reduction (Chap. 10).

8.6 Other Supervised Approaches

8.6.1 The Mahalanobis Classifier

Consider the discriminant function for the maximum likelihood classifier, for the special case of equal prior probabilities, as defined in (8.8). If the sign of this function is reversed it can be considered as a distance squared measure since the quadratic entry has those dimensions and the other term is a constant. Thus we can define

$$\mathrm{d}(\boldsymbol{x}, \boldsymbol{m}_i)^2 = \ln |\Sigma_i| + (\boldsymbol{x} - \boldsymbol{m}_i)^t \Sigma_i^{-1} (\boldsymbol{x} - \boldsymbol{m}_i) \tag{8.12}$$

and classify on the basis of the smallest $\mathrm{d}(\boldsymbol{x}, \boldsymbol{m}_i)$ as for the Euclidean minimum distance classifier. Thus the maximum likelihood classifier can be regarded as a minimum distance-like classifier but with a distance measure that is direction sensitive and modified according to class.

Consider the case now where all class covariances are equal – i.e. $\Sigma_i = \Sigma$ for all i. Clearly the $\ln |\Sigma_i|$ term is now not discriminating and can be ignored. The

distance measure then reduces to

$$\mathrm{d}(\boldsymbol{x}, \boldsymbol{m}_i)^2 = (\boldsymbol{x} - \boldsymbol{m}_i)^t \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{m}_i) \tag{8.13}$$

Such a classifier is referred to as a *Mahalanobis distance* classifier, although sometimes the term is applied to the more general measure of (8.12). Mahalanobis distance is understood as the square root of (8.13). Under the additional constraint that $\Sigma = \sigma^2 I$ the Mahalanobis classifier reduces, as before, to the minimum Euclidean distance classifier.

The advantage of the Mahalanobis classifier over the maximum likelihood procedure is that it is faster and yet retains a degree of direction sensitivity via the covariance matrix Σ, which could be a class average or a pooled variance.

8.6.2 Table Look Up Classification

Since the set of discrete brightness values that can be taken by a pixel in each spectral band is limited, there is a finite, although large, number of pixel vectors in any particular image. For a given class in that image the number of distinct pixel vectors may not be very extensive. Consequently a viable classification scheme is to note the set of pixel vectors corresponding to a given class, using representative training data, and then use those to classify the image by comparing unknown image pixels with each pixel in the training data until a match is found. No arithmetic operations are required and, notwithstanding the number of comparisons that might be necessary to determine a match, it is a fast classifier. It is referred to as a look up table approach since the pixel brightnesses are stored in tables that point to the corresponding classes.

An obvious drawback with this approach is that the chosen training data must contain one of every possible pixel vector for each class. Should some be missed then the corresponding pixels in the image will be left unclassified. This is in contrast to the procedures treated above.

8.6.3 The kNN (Nearest Neighbour) Classifier

A classifier that is particularly simple in concept, but can be time consuming to apply, is the k-Nearest Neighbour classifier. It assumes that pixels close to each other in feature space are likely to belong to the same class. In its simplest form an unknown pixel is labelled by examining the available training pixels in multispectral space and choosing the class most represented among a pre-specified number of nearest neighbours. The comparison essentially requires the distances from the unknown pixel to all training pixels to be computed.

Suppose there are k_i neighbours labelled as class ω_i out of k nearest neighbours for a pixel vector $\boldsymbol{x}$, noting that $\sum_{i=1}^{M} k_i = k$ where M is the total number of classes.

In the basic *kNN* rule we define the discriminant function for the ith class as

$$g_i(\boldsymbol{x}) = k_i$$

and the decision rule is:

$$\boldsymbol{x} \in \omega_i, \qquad \textit{if} \qquad g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \qquad \text{for all} \qquad j \neq i$$

The basic rule does not take the distance of each neighbour to the current pixel vector into account and may lead to tied results. An improvement is to distance-weight the discriminant function:

$$g_i(\boldsymbol{x}) = \frac{\sum_{j=1}^{k_i} 1/\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i^j)}{\sum_{i=1}^{M} \sum_{j=1}^{k_i} 1/\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i^j)}$$

where $\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i^j)$ is the spectral distance (commonly Euclidean) between the unknown pixel vector $\boldsymbol{x}$ and its neighbour $\boldsymbol{x}_i^j$, the jth of the k_i pixels in class ω_i.

If the training data for each class is not in proportion to its respective population, $p(\omega_i)$, in the image, a Bayesian Nearest-Neighbour rule can be used:

$$g_i(\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\,\omega_i)\,p(\omega_i)}{\sum_{j=1}^{M} p(\boldsymbol{x}|\,\omega_j)\,p(\omega_j)} = \frac{k_i\,p(\omega_i)}{\sum_{j=1}^{M} k_j\,p(\omega_j)}$$

In the *kNN* algorithm as many spectral distances as there are training pixels must be evaluated for each training pixel to be labelled; that requires an impractically high computational load, especially when the number of spectral bands and/or the number of training samples is large. The method is not well-suited therefore to hyperspectral data sets, although it is possible to improve the efficiency of the distance search process (Dasarathy, 1991).

8.7 Gaussian Mixture Models

In Sect. 3.5 the prospect of information classes being composed of sets of spectral classes was raised. The material in Sect. 8.2, however, has been derived on the basis that an information class is composed of a single spectral class that can be represented by a multidimensional normal distribution. In practice that is rarely the case: in order to represent the data effectively more than one normally distributed spectral class is required to model properly the distribution of pixel vectors in a given information class.

One of the challenges to successful image classification is to find an acceptable set of spectral classes for each information class. In Sect. 9.1 it is suggested that clustering algorithms can be used for that purpose, and indeed they can be applied

very successfully to that end. Another, more theoretically appealing approach is to try to learn the mixture of spectral classes for each information class from the available training data, as in the following.

If we assume that a given information class is composed of a number of unimodal normally distributed spectral classes, then it is natural to attempt to devise a (information) class model of the form

$$f(\boldsymbol{x}) = \sum_{c=1}^{C} \alpha_c p(\boldsymbol{x}|\boldsymbol{m}_c, \Sigma_c)$$

where $\boldsymbol{m}_c$ and Σ_c are the mean vector and covariance matrix of the cth spectral class conditional normal distribution; the α_c are weighting parameters (which sum to unity) such that the mixture model expressed by $f(\boldsymbol{x})$ fits the available training data. The total number of spectral class components is C.

We have to estimate the set of parameters $\{C, \alpha_c, m_c, \Sigma_c\}$ from the training data, and that is a considerable challenge in practice. Kuo and Landgrebe (2002) show how this can be achieved.

8.8 Context Classification

8.8.1 The Concept of Spatial Context

The classifiers treated so far are often referred to as point or pixel-specific classifiers in that they label a pixel on the basis of its spectral properties alone, with no account taken of how any neighbouring pixels are labelled. Yet, in any real image, adjacent pixels are related or correlated, both because imaging sensors acquire significant portions of energy from adjacent pixels[1] and because ground cover types generally occur over a region that is large compared with the size of a pixel. In an agricultural area, for example, if a particular image pixel represents wheat it is highly likely that its neighbouring pixels will also be wheat. This knowledge of neighbourhood relationships is a rich source of information that is not exploited in simple, traditional classifiers. In this section we consider the importance of spatial context and see the benefit of taking it into account when making classification decisions. Not only is the inclusion of context important because it exploits spatial information, as such, but, in addition, sensitivity to the correct context for a pixel can improve a thematic map by helping to remove individual pixel labelling errors that might result from noisy data, or unusual classifier performance (see Problem 8.6).

Classification methods that take into account the labelling of neighbours when seeking to determine the most appropriate class for a pixel are said to be context sensitive, or simply context classifiers. They attempt to develop a thematic map that is consistent both spectrally and spatially.

[1] This is referred to as the point spread function effect, which is discussed in Forster (1982).

The degree to which adjacent pixels are strongly correlated will depend on the spatial resolution of the sensor and the scale of natural and cultural regions on the earth's surface. Adjacent pixels over an agricultural region will be strongly correlated, whereas for the same sensor, adjacent pixels over a busier, urban region would not show strong correlation. Likewise, for a given area, neighbouring Landsat MSS pixels, being larger, may not demonstrate as much correlation as adjacent SPOT HRV pixels. In general terms, context classification techniques usually warrant consideration when processing higher resolution imagery.

8.8.2 Context Classification by Image Pre-processing

Perhaps the simplest method for exploiting spatial context is to process the image data before classification in order to modify or enhance its spatial properties. A median filter (Sect. 5.5.2), for example, will help in reducing salt and pepper noise that would lead to inconsistent class labels. Moreover, the application of simple averaging filters (possibly with edge preserving thresholds) can be used to impose a degree of homogeneity among the brightness values of adjacent pixels thereby increasing the chance that neighbouring pixels may be given the same label.

An alternative is to generate a separate channel of data that associates spatial properties with pixels. For example, a texture channel could be added and classification carried out (using a suitable algorithm such as the minimum distance rule) on the combined multispectral and texture channels. Along this line, Gong and Howarth (1990) have set up a "structural information" channel to bias a classification according to the density of high spatial frequency data in order to improve the classification of image data containing urban segments. The reasoning behind the approach is that urban regions are characterised by high spatial frequency detail whereas, conversely, the high frequency detail present in non-urban regions is low. The additional channel reflects this understanding and accordingly influences the classification which would otherwise be carried out on the basis of spectral data alone.

One of the more useful spatial pre-processing techniques is that used in the ECHO classification methodology. In ECHO (Extraction and Classification of Homogeneous Objects) regions of similar spectral properties are "grown" before classification is performed. Several region growing techniques are available, possibility the simplest of which is to aggregate pixels into small regions by comparing their brightnesses in each channel and then aggregate the small regions into bigger regions in a similar manner. When this is done, ECHO classifies the regions as single objects and only resorts to standard maximum likelihood classification when it has to treat individual pixels that could not be put into regions. Details of ECHO will be found in Kettig and Landgrebe (1976); it is also available in the Multispec image analysis software (http://dynamo.ecn.purdue/∼biehl/Multispec/).

8.8.3 Post Classification Filtering

Once a thematic map has been generated using a simple point classifier some degree of spatial context can be developed by logically filtering the map. For example, if the map is examined in 3×3 windows, a label at the centre of the window might be changed to the label most represented in the window. Clearly this must be done carefully, with the user having some control over the minimum size region of a given cover type that is acceptable in the filtered image product (Harris, 1985). Post classification filtering by this approach has been treated by Townsend (1986).

8.8.4 Probabilistic Label Relaxation

Spatial consistency in a classified image product can also be developed using the process of label relaxation. While it has little theoretical foundation, and is more complex than the methods outlined in the previous sections, it does allow the spatial properties of a region to be carried into the classification process in a logically consistent way.

8.8.4.1 The Basic Algorithm

The process commences by assuming that a classification, based on spectral data alone, has already been carried out. There is available therefore, for each pixel, a set of probabilities that describe the chance that the pixel belongs to each of the possible ground cover classes under consideration. This set of probabilities could be computed from (8.6) and (8.7) if maximum likelihood classification had been used first. If another classification method had been employed, then some other assignment process will be required. It could even be as simple as allocating a high probability to the most favoured class label and lower probabilities to the rest. Let the set of probabilities for a pixel (m) currently of interest be represented by

$$p_m(\omega_i) \qquad i = 1, \ldots K \tag{8.14}$$

where K is the total number of classes; $p_m(\omega_i)$ should be read as "the probability that ω_i is the correct class for pixel m." Note that the full set of $p_m(\omega_i)$ must sum to unity for a given pixel – viz.

$$\sum_i p_m(\omega_i) = 1.$$

Suppose now that a neighbourhood is defined surrounding pixel m. This can be of any size and, in principle, should be large enough to ensure that all the pixels considered to have any spatial correlation with m are included. For high resolution imagery this is not practical and simple neighbourhoods such as that shown in Fig. 8.8 are often adopted.

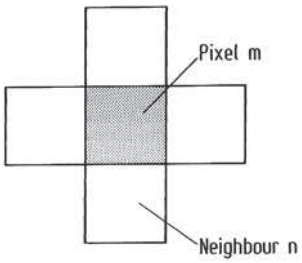


Fig. 8.8. Definition of a simple neighbourhood about pixel m

Now assume that a *neighbourhood function* $Q_m(\omega_i)$ can be found (by means to be described below) which allows the pixels in the prescribed neighbourhood to influence the possible classification of pixel m. This influence is exerted by multiplying the label probabilities in (8.14) by the $Q_m(\omega_i)$. However, so that the new set of label probabilities sum to one, these new values are divided by their sum:

$$p'_m(\omega_i) = \frac{p_m(\omega_i)Q_m(\omega_i)}{\sum_i p_m(\omega_i)Q_m(\omega_i)} \tag{8.15}$$

Such a modification is made to the set of label probabilities for all pixels by moving over the image from its top left hand to bottom right hand corners. In the following it will be seen that the neighbourhood function $Q_m(\omega_i)$ depends on the label probabilities of the neighbouring pixels, so that if all the pixel probabilities are modified in the manner just described then the neighbours for any given pixel have also been altered. Consequently, (8.15) should be applied again to give newer estimates still of the label probabilities. Indeed, (8.15) is applied as many times as necessary to ensure that the $p'_m(\omega_i)$ have stabilised – i.e. that they do not change with further iteration. It is assumed that the $p'_m(\omega_i)$ then represent the correct set of label probabilities for the pixel, having taken account both of spectral data (in the initial determination of label probabilities) and spatial context (via the neighbourhood functions). Since the process is iterative, (8.15) is usually written as an explicit iteration formula:

$$p_m^{k+1}(\omega_i) = \frac{p_m^k(\omega_i)Q_m^k(\omega_i)}{\sum_i p_m^k(\omega_i)Q_m^k(\omega_i)} \tag{8.16}$$

where k is the iteration counter. Depending on the size of the image and its spatial complexity, the number of iterations required to stabilise the label probabilities may be quite large. However, most change in the label probabilities occurs in the first few iterations and there is good reason to believe that proceeding beyond say 5 to 10 iterations may not be necessary in most cases (see Sect. 8.8.4.4).

8.8.4.2 The Neighbourhood Function

Consider just one of the neighbours of pixel m in Fig. 8.8 – call it pixel n. Suppose there is available a measure of compatibility of the current labelling of pixel m and

its neighbouring pixel n. For example let $r_{mn}(\omega_i, \omega_j)$ describe numerically how compatible it is to have pixel m classified as ω_i and neighbouring pixel n classified as ω_j. It would be expected, for example, that this measure will be high if the adjoining pixels are both labelled wheat in an agricultural region, but low if one of the neighbours was classified as snow. There are several ways these *compatibility coefficients*, as they are called, can be defined. An intuitively appealing definition is based on conditional probabilities. Thus, the compatibility measure $p_{mn}(\omega_i|\omega_j)$ is the probability that ω_i is the correct label for pixel m if ω_j is the correct label on pixel n. A small piece of evidence in favour of ω_i being correct for pixel m is $p_{mn}(\omega_i|\omega_j)p_n(\omega_j)$ – i.e. the probability that ω_i is correct for pixel m if ω_j is correct for pixel n multiplied by the probability that ω_j is correct for pixel n[2]. Since probabilities for all possible labels on pixel n are available (even though some might be very small) the total evidence from pixel n in favour of ω_i being the correct class for pixel m will be the sum of the contributions from all pixel n's labelling possibilities, viz.

$$\sum_j p_{mn}(\omega_i|\omega_j)p_n(\omega_j).$$

Consider now the full neighbourhood of the pixel m. In a like manner all the neighbours contribute evidence in favour of labelling pixel m as coming from class ω_i. All these contributions are simply added[3], via the use of *neighbour weights* d_n that recognise that some neighbours may be more influential than others (as for example, pixels along a scan line in MSS data compared with those running down an image, owing to the oversampling that occurs along rows – see Fig. A.2). Thus, at the kth iteration, the total neighbourhood support for pixel m being classified as ω_i is:

$$Q_m^k(\omega_i) = \sum_n d_n \sum_j p_{mn}(\omega_i|\omega_j)p_n^k(\omega_j) \tag{8.17}$$

This is the definition of the neighbourhood function. In (8.16) and (8.17) it is common to include pixel m in its own neighbourhood so that the modification process is not entirely dominated by the neigbours, particularly if the number of iterations is so large as to take the process quite a long way from its starting point.

Unless there is good reason to do otherwise the neighbour weights are generally chosen all to be the same.

8.8.4.3 Determining the Compatibility Coefficients

Several methods are possible for determining values for the compatibility coefficients $p_{mn}(\omega_i|\omega_j)$. One is to have available a spatial model for the region under consideration, derived from some other data source. In an agricultural region, for example,

[2] This is the probability of the joint event that pixel m is labelled ω_i and pixel n is labelled ω_j.

[3] An alternative way of handling the full neighbourhood is to take the geometric mean of the neighbourhood contributions.

some general idea of field sizes along with a knowledge of the pixel size of the sensor being used should make it possible to estimate how often one particular class occurs following a given class on an adjacent pixel. Another approach is to compute values for the compatibility coefficients from ground truth pixels, although the ground truth needs to be in the form of training regions that contain heterogeneous and spatially representative cover types.

8.8.4.4 The Final Step – Stopping the Process

While the relaxation process operates on label probabilities, the user is interested in the actual labels themselves. At the completion of relaxation, or at any intervening stage, each of the pixels can be classified according to the highest label probability. Thought has to be given as to how and when the iterations should be terminated. As suggested earlier, the process can be allowed to go to a natural completion at which further iteration leads to no changes in the label probabilities for all pixels. This however presents two difficulties. First, up to several hundred iterations may be involved leading to a costly post classification step. Secondly, it is observed in practice that the relaxation process improves the classification results in the first few iterations, by the embedding of spatial information, often to deteriorate later in the process (Richards, Landgrebe and Swain, 1981). Indeed, if the process is not terminated, the thematic map, after a large number of iterations of relaxation, can be worse than before the technique was applied.

To avoid these difficulties, a stopping rule or other controlling mechanism is needed. As seen in the example of the following section, stopping after just a few iterations may allow most of the benefit to be drawn from the process. Alternatively, the labelling errors remaining at each iteration can be checked against ground truth, if available, and the iterations terminated when the labelling error is seen to be minimised (Gong and Howarth, 1989).

Another approach is to control the propagation of contextual information as iteration proceeds (Lee, 1984). A little thought will reveal that, in the first iteration, only the immediate neighbours of a pixel have an influence on its labelling. In the second iteration the neighbours two away will now have an influence via the intermediary of the intervening pixels. Similarly, as iterations proceed, information from neighbours further away is propagated into the pixel of interest to modify its label probabilities. If the user has a view of the separation between neighbours at which the spatial correlation has dropped to negligible levels, then the appropriate number of iterations should be able to be identified at which to terminate the process without unduly sacrificing any further improvement in labelling accuracy. Noting also that the nearest neighbours should be most influential, with those further out being less important, a useful variation is to reduce the values of the neighbour weights d_n as iteration proceeds so that after say 5 to 10 iterations they have been brought to zero. Further iterations will then have no effect, and degradation in labelling accuracy cannot occur (Lee and Richards, 1989).

8.8.4.5 Examples

Figure 8.9 illustrates a simple application of relaxation labelling, in which a hypothetical image of 100 pixels has been classified into just two classes – grey and white. The ground truth for the region is shown, along with the thematic map (initial labelling) assumed to have been generated from a point classifier such as the maximum likelihood rule. Also shown are the compatibility coefficients, expressed as conditional probabilities, computed from the ground truth map. Label probabilities were assumed to be 0.9 for the favoured label in the initial labelling and 0.1 for the less likely label. The initial labelling, by comparison with the ground truth, can be seen to have an accuracy of 82% (there are 12 pixels in error). The labelling (selected on the basis of the largest current label probability) at significant stages during iteration is shown, illustrating the reduction in classification error resulting from the incorporation of spatial information into the process. After 15 interations all initial labelling errors have been removed, leading to a thematic map 100% in agreement with the ground truth. In this case the relaxation process was allowed to proceed to completion and there have been no ill effects. As pointed out in the previous section, however, this is the exception and stopping rules may have to be applied in most cases. Other simple examples where this is the case will be found in Richards et al., (1981).

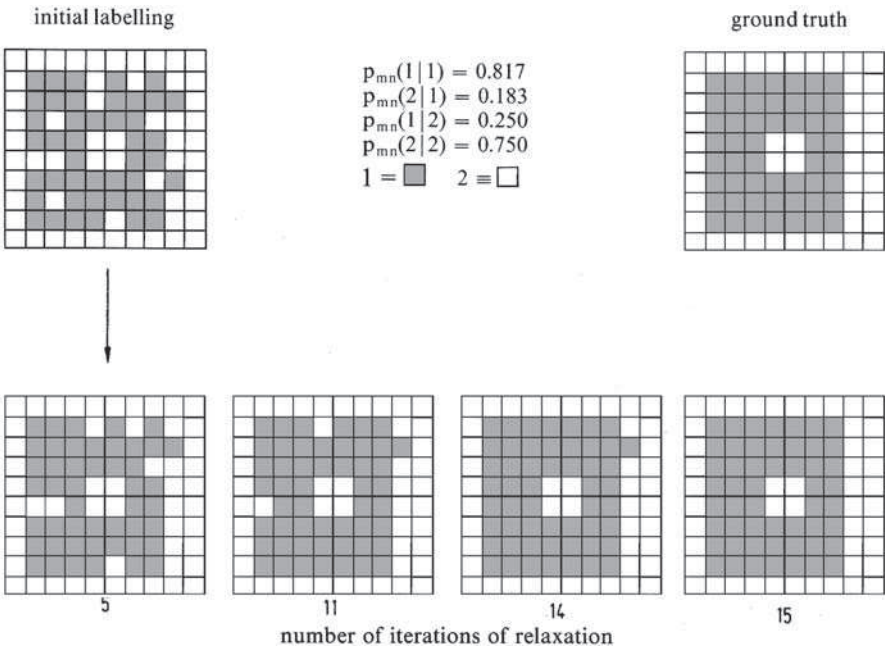


Fig. 8.9. Simple demonstration of pixel relaxation labelling

As a second example, the leftmost 82×100 pixels of the agricultural image shown in Fig. 3.1 have been chosen. Figure 8.10a shows the ground truth for the image segment and Fig. 8.10b shows the result of a maximum likelihood classification. The initial classification accuracy is 65.6%. The relaxation process was initialised using actual probability estimates from the maximum likelihood rule. Conditional probabilities as such were not used as compatibility coefficients. Instead, a slightly different set of compatibilities as proposed by Peleg and Rosenfeld (1980) was adopted. Also, to control the propagation of context information and thereby obviate any deleterious effect of allowing the relaxation process to proceed unconstrained, the neighbourhood weights were diminished with iteration count as described in previous section. Figure 8.10c shows the final labelling, which has an accuracy of 72.4%. Full details of this example are available in Lee and Richards (1989).

8.8.5 Handling Spatial Context by Markov Random Fields

The effect of spatial context can also be incorporated into a classification using the concept of the Markov Random Field (MRF). It is useful in developing the Markov Random Field approach to commence by considering the whole image, rather than just a local neighbourhood. We will restrict our attention to a neighbourhood once we have established some fundamental concepts.

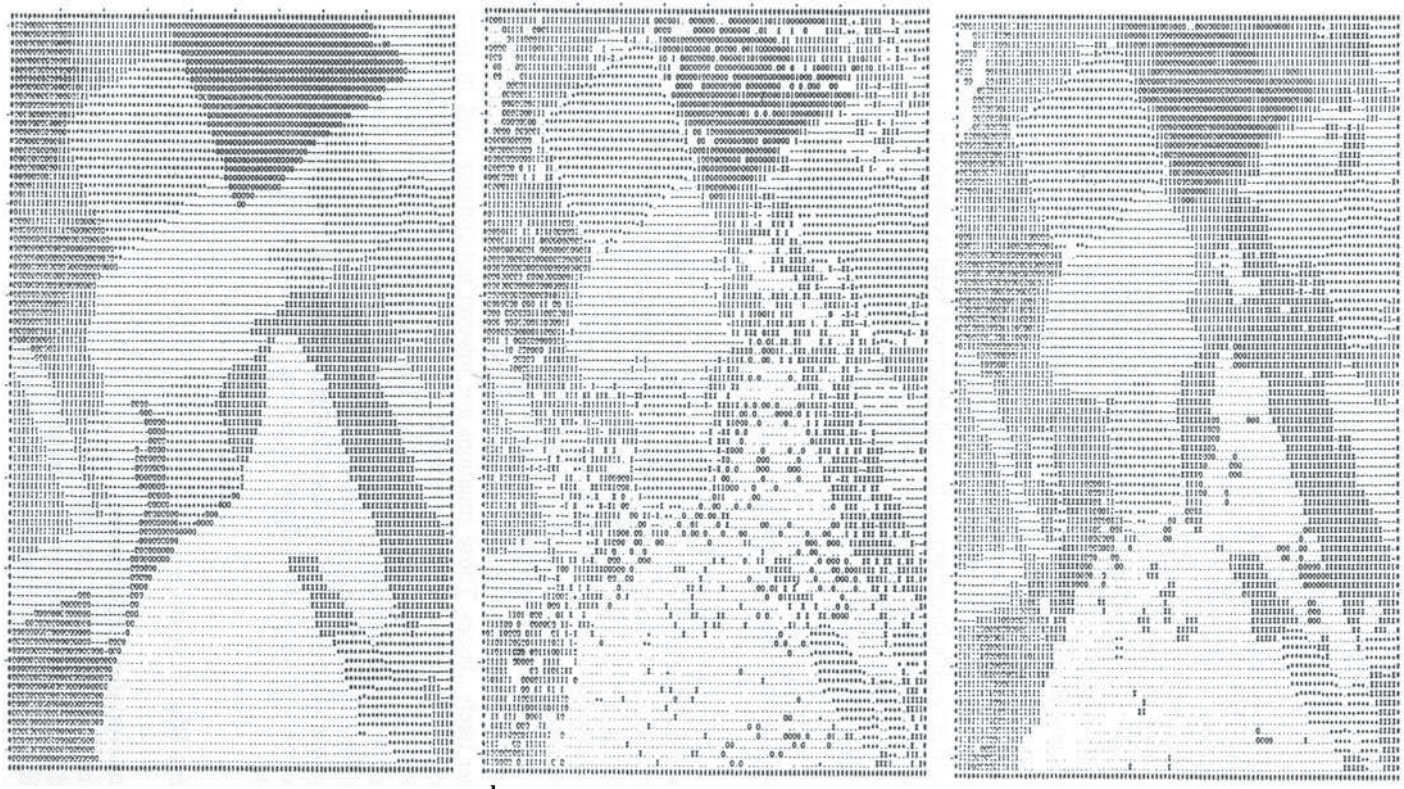
Suppose there is a total of M pixels in the image to be classified, with measurement vectors $\boldsymbol{x}_1, \ldots \boldsymbol{x}_M$. Alternatively, the measurement vectors can be expressed $\{\boldsymbol{x}_m : m = 1, \ldots M\}$, in which $m \equiv (i, j)$ in our usual way of indexing the pixels in an image. We can describe the full set of measurement vectors by $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots \boldsymbol{x}_M\}$. Further, suppose the class labels on each of the M pixels can be represented by the set $\Omega = \{\omega_{c1}, \ldots \omega_{cM}\}$; we could refer to that as the *scene labelling,* because it looks at the classification of every pixel in the scene. Each ω_{cm} can be one of $c = 1, \ldots C$ available classes. By classification what we want to find, of course, is the scene labelling (or our best estimate) that matches the ground truth – i.e. the actual classes of the pixels on the earth's surface. Let the actual labels on the ground be represented by Ω^*.

There will be a probability distribution $p(\Omega)$ associated with the labelling Ω of the whole scene which describes the likelihood of finding that distribution of labels over the image. Ω is sometimes referred to as a *random field.*

In principle, what we would like to do is find the scene labelling $\hat{\Omega}$ – that is the classification of all pixels – that maximises the global posterior probability $p(\Omega|\boldsymbol{X})$, the probability that Ω is the correct overall scene labelling given that the full set of measurement vectors for the scene is $\boldsymbol{X}$. By using Bayes' theorem we can express this as

$$\hat{\Omega} = \arg\max_{\Omega} \{p(\boldsymbol{X}|\Omega)\, p(\Omega)\} \tag{8.18}$$

in which the argmax function says that we choose the value of Ω that maximises its argument. The distribution $p(\Omega)$ is the prior probability of the scene labelling.



a **b** **c**

Fig. 8.10. a Ground truth for the left-hand side of the image in Fig. 3.1. The symbols are: · = red soil, ∗ = cotton crop, 0 = bare soil (low moisture), I = dry bare soil, + = early vegetation growth, X = mixed bare soil, − = bare soil (moist or ploughed). **b** Result of a maximum likelihood classification of Landsat MSS data. **c** Result of applying relaxation labelling to the result in **b**, incorporating a reduction in the neighbour weights with iteration

What we need to do now essentially is to perform the maximisation in (8.18), recognising however that the pixels are contextually dependent ie. there is some spatial correlation among them because adjacent pixels are likely to come from the same class. To render the problem tractable we consider the posterior probability just at the individual pixel level, so that our objective, for pixel m, is to find the class c that maximises $p(\omega_{cm}|\boldsymbol{x}_m,\ \omega_{\partial m})$ where $\omega_{\partial m}$ is the labelling on the pixels in a neighbourhood about pixel m. A possible neighbourhood is that shown in Fig.8.8, although often the immediately diagonal neighbours about m can also be included. Now we note

$$\begin{aligned} p(\omega_{cm}|\boldsymbol{x}_m, \omega_{\partial m}) &= p(\boldsymbol{x}_m, \omega_{\partial m}, \omega_{cm})/p(\boldsymbol{x}_m, \omega_{\partial m}) \\ &= p(\boldsymbol{x}_m|\omega_{\partial m}, \omega_{cm})p(\omega_{\partial m}, \omega_{cm})/p(\boldsymbol{x}_m, \omega_{\partial m}) \\ &= p(\boldsymbol{x}_m|\omega_{\partial m}, \omega_{cm})p(\omega_{cm}|\omega_{\partial m})p(\omega_{\partial m})/p(\boldsymbol{x}_m, \omega_{\partial m}) \end{aligned}$$

The first term on the right hand side is similar to the class conditional distribution function, but conditional also on the neighbourhood labelling. It is reasonable to assume that the class conditional density is independent of the neighbourhood labelling so that $p(\boldsymbol{x}_m|\omega_{\partial m}, \omega_{cm}) = p(\boldsymbol{x}_m|\omega_{cm})$. Note also that the measurement vector $\boldsymbol{x}_m$ and the neighbourhood labelling are independent of each other so that $p(\boldsymbol{x}_m, \omega_{\partial m}) = p(\boldsymbol{x}_m)p(\omega_{\partial m})$, so that the last expression becomes

$$\begin{aligned} p(\omega_{cm}|\boldsymbol{x}_m, \omega_{\partial m}) &= p(\boldsymbol{x}_m|\omega_{cm})p(\omega_{cm}|\omega_{\partial m})p(\omega_{\partial m})/p(\boldsymbol{x}_m)p(\omega_{\partial m}) \\ &= p(\boldsymbol{x}_m|\omega_{cm})p(\omega_{cm}|\omega_{\partial m})/p(\boldsymbol{x}_m) \end{aligned}$$

Since $1/p(\boldsymbol{x}_m)$ does not contribute to the decision concerning the correct label for pixel m it can be removed from the last expression, leaving

$$p(\omega_{cm}|\boldsymbol{x}_m, \omega_{\partial m}) \propto p(\boldsymbol{x}_m|\omega_{cm})p(\omega_{cm}|\omega_{\partial m}) \tag{8.19}$$

Now consider the probability $p(\omega_{cm}|\omega_{\partial m})$. Essentially it is the probability that the correct class for pixel m is c given the classes currently on the neighbours of pixel m. In many ways it is analogous to the neighbourhood function for probabilistic relaxation in (8.17). It is also a conditional prior probability – i.e. a prior probability for the class on pixel m conditional on its neighbourhood. Because of this conditionality, the random fields of labels we are considering are now referred to as *Markov* Random Fields (*MRF*).

The question is how do we now find a value for $p(\omega_{cm}|\omega_{\partial m})$? It is a property of MRFs that we can express the conditional prior distribution in the form of a Gibbs distribution

$$p(\omega_{cm}|\omega_{\partial m}) = \frac{1}{Z}\exp\{-U(\omega_{cm})\} \tag{8.20a}$$

in which (based on the so-called Ising model)

$$U(\omega_{cm}) = \sum_{\partial m} \beta[1 - \delta(\omega_{cm}, \omega_{\partial m})] \tag{8.20b}$$

where $\delta(\omega_{cm}, \omega_{\partial m})$ is the Kroneker delta, which is unity if the arguments are equal and zero otherwise; $\beta > 0$ is a parameter with value fixed by the user when applying the MRF technique to control the influence of the neighbours.

Equation (8.20) is now substituted into (8.19) to generate a posterior probability that depends on the class conditional probability found from the available spectral measurements (the first term on the right hand side) and the effect of the spatial neighbourhood. However, as with (8.4), it is convenient to take the logarithm of (8.19) to yield (with the choice of $Z = 1$), an MRF-based discriminant function for the class on pixel m assuming a multivariate normal class conditional density function:

$$g_{cm}(\boldsymbol{x}_m) = -\frac{1}{2}\ln|\Sigma_c| - \frac{1}{2}(\boldsymbol{x}_m - \boldsymbol{m}_c)\Sigma_c^{-1}(\boldsymbol{x}_m - \boldsymbol{m}_c)^t - \sum_{\partial m}\beta[1 - \delta(\omega_{cm}, \omega_{\partial m})] \; .$$

Recall that classification is carried out on the basis of finding the class for the pixel that maximises the discriminant function. Noting the negative signs above, the most appropriate class for pixel m can be found by minimising the expression

$$d_{cm}(\boldsymbol{x}_m) = \frac{1}{2}\ln|\Sigma_c| + \frac{1}{2}(\boldsymbol{x}_m - \boldsymbol{m}_c)\Sigma_c^{-1}(\boldsymbol{x}_m - \boldsymbol{m}_c)^t + \sum_{\partial m}\beta[1 - \delta(\omega_{cm}, \omega_{\partial m})] \tag{8.21}$$

To use (8.21) there needs to be an allocation of classes over the scene before the last term can be computed. Accordingly, an initial classification would be performed, say with the maximum likelihood classifier of Sect. 8.2.3. Equation (8.21) would then be used to modify the labels attached to the individual pixels to incorporate the effect of context. However, in so doing some (or initially many) of the labels on the pixels will be modified. The process should then be run again, and indeed as many times presumably until there are no further changes.

8.9 Non-parametric Classification: Geometric Approaches

Statistical classification algorithms are the most commonly encountered labelling techniques used in remote sensing and, for this reason, have been the principal methods treated in this chapter. One of the valuable aspects of a statistical approach is that a *set* of relative likelihoods is produced. Even though, in the majority of cases, the *maximum* of the likelihoods is chosen to indicate the most probable label for a pixel, there remains nevertheless information in the remaining likelihoods that could be made use of in some circumstances, either to initiate a process such as relaxation labelling (Sect. 8.8.4) or simply to provide the user with some feeling for the other likely classes. Those situations are however not common and, in most applications, the maximum selection is made. That being so, the material in Sects. 8.2.4 and 8.3.4

shows that the decision process has a geometric counterpart in that a comparison of statistically derived discriminant functions leads equivalently to a decision rule that allows a pixel to be classified on the basis of its position in multispectral space compared with the location of a decision surface. This leads us to question whether a geometric interpretation can be adopted in general, without needing first to use statistical models.

8.9.1 Linear Discrimination

8.9.1.1 Concept of a Weight Vector

Consider the simple two class multispectral space shown in Fig. 8.11, which has been constructed intentionally so that a simple straight line can be drawn between the pixels as shown. This straight line, which will be a multidimensional linear surface in general and which is called a hyperplane, can function as a decision surface for classification. In the two dimensions shown, the equation of the line can be expressed

$$w_1 x_1 + w_2 x_2 + w_3 = 0$$

where the x_i are the brightness value co-ordinates of the multispectral space and the w_i are a set of coefficients, usually called weights. There will be as many weights as the number of channels in the data, plus one. In general, if the number of channels or bands is N, the equation of a linear surface is

$$w_1 x_1 + w_2 x_2 + \ldots + w_N x_N + w_{N+1} = 0$$

which can be written as

$$\boldsymbol{w}^t \boldsymbol{x} + w_{N+1} = 0 \tag{8.22}$$

where $\boldsymbol{x}$ is the co-ordinate vector and $\boldsymbol{w}$ is called the weight vector. The transpose operation has the effect of turning the column vector into a row vector so that the product gives the correct expanded form of the previous equation.

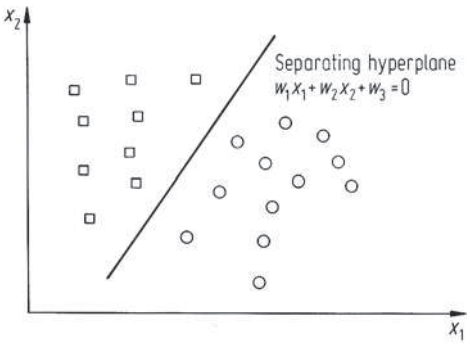


Fig. 8.11. Two dimensional multispectral space, with two classes of pixel that can be separated by a linear surface

In a real exercise the position of the separating surface would be unknown initially. Training a linear classifier amounts to determining an appropriate set of the weights that places the decision surface between the two sets of training samples. There is not necessarily a unique solution – any of an infinite number of (marginally different) decision hyperplanes will suffice to separate the two classes.

For a given data set, an explicit equation for the separating surface can be obtained using the minimum distance rule, as discussed in Sect. 8.3, which entails finding the mean vectors of the two class distributions. An alternative method is outlined in the following, based on selecting an arbitrary surface and then iterating it into an acceptable position. Even though not often used anymore, this method is useful to consider since it establishes some of the concepts used in neural networks and support vector machines (see Sect. 8.9.2).

8.9.1.2 Testing Class Membership

The calculation in (8.22) will be exactly zero only for values of $\boldsymbol{x}$ lying on the decision surface. If we substitute into that equation values of $\boldsymbol{x}$ corresponding to the pixel points indicated in Fig. 8.11 the left hand side will be non-zero. For pixels in one class a positive result will be given, while pixels on the other side will give a negative result. Thus, once the decision surface has been identified (i.e. trained), then a decision rule is

$$\begin{aligned} \boldsymbol{x} &\in \text{ class 1 if } \boldsymbol{w}^t\boldsymbol{x} + w_{N+1} > 0 \\ \boldsymbol{x} &\in \text{ class 2 if } \boldsymbol{w}^t\boldsymbol{x} + w_{N+1} < 0 \end{aligned} \tag{8.23}$$

8.9.1.3 Training

A full discussion of linear classifier training is given in Nilsson (1965, 1990); only those aspects helpful to the neural network development following are treated here.

It is expedient to define a new, augmented pixel vector according to

$$\boldsymbol{y} = [\boldsymbol{x}^t, 1]^t$$

If, in (8.22), we also take the term w_{N+1} into the definition of the weight vector, viz.

$$\boldsymbol{w} = [\boldsymbol{w}^t, w_{N+1}]^t$$

then the equation of the decision surface, can be expressed more compactly as

$$\boldsymbol{w}^t\boldsymbol{y} = 0 \quad \text{(or equivalently } \boldsymbol{w} \cdot \boldsymbol{y} = 0)$$

so that the decision rule of (8.23) can be restated

$$\begin{aligned} \boldsymbol{x} &\in \text{ class 1 if } \boldsymbol{w}^t\boldsymbol{y} > 0 \\ \boldsymbol{x} &\in \text{ class 2 if } \boldsymbol{w}^t\boldsymbol{y} < 0 \end{aligned} \tag{8.24}$$

We usually think of $\boldsymbol{w}^t\boldsymbol{y} = 0$ as defining a linear surface in the $\boldsymbol{x}$ (or now $\boldsymbol{y}$) multispectral space, in which the coefficients of the variables (y_1, y_2, etc.) are the

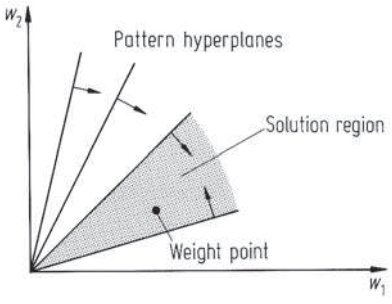


Fig. 8.12. Representation of pixels as hyperplanes and weight vectors as points in so-called weight space. The arrows indicate the side of each pixel plane on which the weight point must lie for correct classification

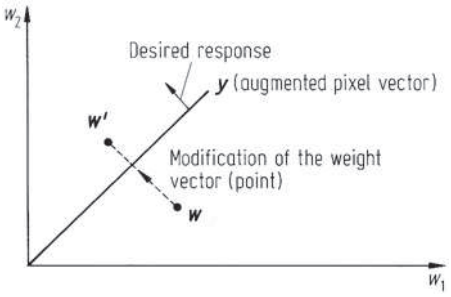


Fig. 8.13. Modification of the weight point to give the correct response

weights w_1, w_2, etc. However it is also possible to think of the equation as describing a linear surface in which the y's are the coefficients and the w's are the variables. This interpretation will see these surfaces plotted in a co-ordinate system which has axes w_1, w_2, etc. A two-dimensional version of this *weight space*, as it is called, is shown in Fig. 8.12, in which have been plotted a number of *pattern hyperplanes*; these are specific linear surfaces in the new co-ordinates that pass through the origin and have, as their coefficients, the components of the (augmented) pixel vectors. Thus, while the pixels plot as points in multispectral space, they plot as linear surfaces in weight space. Likewise, a set of weight coefficients will define a surface in multispectral space, but will plot as a point in weight space. Although this is an abstract concept it will serve to facilitate an understanding of how a linear classifier can be trained.

In weight space the decision rule of (8.24) still applies – however now it tests that the *weight point* is on the appropriate side of the *pattern hyperplane.* For example, Fig. 8.12 shows a single weight point which lies on the correct side of each pixel and thus defines a suitable decision surface in multispectral space. In the diagram, small arrows are attached to each pixel hyperplane to indicate the side on which the weight point must lie in order that the test of (8.24) succeeds for all pixels. The purpose of training the linear classifier is to ensure that the weight point is located somewhere within the *solution region.* If, through some initial guess, the weight point is located somewhere else in weight space then it has to be moved to the solution region.

Suppose an initial guess is made for the weight vector $\boldsymbol{w}$, but that this places the weight point on the wrong side of a particular pixel hyperplane as illustrated in Fig. 8.13. Clearly, the weight point has to be shifted to the other side to give a

correct response in (8.24). The most direct manner in which the weight point can be modified is to move it straight across the pixel hyperplane. This can be achieved by adding a scaled amount of the pixel vector to the weight vector[4]. The new position of the weight point is then

$$\boldsymbol{w}' = \boldsymbol{w} + c\,\boldsymbol{y} \tag{8.25}$$

where c is called the correction increment, the size of which determines by how much the original weight point is moved orthogonal to the pixel hyperplane. If it is large enough the weight point will be shifted right across the pixel plane, as required. Having so modified the weight vector, the product in (8.24) then becomes

$$\begin{aligned}\boldsymbol{w}'^t\,\boldsymbol{y} &= \boldsymbol{w}^t\,\boldsymbol{y} + c\,\boldsymbol{y}^t\,\boldsymbol{y} \\ &= \boldsymbol{w}^t\,\boldsymbol{y} + c|\boldsymbol{y}|^2\end{aligned}$$

Clearly, if the initial $\boldsymbol{w}^t\,\boldsymbol{y}$ was erroneously negative a suitable positive value of c will give a positive value of $\boldsymbol{w}'^t\,\boldsymbol{y}$; otherwise a negative value of c will correct an erroneous initial positive value of the product.

Using the class membership test in (8.24) and the correction formula of (8.25) the following iterative nonparametric training procedure, referred to as *error correction feedback*, is adopted.

First, an initial position for the weight point is chosen arbitrarily. Then, pixel vectors from training sets are presented one at a time. If the current weight point position classifies a pixel correctly then no action need be taken; otherwise the weight vector is modified as in (8.25) with respect to that particular pixel vector. This procedure is repeated for each pixel in the training set, and the set is scanned as many times as necessary to move the weight point into the solution region. If the classes are linearly separable then such a solution will be found.

8.9.1.4 Setting the Correction Increment

Several approaches can be adopted for choosing the value of the correction increment, c. The simplest is to set c equal to a positive or negative constant (according to the change required in the $\boldsymbol{w}^t\,\boldsymbol{y}$ product). A common choice is to make $c = \pm 1$ so that application of (8.25) amounts simply to adding the augmented pixel vector to or subtracting it from the weight vector, thereby obviating multiplications and giving fast training.

Another rule is to choose the correction increment proportional to the difference between the desired and actual response of the classifier:

$$c = \eta(t - \boldsymbol{w}^t\,\boldsymbol{y})$$

[4] The hyperplane in $\boldsymbol{w}$ coordinates is given by $\boldsymbol{w}^t\,\boldsymbol{y} = 0$; a vector normal to that hyperplane is the vector of the coefficients of $\boldsymbol{w}$. This can be checked for a simple two dimensional example. A line through the origin with unity slope is $-w_1 + w_2 = 0$. A vector normal to the line joins the origin to $(-1, 1)$, i.e. $\boldsymbol{y} = [-1, 1]^t$.

so that (8.25) can be written

$$\boldsymbol{w}' = \boldsymbol{w} + \Delta\boldsymbol{w}$$

with

$$\Delta\boldsymbol{w} = \eta(t - \boldsymbol{w}^t\boldsymbol{y})\boldsymbol{y} \tag{8.26}$$

where t is the desired response to the training pattern $\boldsymbol{y}$ and $\boldsymbol{w}^t\boldsymbol{y}$ is the actual response; η is a factor which controls the degree of correction applied. Usually t would be chosen as $+1$ for one class and -1 for the other.

8.9.1.5 Classification – The Threshold Logic Unit

After the linear, two category classifier has been trained, so that the final version of the weight vector $\boldsymbol{w}$ is available, it is ready to be presented with pixels it has not seen before in order to attach ground cover class labels to those pixels. This is achieved through application of the decision rule in (8.24). It is useful, in anticipation of neural networks, to picture the classification rule in diagrammatic form as depicted in Fig. 8.14a. Simply, this consists of weighting elements, a summing device and an output element which, in this case, performs the maximum selection. Together these are referred to as a *threshold logic unit* (TLU). It bears substantial similarity to the concept of a *processing element* used in neural networks for which the output thresholding unit is replaced by a more general function and the pathway for the unity input in the augmented pattern vector is actually incorporated into the output function. The latter can be done for a simple TLU as shown in Fig. 8.14b, in which the simple thresholding element has been replaced by a functional block which performs

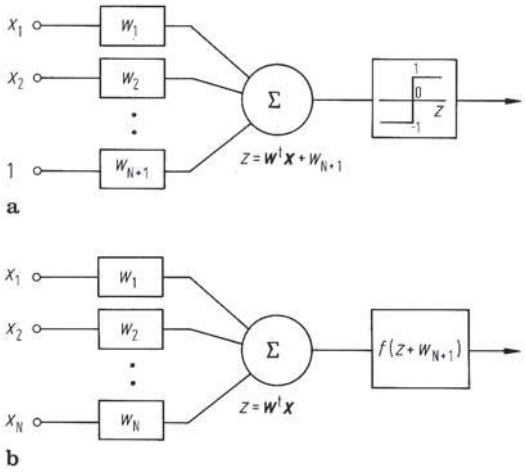


Fig. 8.14. a Diagrammatic representation of (8.24). **b** More useful representation of a processing element in which the thresholding function is generalised

the addition of the final weighting coefficient to the weighted sum of the input pixel components, and then performs a thresholding (or more general nonlinear) operation.

8.9.1.6 Multicategory Classification

The foregoing work on linear classification has been based on an approach that can perform separation of pixel vectors into just two categories. Were it to be considered for remote sensing, it needs to be extended to be able to cope with a multiclass problem.

Multicategory classification can be carried out in one of two ways. First a decision tree of linear classifiers (TLUs) can be constructed, as seen in Fig. 8.15, at each decision node of which a decision of the type (water or not water) is made. At a subsequent node the (not water) category might be differentiated as (soil or not soil) etc. It should be noted that the decision process at each node has to be trained separately.

Alternatively, a multicategory version of the simple binary linear classifier can be derived. This reverts, for its derivation, to the concept of a discriminant function and, specifically, defines the linear classifier discriminant function for class i as

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^t \boldsymbol{y} \qquad i = 1, \ldots M$$

Class membership is then decided on the basis of the usual decision rule expressed in (8.11a), i.e. according to the largest of the $g_i(\boldsymbol{x})$ for the given pixel vector $\boldsymbol{x}$. For training, an initial arbitrary set of weight vectors and thus discriminant functions is chosen. Then each of the training pixels is checked in turn. Suppose, for a particular pixel the jth discriminant function is erroneously largest, when in fact the pixel belongs the ith category. A correction is carried out by adjusting the weight vectors for these two discriminant functions, to increase that for the correct class for the pixel and to decrease that for the incorrect class, according to

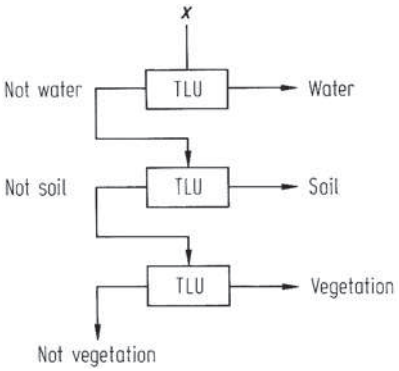


Fig. 8.15. Binary decision tree of TLUs used for multicategory classification

$$\begin{aligned} \boldsymbol{w}_i' &= \boldsymbol{w}_i + c\,\boldsymbol{y} \\ \boldsymbol{w}_j' &= \boldsymbol{w}_j - c\,\boldsymbol{y} \end{aligned} \tag{8.27}$$

where c is the correction increment. Again this correction procedure is iterated over the training set of pixels as many times as necessary to obtain a solution. Nilsson (1965, 1990) shows that a solution is possible by this approach.

8.9.2 Support Vector Classifiers

8.9.2.1 Linearly Separable Data

The training process outlined in Sect. 8.9.1.3 can lead to many, non-unique, yet acceptable solutions for the weight vector. The actual size of the solution region depicted in Fig. 8.12 is an indication of that. Also, every training pixel takes part in the training process; yet examination of Fig. 8.11 suggests that it is only those pixels in the vicinity of the separating hyperplane that define where the hyperplane needs to lie in order to give a reliable classification.

The support vector machine (SVM) provides a training approach that depends only on those pixels in the vicinity of the separating hyperplane (called the support *pixel* vectors). It also leads to a hyperplane position that is in a sense optimal for the available training patterns, as will be seen shortly.

The support vector concept was introduced to remote sensing image classification by Gualtieri and Cromp (1998). Two recent reviews that contain more detail than is given in the following treatment are by Burges (1998) and Huang et al. (2002). A very good recent treatment from a remote sensing perspective has been given by Melgani and Bruzzone (2004).

If we expand the region in the vicinity of the hyperplane in Fig. 8.11 we can see, as suggested in Fig. 8.16, that the optimal orientation of the hyperplane is when there is a maximum separation between the patterns in the two classes. We can then draw two further hyperplanes parallel to the separating hyperplane, as shown, bordering the nearest training pixels from the two classes. The equations for the hyperplanes are shown in the figure. Note that the choice of unity on the right hand side of the equations for the two marginal hyperplanes is arbitrary, but it helps in the analysis. If it were otherwise it could be scaled to unity by appropriately scaling the weighting coefficients w_k. Note that for pixels that lie *beyond the marginal hyperplanes*, we have

$$\text{for class 1 pixels} \qquad \boldsymbol{w}.\boldsymbol{x} + w_{N+1} \geq 1 \tag{8.28a}$$

$$\text{for class 2 pixels} \qquad \boldsymbol{w}.\boldsymbol{x} + w_{N+1} \leq -1 \tag{8.28b}$$

It is useful now to describe the class label of the ith pixel by the variable y_i, which takes the value $+1$ for class 1 and -1 for class 2 pixels. Equations (8.28a) and (8.28b) can then be written as a single expression valid for pixels from both classes:

$$(\boldsymbol{w}.\boldsymbol{x} + w_{N+1})y_i \geq 1 \text{ for pixel } i \text{ in its correct class.}$$

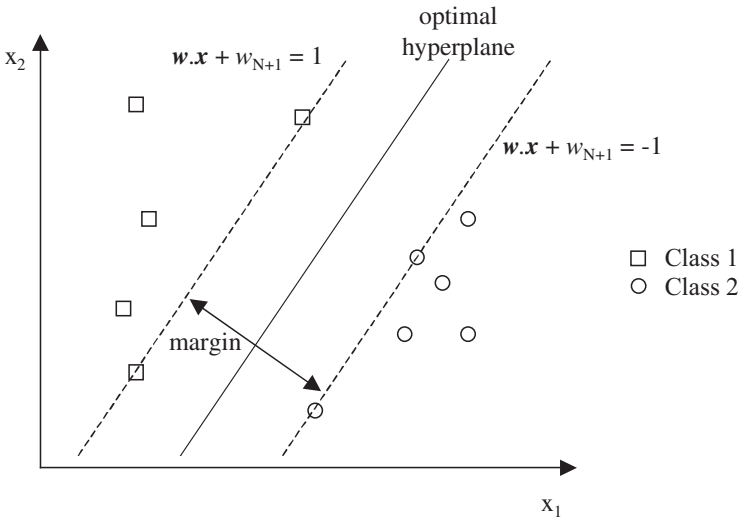


Fig. 8.16. Expanded version of Fig. 8.11 showing that an optimal separating hyperplane orientation exists determined by finding the maximum separation between the training pixels. Under that condition two marginal hyperplanes can be constructed using only those pixels vectors closest to the separating surface

Alternatively

$$\boldsymbol{w}.\boldsymbol{x} + w_{N+1})y_i - 1 \geq 0 \tag{8.29}$$

Equation (8.29) must hold for all pixels if the data is linearly separated by the two marginal hyperplanes of Fig 8.16. Those hyperplanes, defined by the equalities in (8.28), are described by

$$\boldsymbol{w}.\boldsymbol{x} + w_{N+1} - 1 = 0$$
$$\boldsymbol{w}.\boldsymbol{x} + w_{N+1} + 1 = 0$$

The perpendicular distances of these hyperplanes from the origin, respectively, are $-(w_{N+1} - 1)/||\boldsymbol{w}||$ and $-(w_{N+1} + 1)/||\boldsymbol{w}||$, where $\|\boldsymbol{w}\|$ is the Euclidean length of the weight vector. Therefore, the distance between the two hyperplanes, which is the *margin* in Fig. 8.16, is $2/\|\boldsymbol{w}\|$.

The best position (orientation) for the separating hyperplane will be that for which $2/\|\boldsymbol{w}\|$ is a maximum, or equivalently when the magnitude of the weight vector, $\|\boldsymbol{w}\|$, is a minimum. However there is a constraint! As we seek to maximise the margin between the two marginal hyperplanes by minimising $\|\boldsymbol{w}\|$ we must not allow (8.29) to be invalidated. In other words, all the training pixels must be on their correct side of the marginal hyperplanes. We handle the process of minimising $\|\boldsymbol{w}\|$ subject to that constraint by the process known as Lagrange multipliers. This requires us to set up a function (called the Lagrangrian) which includes the expression to be minimised ($\|\boldsymbol{w}\|$) from which is subtracted a proportion (α_i) of each constraint (one for each training pixel) in the following manner:

$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i \{y_i(\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1}) - 1\} \tag{8.30}$$

The α_i are called Lagrange multipliers and are positive by definition, i.e.

$$\alpha_i \geq 0 \text{ for all } i.$$

By minimising L we minimise $\|\boldsymbol{w}\|$ subject to the constraint (8.29).

In (8.30) it is convenient to substitute

$$f(\boldsymbol{x}_i) = (\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1})y_i - 1$$

to give

$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i f(x_i)$$

noting that for pixels in their correct class $f(\boldsymbol{x}_i) \geq 0$.

It is useful here to remember what our task is. We have to find the *most* separated marginal hyperplanes in Fig. 8.16. In other words we need to find the $\boldsymbol{w}$ and w_{N+1} that minimises L and thus maximises the *margin* shown in the figure.

But in seeking to minimise L (essentially during the training process) how do we treat the α_i? Suppose (8.29) is violated, as could happen for some pixels during training; then $f(\boldsymbol{x}_i)$ will be negative. Noting that α_i is positive that would cause L to increase. But we need to find values for $\boldsymbol{w}$ and w_{N+1} such that L is minimised. The worst possible case to handle is when the α_i are such as to cause L to be a maximum, since that forces us to minimise L with respect to $\boldsymbol{w}$ and w_{N+1} while the α_i are trying to make it as large as possible. The most robust approach to finding $\boldsymbol{w}$ and w_{N+1} (and thus the hyperplanes) therefore is to find the values of $\boldsymbol{w}$ and w_{N+1} that minimise L while simultaneously finding the α_i that try to maximise it.

Thus we require, first, that:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$$

$$\text{so that } \boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i \ . \tag{8.31a}$$

Secondly we require:

$$\frac{\partial L}{\partial w_{N+1}} = -\sum_i \alpha_i y_i = 0$$

$$\text{so that } \sum_i \alpha_i y_i = 0 \ . \tag{8.31b}$$

Before proceeding, examine (8.30) again, this time for training pixels that satisfy the requirement of (8.29). What value(s) of α_i in (8.30) for those pixels maximise L? Since $y_i(\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1}) - 1$ is now always positive then the only (non-negative) value of α_i that makes L as big as possible is $\alpha_i = 0$. Therefore, for any training

pixels on the correct side of the marginal hyperplanes, $\alpha_i = 0$. This is an amazing, yet intuitive, result. It says we do not have to use any of the training pixel vectors, other than those that reside exactly on one of the marginal hyperplanes. The latter are called *support vectors* since they are the only ones that support the process of finding the marginal hyperplanes. Thus, in applying (8.31a) to find $\boldsymbol{w}$ we only have to use those pixels on the marginal hyperplanes.

But the training is not yet finished! We still have to find the relevant α_i (i.e. those that maximise L and are non-zero).

To proceed, note that we can put $\|\boldsymbol{w}\| = \boldsymbol{w}.\boldsymbol{w}$ in (8.30). Now (8.30), along with (8.31a), can be written most generally as:

$$\begin{aligned} L &= \frac{1}{2}\left(\sum_i \alpha_i y_i \boldsymbol{x}_i\right)\cdot\left(\sum_j \alpha_j y_j \boldsymbol{x}_j\right) \\ &\qquad - \sum_i \alpha_i \left[y_i \left(\left(\sum_j \alpha_j y_j \boldsymbol{x}_j \right).\boldsymbol{x}_i + w_{N+1} \right) - 1 \right] \\ &= \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i.\boldsymbol{x}_j - \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i.\boldsymbol{x}_j - w_{N+1}\sum_i \alpha_i y_i + \sum_i \alpha_i \end{aligned}$$

Using (8.31b) this simplifies to

$$L = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i.\boldsymbol{x}_j \tag{8.32}$$

which has to be maximised by the choice of α_i. This usually requires a numerical procedure to solve for any real problem. Once we have found the α_i – call them α_i° – we can substitute them into (8.31a) to give the optimal training vector:

$$\boldsymbol{w}^o = \sum_i \alpha_i^o y_i \boldsymbol{x}_i \tag{8.33a}$$

But we still do not have a value for w_{N+1}. Recall that on a marginal hyperplane

$$(\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1})y_i - 1 = 0\ .$$

Choose two support (training) vectors $\boldsymbol{x}(1)$ and $\boldsymbol{x}(-1)$ on each of the two marginal hyperplanes respectively for which $y = 1$ and -1. For these vectors we have

$$\boldsymbol{w}.\boldsymbol{x}(1) + w_{N+1} - 1 = 0$$

and

$$-\boldsymbol{w}.\boldsymbol{x}(-1) - w_{N+1} - 1 = 0$$

so that

$$w_{N+1} = \frac{1}{2}(\boldsymbol{w}.\boldsymbol{x}(1) + \boldsymbol{w}.\boldsymbol{x}(-1))\ . \tag{8.33b}$$

Normally sets of $\boldsymbol{x}(1)$, $\boldsymbol{x}(-1)$, would be used, with w_{N+1} found by averaging.

With the values of α_i° determined by numerical optimisation, (8.33a,b) now give the parameters of the separating hyperplane that provides the largest margin between the two sets of training data. In terms of the training data, the equation of the hyperplane is:

$$\boldsymbol{w}^\circ.\boldsymbol{x} + w_{N+1} = 0$$

so that the discriminant function, for an unknown pixel $\boldsymbol{x}$ is

$$g(\boldsymbol{x}) = \text{sgn}\ (\boldsymbol{w}^\circ.\boldsymbol{x} + w_{N+1}) \tag{8.34}$$

8.9.2.2 Linear Inseparability – The Use of Kernel Functions

If the pixel space is not linearly separable then the development of the previous section will not work without modification. A transformation of the pixel vector $\boldsymbol{x}$ to a different (usually higher order) feature space can be applied that renders the data linearly separable allowing the earlier material to be applied.

The two significant equations for the linear support vector approach of the preceding section are (8.32) (for finding α_i°) and (8.34) (the resulting discriminant function). By using (8.33a), (8.34) can be rewritten

$$g(\boldsymbol{x}) = \text{sgn}\ \left\{\sum \alpha_i^\circ y_i \boldsymbol{x}_i.\boldsymbol{x} + w_{N+1}\right\} \tag{8.35}$$

Now introduce the feature space transformation $\boldsymbol{x} \rightarrow \Phi(\boldsymbol{x})$ so that (8.32) and (8.35) become

$$L = \sum_i \alpha_i - 0.5 \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi\ (\boldsymbol{x}_i)\ .\Phi\ \left(\boldsymbol{x}_j\right) \tag{8.36a}$$

$$g(\boldsymbol{x}) = \text{sgn}\ \left\{\sum \alpha_i^\circ y_i \Phi(\boldsymbol{x}_i).\Phi(\boldsymbol{x}) + w_{N+1}\right\} \tag{8.36b}$$

In both (8.32) and (8.35) the pixel vectors occur only in the dot products. As a result the $\Phi(\boldsymbol{x})$ also appear only in dot products. So to use (8.36) it is strictly not necessary to know $\Phi(\boldsymbol{x})$ but only a scalar quantity equivalent to $\Phi(\boldsymbol{x}_i).\Phi(\boldsymbol{x}_j)$.

We call the product $\Phi(\boldsymbol{x}_i).\Phi(\boldsymbol{x}_j)$ a *kernel*, and represent it by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ so that (8.36) becomes

$$L = \sum_i \alpha_i - 0.5 \sum_{i,j} \alpha_i \alpha_j y_i y_j k\left(\boldsymbol{x}_i.\boldsymbol{x}_j\right)$$

$$g(\boldsymbol{x}) = \text{sgn}\ \left\{\sum \alpha_i^\circ y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + w_{N+1}\right\}$$

Provided we know the form of the kernel we never actually need to know the underlying transformation $\Phi(\boldsymbol{x})$! Thus, after choosing $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ we then find the α_i° that maximise L and use that value in $g(\boldsymbol{x})$ to perform a classification.

Two commonly used kernels in remote sensing are:

The polynomial kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = [(\boldsymbol{x}_i).(\boldsymbol{x}_j) + 1]^p$

The radial basis function kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$

in which p and γ are constants to be chosen.

8.9.2.3 Multicategory Classification

As in Sect. 8.9.1.6, the binary classifiers developed above can be used to perform multicategory classification by embedding them in a binary decision tree.

8.9.3 Networks of Classifiers – Solutions of Nonlinear Problems

The decision tree structure shown in Fig. 8.15 is a classifier network in that a collection of simple classifiers (in that case TLUs) is brought together to solve a complex problem. Nilsson (1965, 1990) has proposed a general network structure under the name of *layered classifiers* consisting entirely of interconnected TLUs, as shown in Fig. 8.17. The benefit of forming a classifier network is that data sets that are inherently not separable with a simple linear decision surface should, in principle, be able to be handled since the layered classifier is known to be capable of implementing nonlinear surfaces. The drawback however, is that training procedures for layered classifiers, consisting of *TLU*s, are difficult to determine.

One specific manifestation of a layered classifier, known as a committee machine, is depicted in Fig. 8.18. Here the first layer consists simply of a set of TLUs, to

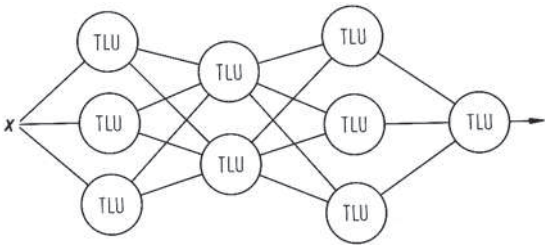


Fig. 8.17. Layered TLU classifier

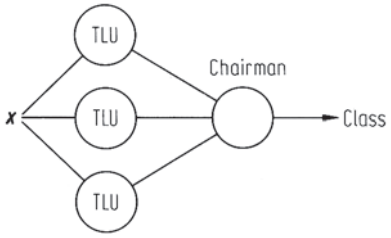


Fig. 8.18. Committee classifier

each of which a pixel vector under test is submitted to see which of two classes is recommended. The second layer is a single element which has the responsibility of judging the recommendations of each of the nodes in the first layer. It is therefore of the nature of a chairman or vote taker. It can make its decision on the basis of several sets of logic. First, it can decide class membership on the basis of the majority vote of the first layer recommendations. Secondly, it can decide on the basis of veto, in which all first layer classifiers have to agree before the vote taker will recommend a class. Thirdly, it could use a form of seniority logic in which the chairman rank orders the decisions of the first layer nodes. It always refers to one first. If that node has a solution then the vote taker accepts it and goes no further. Otherwise it consults the next most senior of the first layer nodes, etc. A committee classifier based on seniority logic has been developed for remote sensing applications by Lee and Richards (1985).

8.9.4 The Neural Network Approach

For the purposes of this treatment a neural network is taken to be of the nature of a layered classifier such as depicted in Fig. 8.17, but with the very important difference that the nodes are not TLUs, although resembling them closely. The node structure in Fig. 8.14b can be made much more powerful, and coincidentally lead to a training theorem for multicategory nonlinear classification, if the output processing element does not apply a thresholding operation to the weighted input but rather applies a softer, and mathematically differentiable, operation.

8.9.4.1 The Processing Element

The essential processing node in the neural network to be considered here (sometimes called a neuron by analogy to biological data processing from which the term neural network derives) is an element as shown in Fig. 8.14b with many inputs and with a single output, depicted simply in Fig. 8.19a. Its operation is described by

$$o = f(\boldsymbol{w}^t \boldsymbol{x} + \theta) \tag{8.37}$$

where θ is a threshold (sometimes set to zero), $\boldsymbol{w}$ is a vector of weighting coefficients and $\boldsymbol{x}$ is the vector of inputs. For the special case when the inputs are the band values of a particular multispectral pixel vector it could be envisaged that the threshold θ takes the place of the weighting coefficient w_{N+1} in (8.22). If the function f is a thresholding operation this processing element would behave as a TLU. In general, the number of inputs to a node will be defined by network topology as well as data dimensionality, as will become evident.

The major difference between the layered classifier of TLUs shown in Fig. 8.17 and the neural network, known as the multilayer perceptron, is in the choice of the function f, called the *activation function.* Its specification is simply that it emulate

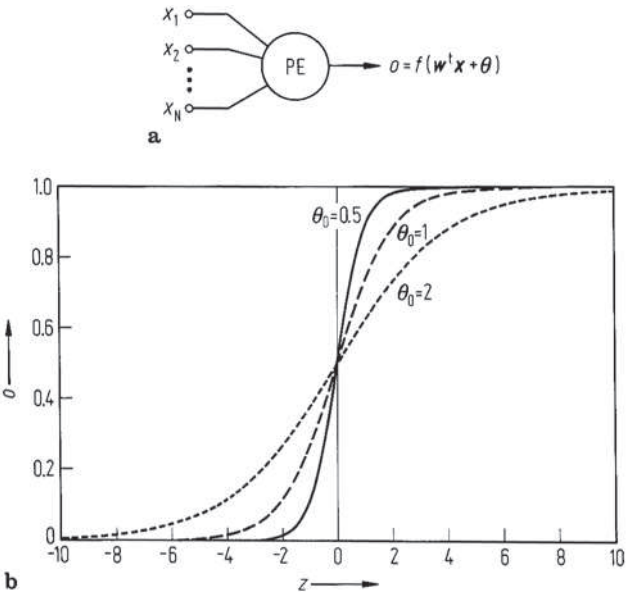


Fig. 8.19. a Neural network processing element. **b** Plots of (8.38) for various θ_0

thresholding in a soft or asymptotic sense and be differentiable. The most commonly encountered expression is

$$f(z) = \frac{1}{1 + \mathrm{e}^{-z/\theta_0}} \tag{8.38}$$

where the argument z is $\boldsymbol{w}^t\boldsymbol{x} + \theta$ as seen in (8.37) and θ_0 is a constant. This approaches 1 for z large and positive and 0 for z large and negative and is thus asymptotically thresholding. It is important to recognise that the outcome of the product $\boldsymbol{w}^t\boldsymbol{x}$ is a simple scalar; when plotted with $\theta = 0$, (8.38) appears as shown in Fig. 8.19b. For θ_0 very small the activation function approaches a thresholding operation. Usually $\theta_0 = 1$.

A neural network for use in remote sensing image analysis will appear as shown in Fig. 8.20, being a layered classifier composed of processing elements of the type shown in Fig. 8.19a. It is conventionally drawn with an input layer of nodes (which has the function of distributing the inputs to the processing elements of the next layer, and scaling them if necessary) and an output layer from which the class labelling information is provided. In between there may be one or more so-called hidden or other processing layers of nodes. Usually one hidden layer will be sufficient, although the number of nodes to use in the hidden layer is often not readily determined. We return to this issue in Sect. 8.9.4.3 below.

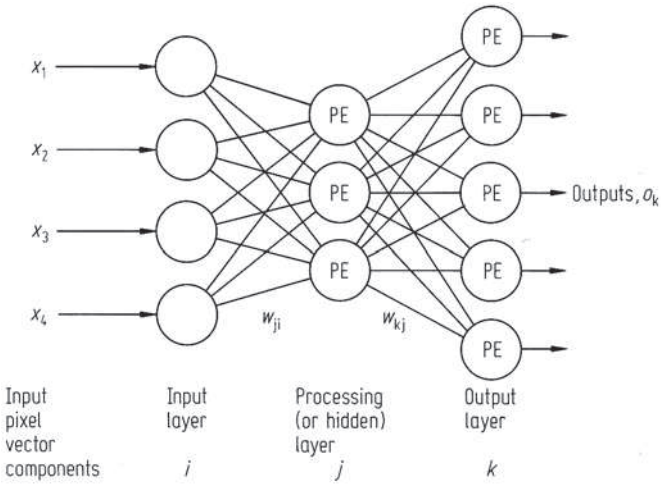


Fig. 8.20. A multilayer perceptron neural network, and the nomenclature used in the derivation of the backpropagation training algorithm

8.9.4.2 Training the Neural Network – Backpropagation

Before it can perform a classification, the network of Fig. 8.20 must be trained. This amounts to using labelled training data to help determine the weight vector $\boldsymbol{w}$ and the threshold θ in (8.37) for each processing element connected into the network. Note that the constant θ_0 in (8.38), which governs the gradient of the activation function as seen in Fig. 8.19b, is generally pre-specified and does not need to be estimated from the training data.

Part of the complexity in understanding the training process for a neural net is caused by the need to keep careful track of the parameters and variables over all layers and processing elements, how they vary with the presentation of training pixels and (as it turns out) with iteration count. This can be achieved with a detailed subscript convention, or by the use of a simpler generalised notation. We will adopt the latter approach, following essentially the development given by Pao (1989). The derivation will be focussed on a 3 layer neural net, since this architecture has been found sufficient for many applications. However the results generalise to more layers.

Figure 8.20 incorporates the nomenclature used. The three layers are lettered as i, j, k with k being the output. The set of weights linking layer i PEs with those in layer j are represented generally by w_{ji}, while those linking layers j and k are represented by w_{kj}. There will be a very large number of these weights, but in deriving the training algorithm it is not necessary to refer to them all individually. Similarly the general activation function arguments z_i and outputs o_i, can be used to represent all the arguments and outputs in the corresponding layer.

For j and k layer PEs (8.37) is

$$o_j = f(z_j) \quad \text{with} \qquad z_j = \sum_j w_{ji} o_i + \theta_j \tag{8.39a}$$

$$o_k = f(z_k) \quad \text{with} \qquad z_k = \sum_k w_{kj} o_j + \theta_k \tag{8.39b}$$

The sums in (8.39) are shown with respect to the indices j and k. This should be read as meaning the sums are taken over all inputs of particular layer j and layer k PEs respectively. Note also that the sums are expressed in terms of the outputs of the previous layer since these outputs form the inputs to the PEs in question.

An untrained or poorly trained network will give erroneous outputs. Therefore, as a measure of how well a network is functioning during training, we can assess the outputs at the last layer (k). A suitable measure along these lines is to use the sum of the squared output error. The error made by the network when presented with *a single training pixel* can thus be expressed

$$E = \frac{1}{2} \sum_k (t_k - o_k)^2 \tag{8.40}$$

where the t_k represent the desired or target outputs[5] and o_k represents the actual outputs from the output layer PEs in response to the training pixel. The factor of $\frac{1}{2}$ is included for arithmetic convenience in the following. The sum is over all output layer PEs.

A useful training strategy is to adjust the weights in the processing elements until the error has been minimised, at which stage the actual outputs are as close as possible to the desired outputs.

A common approach for adjusting weights to reduce (and thus minimise) the value of a function of which they are arguments, is to modify their values proportional to the negative of the partial derivative of the function. This is called a gradient descent technique[6]. Thus for the weights linking the j and k layers let

$$w'_{kj} = w_{kj} + \Delta w_{kj}$$

with

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}$$

where η is a positive constant that controls the amount of adjustment. This requires an expression for the partial derivative, which can be determined using the chain rule

[5] These will be specified from the training data labelling. The actual value taken by t_k however will depend on how the outputs themselves are used to represent classes. Each output could be a specific class indicator (e.g. 1 for class 1 and 0 class 2); alternatively some more complex coding of the outputs could be adopted. This is considered in Sect. 8.9.4.3.

[6] Another optimisation procedure used successfully for neural network training in remote sensing is the conjugate gradient method (Benediktsson et al., 1993).

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}} \tag{8.41}$$

each term of which must now be evaluated.

From (8.39b) and (8.38) we see (for $\theta_0 = 1$)

$$\frac{\partial o_k}{\partial z_k} = f'(z_k) = (1 - o_k)o_k \tag{8.42a}$$

and

$$\frac{\partial z_k}{\partial w_{kj}} = o_j \tag{8.42b}$$

Now from (8.40)

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{8.42c}$$

Thus the correction to be applied to the weights is

$$\Delta w_{kj} = \eta(t_k - o_k)(1 - o_k)o_k o_j \tag{8.43}$$

For a given trial, all of the terms in this expression are known so that a beneficial adjustments can be made to the weights which link the hidden layer to the output layer.

Now consider the weights that link the i and j layers. The weight adjustments are

$$\Delta w_{ji} = -\eta\frac{\partial E}{\partial w_{ji}} = -\eta\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}}$$

In a similar manner to the above development we have

$$\Delta w_{ji} = -\eta\frac{\partial E}{\partial o_j}(1 - o_j)o_j o_i$$

Unlike the case with the output layer, however, we cannot obtain an expression for the remaining partial derivative from the error formula, since the o_j are not the outputs at the final layer, but rather those from the hidden layer. Instead we express the derivative in terms of a chain rule involving the output PEs. Specifically

$$\begin{aligned}\frac{\partial E}{\partial o_j} &= \sum_k \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial o_j} \\ &= \sum_k \frac{\partial E}{\partial z_k} w_{kj}\end{aligned}$$

The remaining partial derivative can be obtained from (8.42a) and (8.42c) as

$$\frac{\partial E}{\partial z_k} = -(t_k - o_k)(1 - o_k)o_k$$

so that

$$\Delta w_{ji} = \eta(1 - o_j)o_j o_i \sum_k (t_k - o_k)(1 - o_k)o_k w_{kj} \tag{8.44}$$

Having determined the w_{kj} from (8.43), it is now possible to find values for the w_{ji} since all other entries in (8.44) are known or can be calculated readily.

For convenience we now define

$$\delta_k = (t_k - o_k)(1 - o_k)o_k \tag{8.45a}$$

and

$$\begin{aligned}\delta_j &= (1 - o_j)o_j \sum_k (t_k - o_k)(1 - o_k)o_k w_{kj} \\ &= (1 - o_j)o_j \sum_k \delta_k w_{kj}\end{aligned} \tag{8.45b}$$

so that we have

$$\Delta w_{kj} = \eta \delta_k o_j \tag{8.46a}$$

and

$$\Delta w_{ji} = \eta \delta_j o_i \tag{8.46b}$$

both of which should be compared with (8.26) to see the effect of a differentiable activation function.

The thresholds θ_j and θ_k in (8.39) are found in exactly the same manner as for the weights in that (8.46) is used, but with the corresponding inputs chosen to be unity.

Now that we have the mathematics in place it is possible to describe how training is carried out. The network is initialised with an arbitrary set of weights in order that it can function to provide an output. The training pixels are then presented one at a time to the network. For a given pixel the output of the network is computed using the network equations. Almost certainly the output will be incorrect to start with – i.e. the o_k will not match the desired class t_k for the pixel, as specified by its labelling in the training data. Correction to the output PE weights, described in (8.46a), is then carried out, using the definition of δ_k in (8.45a). With these new values of δ_k and thus w_{kj} (8.45b) and (8.46b) can be applied to find the new weight values in the earlier layers. In this way the effect of the output being in error is propagated back through the network in order to correct the weights. The technique is thus often referred to as *back propagation.*

Pao (1989) recommends that the weights not be corrected on each presentation of a single training pixel, but rather that the corrections for all pixels in the training set be aggregated into a single adjustment. Thus for p training patterns the bulk adjustments are[7]

$$\Delta w'_{kj} = \sum_p \Delta w_{kj} \qquad \text{and} \qquad \Delta w'_{ji} = \sum_k \Delta w_{ji}$$

[7] This is tantamount to deriving the algorithm with the error being calculated over all pixels p in the training set, viz $E_p = \sum_p E$, where E is the error expressed for a single pixel in (8.40).

After the weights have been so adjusted the training pixels are presented to the network again and the outputs re-calculated to see if they correspond better to the desired classes. Usually they will still be in error and the process of weight adjustment is repeated. Indeed the process is iterated as many times as necessary in order that the network respond with the correct class for each of the training pixels or until the number of errors in classifying the training pixels is reduced to an acceptable level.

8.9.4.3 Choosing the Network Parameters

When considering the use of the neural network approach to classification it is necessary to make several key decisions beforehand. First, the number of layers to use must be chosen. Generally, a three layer network is sufficient, with the purpose of the first layer being simply to distribute (or fan out) the components of the input pixel vector to each of the processing elements in the second layer. Thus the first layer does no processing as such, apart perhaps from scaling the input data, if required.

The next choice relates to the number of elements in each layer. The input layer will generally be given as many nodes as there are components (features) in the pixel vectors. The number to use in the output node will depend on how the outputs are used to represent the classes. The simplest method is to let each separate output signify a different class, in which case the number of output processing elements will be the same as the number of training classes. Alternatively, a single PE could be used to represent all classes, in which case a different value or level of the output variable will be attributed to each class. A further possibility is to use the outputs as a binary code, so that two output PEs can represent four classes, three can represent 8 classes and so on.

As a general guide the number of PEs to choose for the hidden or processing layers should be the same as or larger than the number of nodes in the input layer (Lippmann, 1987).

8.9.4.4 Examples

It is instructive to consider a simple example to see how a neural network is able to develop the solution to a classification problem. Figure 8.21 shows two classes of data, with three points in each, arranged so that they cannot be separated linearly. The network shown in Fig. 8.22 will be used to discriminate the data. The two PEs in the first processing layer are described by activation functions with no thresholds – i.e. $\theta = 0$ in (8.37), while the single output PE has a non-zero threshold in its activation function.

Table 8.3 shows the results of training the network with the backpropagation method of the previous sections, along with the error measure of (8.40) at each step. It can be seen that the network approaches a solution quickly (approximately 50 iterations) but takes more iterations (approximately 250) to converge to a final result.

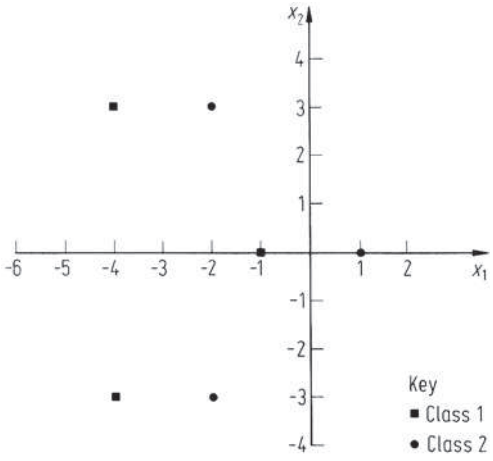
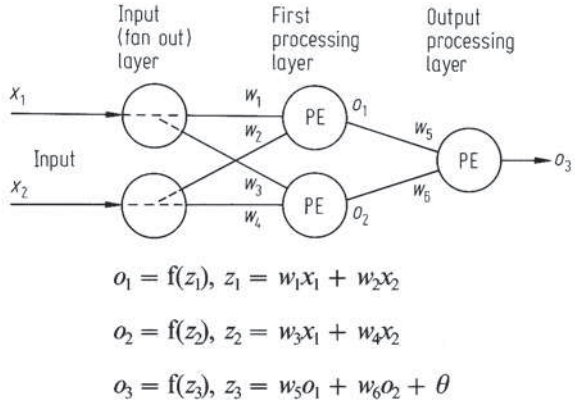


Fig. 8.21. Two-class data set, which is not linearly separable



$$o_1 = \mathrm{f}(z_1),\ z_1 = w_1x_1 + w_2x_2$$

$$o_2 = \mathrm{f}(z_2),\ z_2 = w_3x_1 + w_4x_2$$

$$o_3 = \mathrm{f}(z_3),\ z_3 = w_5o_1 + w_6o_2 + \theta$$

Fig. 8.22. Two processing layer neural network to be applied to the data of Fig. 8.21

Having trained the network it is now possible to understand how it implements a solution to the nonlinear pattern recognition problem. The arguments of the activation functions of the PEs in the first processing layer each define a straight line (hyperplane in general) in the pattern space. Using the result at 250 iterations, these are:

$$2.901x_1 - 2.976x_2 = 0$$
$$2.902x_1 + 2.977x_2 = 0$$

which are shown plotted in Fig. 8.23. An individual line goes some way towards separating the data but cannot accomplish the task fully. It is now important to consider how the output PE operates on the outputs of the first layer PEs to complete the discrimination of the two classes. For pattern points lying exactly on one of the above lines, the output of the respective PE will be 0.5, given that the activation function of (8.38) has been used. However, for patterns a little distance away from

Table 8.3. Training the network of Fig. 8.22

Iteration	w_1	w_2	w_3	w_4	w_5	w_6	θ	Error
0*	0.050	0.100	0.300	0.150	1.000	0.500	−0.500	0.461
1	0.375	0.051	0.418	0.121	0.951	0.520	−0.621	0.424
2	0.450	0.038	0.455	0.118	1.053	0.625	−0.518	0.408
3	0.528	0.025	0.504	0.113	1.119	0.690	−0.522	0.410
4	0.575	0.016	0.541	0.113	1.182	0.752	−0.528	0.395
5	0.606	0.007	0.570	0.117	1.240	0.909	−0.541	0.391
10	0.642	−0.072	0.641	0.196	1.464	1.034	−0.632	0.378
20	0.940	−0.811	0.950	0.882	1.841	1.500	−0.965	0.279
30	1.603	−1.572	1.571	1.576	2.413	2.235	−1.339	0.135
50	2.224	−2.215	2.213	2.216	3.302	3.259	−1.771	0.040
100	2.670	−2.676	2.670	2.677	4.198	4.192	−2.192	0.010
150	2.810	−2.834	2.810	2.835	4.529	4.527	−2.352	0.007
200	2.872	−2.919	2.872	2.920	4.693	4.692	−2.438	0.006
250	2.901	−2.976	2.902	2.977	4.785	4.784	−2.493	0.005

* arbitrary initial set of weights and θ

Table 8.4. Response of the output layer PE

o_1	o_2	o_3
0	0	$0.076 \approx 0$
0	1	$0.908 \approx 1$
1	0	$0.908 \approx 1$
1	1	$0.999 \approx 1$

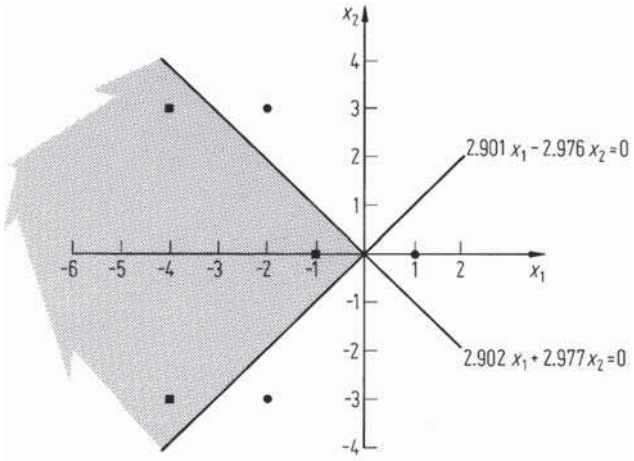


Fig. 8.23. Neural network solution for the data of Fig. 8.21

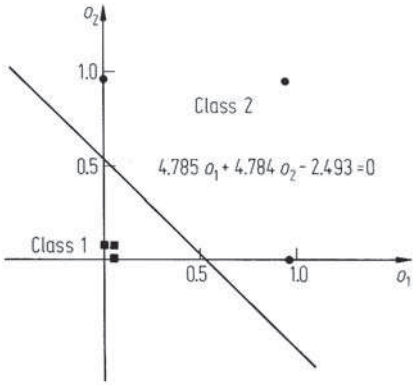


Fig. 8.24. Illustration of how the first processing layer PEs transform the input data into a linearly separable set, which is then discriminated by the output layer hyperplane

those lines the output of the first layer PEs will be close to 0 or 1 depending on which side of the hyperplane they lie. We can therefore regard the pattern space as being divided into two regions – 0 and 1 – by a particular hyperplane. Using these extreme values, Table 8.4 shows the possible responses of the output layer PE for patterns lying somewhere in the pattern space. As seen, for this example the output PE functions in the nature of a logical OR operation; patterns that lie on the 1 side of EITHER input PE hyperplane are labelled as belonging to one class, while those that lie on the 0 side of both hyperplanes will be labelled as belonging to the other class. Thus patterns which lie in the shaded region shown in Fig. 8.23 will generate a 0 at the output of the network and thus will be labelled as belonging to class 1, while patterns in the unshaded regions will generate a 1 response and thus will be labelled as belonging to class 2. Although this exercise is based only on two classes of data, similar functionality of the various PEs in a network can, in principle, be identified. The input PEs will always set up hyperplane divisions of the data and the later PEs will operate on the results of those simple discriminations.

An alternative way of considering how the network determines a solution is to regard the first processing layer PEs as transforming the data in such a way that later PEs (in this example only one) can exercise linear discrimination. Figure 8.24 shows a plot of the outputs of the first layer PEs when fed with the training data of Fig. 8.21. As observed, after transformation, the data is linearly separable. The hyperplane shown is that generated by the argument of the activation function of the output layer PE.

To illustrate how the network of Fig. 8.22 functions on unseen (i.e. testing set) data, Table 8.5 shows its response to the testing patterns indicated in Fig. 8.25. The class decision for a pattern is made by rounding the output PE response to 0 or 1 as appropriate. As noted, for this simple example, all patterns are correctly classified.

Benediktsson, Swain and Esroy (1990) have demonstrated the application of a neural network approach to classification in remote sensing, obtaining classification accuracies as high as 95% on training data although only as high as 52% when the network was applied to a test data set. It is suggested that the training data may not have been fully representative of the image. This is an important issue with neural

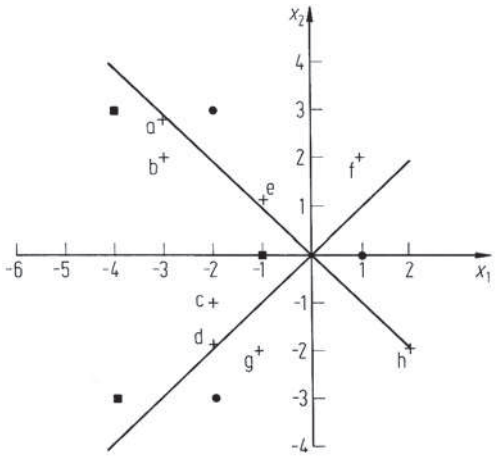


Fig. 8.25. Location of test data, indicated by the lettered crosses

Table 8.5. Performance of the network on the test data

Pattern	x_1	x_2	z_1	o_1	z_2	o_2	z_3	o_3	Class
a	−3.0	2.8	−17.036	0.000	− 0.370	0.408	−0.539	0.368	1
b	−3.0	2.0	−14.655	0.000	− 2.752	0.056	−2.206	0.099	1
c	−2.0	−1.0	− 2.826	0.056	− 8.781	0.000	−2.224	0.098	1
d	−2.0	−1.94	− 0.029	0.493	−11.579	0.000	−0.135	0.466	1
e	−1.0	1.1	− 6.175	0.002	0.373	0.592	0.350	0.587	2
f	1.0	2.0	− 3.051	0.045	8.856	1.000	2.506	0.925	2
g	−1.0	−2.0	3.051	0.955	− 8.856	0.000	2.077	0.889	2
h	2.0	−2.0	11.754	1.000	− 0.150	0.463	4.505	0.989	2

nets, more so than with statistical classification methods such as maximum likelihood, since the parameters in the statistical approach are *estimates* of statistics, and are not strongly affected by outlying training samples. The work of Benediktsson also illustrates, to an extent, the dependence of performance on the network architecture chosen.

Hepner (1990) has also used a neural network to perform a classification; in addition to the spectral properties of a pixel, however, he included the spectral measurements of the 3×3 neighbourhood in order to allow spatial context to influence the labelling. Although quantitative accuracies are not given, Hepner is of the view that the results are better than when using a maximum likelihood classifier trained on spectral data only. Lippmann (1987) and Pao (1989) are good general references to consult for a wider treatment of neural network theory than has been given here, including other training methods. Both demonstrate also how neural networks can be used in unsupervised as well as supervised classification. Paola and Schowengerdt (1995a,b) provide a comprehensive review of the use of the multilayer Perception in remote sensing.

A range of neural network tools is available in MATLAB (1984–2004).