CHAPTER

Linear Classifiers

3

3.1 INTRODUCTION

Our major concern in Chapter 2 was to design classifiers based on probability density or probability functions. In some cases, we saw that the resulting classifiers were equivalent to a set of linear discriminant functions. In this chapter, we will focus on the design of linear classifiers, *regardless of the underlying distributions describing the training data*. The major advantage of linear classifiers is their simplicity and computational attractiveness. The chapter starts with the assumption that *all* feature vectors from the available classes can be classified correctly using a linear classifier, and we will develop techniques for the computation of the corresponding linear functions. In the sequel we will focus on a more general problem, in which a linear classifier cannot correctly classify all vectors, yet we will seek ways to design an *optimal linear* classifier by adopting an appropriate optimality criterion.

3.2 LINEAR DISCRIMINANT FUNCTIONS AND DECISION HYPERPLANES

Let us once more focus on the two-class case and consider linear discriminant functions. Then the respective decision hypersurface in the l-dimensional feature space is a hyperplane, that is

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0 \tag{3.1}$$

where $\boldsymbol{w} = [w_1, w_2, \ldots, w_l]^T$ is known as the *weight vector* and w_0 as the *threshold*. If $\boldsymbol{x}_1, \boldsymbol{x}_2$ are two points on the decision hyperplane, then the following is valid

$$\begin{aligned} 0 = \boldsymbol{w}^T \boldsymbol{x}_1 + w_0 = \boldsymbol{w}^T \boldsymbol{x}_2 + w_0 \quad \Rightarrow \\ \boldsymbol{w}^T (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0 \end{aligned} \tag{3.2}$$

Since the difference vector $\boldsymbol{x}_1 - \boldsymbol{x}_2$ obviously lies on the decision hyperplane (for any $\boldsymbol{x}_1, \boldsymbol{x}_2$), it is apparent from Eq. (3.2) that the vector $\boldsymbol{w}$ is *orthogonal* to the decision hyperplane.

Figure 3.1 shows the corresponding geometry (for $w_1 > 0, w_2 > 0, w_0 < 0$). Recalling our high school math, it is easy to see that the quantities entering in the figure are given by

$$d = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}} \tag{3.3}$$

and

$$z = \frac{|g(\boldsymbol{x})|}{\sqrt{w_1^2 + w_2^2}} \tag{3.4}$$

In other words, $|g(\boldsymbol{x})|$ is a measure of the Euclidean distance of the point $\boldsymbol{x}$ from the decision hyperplane. On one side of the plane $g(\boldsymbol{x})$ takes positive values and on the other negative. In the special case that $w_0 = 0$, the hyperplane passes through the origin.

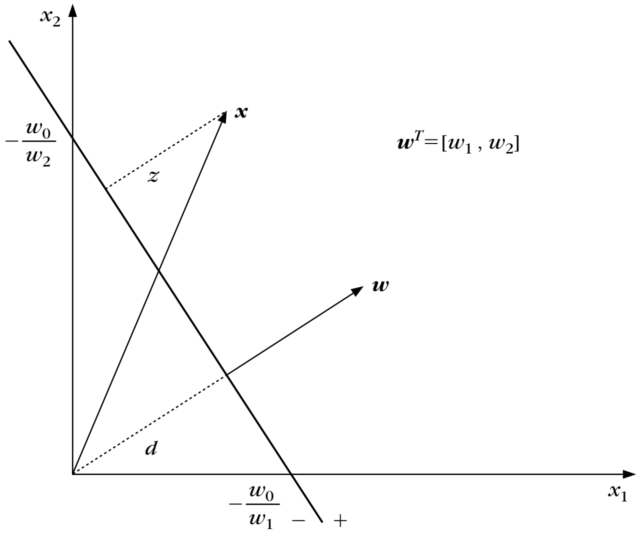

FIGURE 3.1

Geometry for the decision line. On one side of the line it is $g(\boldsymbol{x}) > 0(+)$ and on the other $g(\boldsymbol{x}) < 0(-)$.

3.3 THE PERCEPTRON ALGORITHM

Our major concern now is to compute the unknown parameters $w_i, i = 0, \ldots, l$, defining the decision hyperplane. In this section, we assume that the two classes ω_1, ω_2 are *linearly separable*. In other words, we assume that *there exists* a hyperplane, defined by $\boldsymbol{w}^{*T}\boldsymbol{x} = 0$, such that

$$\begin{aligned} \boldsymbol{w}^{*T}\boldsymbol{x} > 0 \quad \forall \boldsymbol{x} \in \omega_1 \\ \boldsymbol{w}^{*T}\boldsymbol{x} < 0 \quad \forall \boldsymbol{x} \in \omega_2 \end{aligned} \tag{3.5}$$

The formulation above also covers the case of a hyperplane not crossing the origin, that is, $\boldsymbol{w}^{*T}\boldsymbol{x} + w_0^* = 0$, since this can be brought into the previous formulation by defining the extended $(l+1)$-dimensional vectors $\boldsymbol{x}' \equiv [\boldsymbol{x}^T, 1]^T$, $\boldsymbol{w}' \equiv [\boldsymbol{w}^{*T}, w_0^*]^T$. Then $\boldsymbol{w}^{*T}\boldsymbol{x} + w_0^* = \boldsymbol{w}'^T\boldsymbol{x}'$.

We will approach the problem as a typical optimization task (Appendix C). Thus we need to adopt (a) an appropriate cost function and (b) an algorithmic scheme to optimize it. To this end, we choose the *perceptron cost* defined as

$$J(\boldsymbol{w}) = \sum_{\boldsymbol{x} \in Y} (\delta_x \boldsymbol{w}^T \boldsymbol{x}) \tag{3.6}$$

where Y is the subset of the training vectors, which are misclassified by the hyperplane defined by the weight vector $\boldsymbol{w}$. The variable δ_x is chosen so that $\delta_x = -1$ if $\boldsymbol{x} \in \omega_1$ and $\delta_x = +1$ if $\boldsymbol{x} \in \omega_2$. Obviously, the sum in (3.6) is always positive, and it becomes zero when Y becomes the empty set, that is, if there are not misclassified vectors $\boldsymbol{x}$. Indeed, if $\boldsymbol{x} \in \omega_1$ and it is misclassified, then $\boldsymbol{w}^T\boldsymbol{x} < 0$ and $\delta_x < 0$, and the product is positive. The result is the same for vectors originating from class ω_2. When the cost function takes its minimum value, 0, a solution has been obtained, since all training feature vectors are correctly classified.

The perceptron cost function in (3.6) is *continuous and piecewise linear*. Indeed, if we change the weight vector smoothly, the cost $J(\boldsymbol{w})$ changes linearly until the point at which there is a change in the number of misclassified vectors (Problem 3.1). At these points the gradient is not defined, and the gradient function is discontinuous.

To derive the algorithm for the iterative minimization of the cost function, we will adopt an iterative scheme in the spirit of the *gradient descent* method (Appendix C), that is,

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \rho_t \left.\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}(t)} \tag{3.7}$$

where $\boldsymbol{w}(t)$ is the weight vector estimate at the tth iteration step and ρ_t is a sequence of positive real numbers. However, we must be careful here. This is not defined at the points of discontinuity. From the definition in (3.6), and at the points where this is valid, we get

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x} \tag{3.8}$$

Substituting (3.8) into (3.7), we obtain

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \rho_t \sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x} \tag{3.9}$$

The algorithm is known as the *perceptron algorithm* and is quite simple in its structure. *Note that Eq. (3.9) is defined at all points*. The algorithm is initialized from an arbitrary weight vector $\boldsymbol{w}(0)$, and the correction vector $\sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x}$ is formed using the misclassified features. The weight vector is then corrected according to the preceding rule. This is repeated until the algorithm converges to a solution, that is, all features are correctly classified. A pseudocode for the perceptron algorithm is given below.

The Perceptron Algorithm

```
■ Choose $\boldsymbol{w}(0)$ randomly
■ Choose $\rho_0$
■ $t = 0$
■ Repeat
    • $Y = \emptyset$
    • For $i = 1$ to $N$
        ○ If $\delta_{x_i} \boldsymbol{w}(t)^T \boldsymbol{x}_i \geq 0$ then $Y = Y \cup \{\boldsymbol{x}_i\}$
    • End {For}
    • $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \rho_t \sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x}$
    • Adjust $\rho_t$
    • $t = t + 1$
■ Until $Y = \emptyset$
```

Figure 3.2 provides a geometric interpretation of the algorithm. It has been assumed that at step t there is only one misclassified sample, $\boldsymbol{x}$, and $\rho_t = 1$. The perceptron algorithm corrects the weight vector in the direction of $\boldsymbol{x}$. Its effect is to turn the corresponding hyperplane so that $\boldsymbol{x}$ is classified in the correct class ω_1. Note that in order to achieve this, it may take more than one iteration step, depending on the value(s) of ρ_t. No doubt, this sequence is critical for the convergence. We will now show that the perceptron algorithm converges

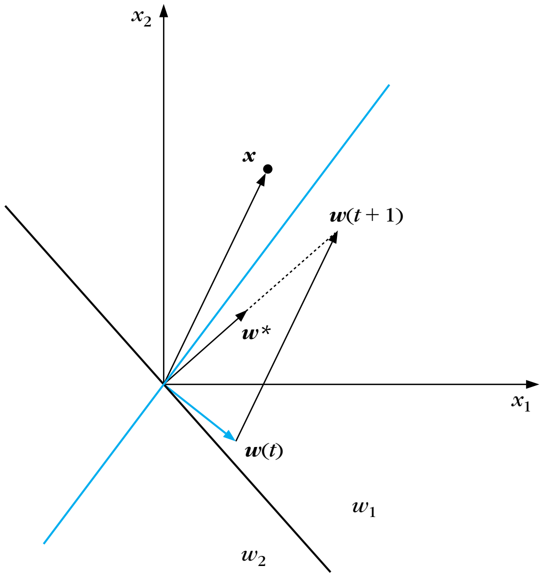

FIGURE 3.2

Geometric interpretation of the perceptron algorithm. The update of the weight vector is in the direction of $\boldsymbol{x}$ in order to turn the decision hyperplane to include $\boldsymbol{x}$ in the correct class.

to a solution in a *finite number of iteration steps*, provided that the sequence ρ_t is properly chosen. The solution is not unique, because there are more than one hyperplanes separating two linearly separable classes. The convergence proof is necessary because the algorithm is not a true gradient descent algorithm and the general tools for the convergence of gradient descent schemes cannot be applied.

Proof of the Perceptron Algorithm Convergence

Let α be a positive real number and $\boldsymbol{w}^*$ a solution. Then from (3.9) we have

$$\boldsymbol{w}(t+1) - \alpha\boldsymbol{w}^* = \boldsymbol{w}(t) - \alpha\boldsymbol{w}^* - \rho_t \sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x} \tag{3.10}$$

Squaring the Euclidean norm of both sides results in

$$\|\boldsymbol{w}(t+1) - \alpha\boldsymbol{w}^*\|^2 = \|\boldsymbol{w}(t) - \alpha\boldsymbol{w}^*\|^2 + \rho_t^2 \|\sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x}\|^2$$
$$- 2\rho_t \sum_{\boldsymbol{x}\in Y} \delta_x (\boldsymbol{w}(t) - \alpha\boldsymbol{w}^*)^T \boldsymbol{x} \tag{3.11}$$

But $-\sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{w}^T(t)\,\boldsymbol{x} < 0$. Hence

$$\|\boldsymbol{w}(t+1) - \alpha\boldsymbol{w}^*\|^2 \leq \|\boldsymbol{w}(t) - \alpha\boldsymbol{w}^*\|^2 + \rho_t^2 \|\sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{x}\|^2 + 2\rho_t\alpha \sum_{\boldsymbol{x}\in Y} \delta_x \boldsymbol{w}^{*T}\boldsymbol{x} \qquad (3.12)$$

Define

$$\beta^2 = \max_{\widetilde{Y}\subseteq\omega_1\cup\omega_2} \|\sum_{\boldsymbol{x}\in\widetilde{Y}} \delta_x \boldsymbol{x}\|^2 \qquad (3.13)$$

That is, β^2 is the maximum value that the involved vector norm can take by considering *all* possible (nonempty) subsets of the available training feature vectors. Similarly, let

$$\gamma = \max_{\widetilde{Y}\subseteq\omega_1\cup\omega_2} \sum_{\boldsymbol{x}\in\widetilde{Y}} \delta_x \boldsymbol{w}^{*T}\boldsymbol{x} \qquad (3.14)$$

Recall that the summation in this equation is negative; thus, its maximum value over all possible subsets of $\boldsymbol{x}$'s will also be a negative number. Hence, (3.12) can now be written as

$$\|\boldsymbol{w}(t+1) - \alpha\boldsymbol{w}^*\|^2 \leq \|\boldsymbol{w}(t) - \alpha\boldsymbol{w}^*\|^2 + \rho_t^2\beta^2 - 2\rho_t\alpha|\gamma| \qquad (3.15)$$

Choose $\alpha = \frac{\beta^2}{2|\gamma|}$ and apply (3.15) successively for steps $t, t-1, \ldots, 0$. Then

$$\|\boldsymbol{w}(t+1) - \alpha\boldsymbol{w}^*\|^2 \leq \|\boldsymbol{w}(0) - \alpha\boldsymbol{w}^*\|^2 + \beta^2\left(\sum_{k=0}^{t}\rho_k^2 - \sum_{k=0}^{t}\rho_k\right) \qquad (3.16)$$

If the sequence ρ_t is chosen to satisfy the following two conditions:

$$\lim_{t\to\infty}\sum_{k=0}^{t}\rho_k = \infty \qquad (3.17)$$

$$\lim_{t\to\infty}\sum_{k=0}^{t}\rho_k^2 < \infty \qquad (3.18)$$

then there will be a constant t_0 such that the right-hand side of (3.16) becomes nonpositive. Thus

$$0 \leq \|\boldsymbol{w}(t_0+1) - \alpha\boldsymbol{w}^*\| \leq 0 \qquad (3.19)$$

or

$$w(t_0 + 1) = \alpha w^* \tag{3.20}$$

That is, the algorithm converges to a solution in a finite number of steps. An example of a sequence satisfying conditions (3.17), (3.18) is $\rho_t = c/t$, where c is a constant. In other words, the corrections become increasingly small. What these conditions basically state is that ρ_t should vanish as $t \to \infty$ [Eq. (3.18)] but on the other hand should not go to zero very fast [Eq. (3.17)]. Following arguments similar to those used before, it is easy to show that the algorithm also converges for constant $\rho_t = \rho$, provided ρ is properly bounded (Problem 3.2). In practice, the proper choice of the sequence ρ_t is vital for the convergence speed of the algorithm.

Example 3.1

Figure 3.3 shows the dashed line

$$x_1 + x_2 - 0.5 = 0$$

corresponding to the weight vector $[1, 1, -0.5]^T$, which has been computed from the latest iteration step of the perceptron algorithm (3.9), with $\rho_t = \rho = 0.7$. The line classifies correctly

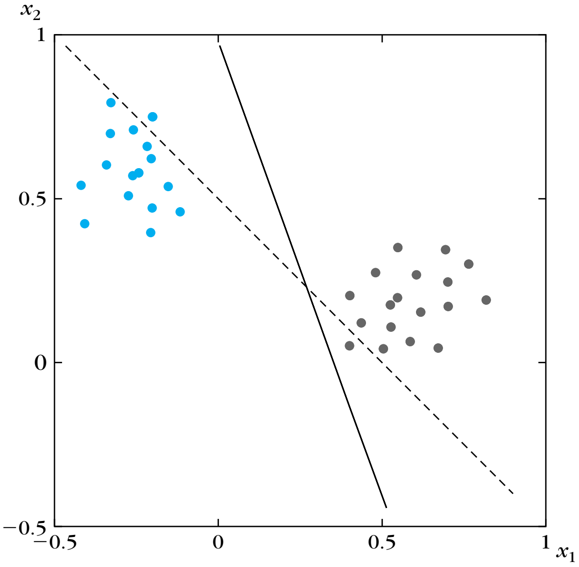

FIGURE 3.3

An example of the perceptron algorithm. After the update of the weight vector, the hyperplane is turned from its initial location (dotted line) to the new one (full line), and all points are correctly classified.

all the vectors except $[0.4, 0.05]^T$ and $[-0.20, 0.75]^T$. According to the algorithm, the next weight vector will be

$$\boldsymbol{w}(t+1) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix} - 0.7(-1) \begin{bmatrix} 0.4 \\ 0.05 \\ 1 \end{bmatrix} - 0.7(+1) \begin{bmatrix} -0.2 \\ 0.75 \\ 1 \end{bmatrix}$$

or

$$\boldsymbol{w}(t+1) = \begin{bmatrix} 1.42 \\ 0.51 \\ -0.5 \end{bmatrix}$$

The resulting new (solid) line $1.42x_1 + 0.51x_2 - 0.5 = 0$ classifies all vectors correctly, and the algorithm is terminated.

Variants of the Perceptron Algorithm

The algorithm we have presented is just one form of a number of variants that have been proposed for the training of a linear classifier in the case of linearly separable classes. We will now state another simpler and also popular form. The N training vectors enter the algorithm cyclically, one after the other. If the algorithm has not converged after the presentation of all the samples once, then the procedure keeps repeating until convergence is achieved—that is, when all training samples have been classified correctly. Let $\boldsymbol{w}(t)$ be the weight vector estimate and $\boldsymbol{x}_{(t)}$ the corresponding feature vector, presented at the tth iteration step. The algorithm is stated as follows:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{x}_{(t)} \in \omega_1 \text{ and } \boldsymbol{w}^T(t)\boldsymbol{x}_{(t)} \leq 0 \tag{3.21}$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{x}_{(t)} \in \omega_2 \text{ and } \boldsymbol{w}^T(t)\boldsymbol{x}_{(t)} \geq 0 \tag{3.22}$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) \quad \text{otherwise} \tag{3.23}$$

In other words, if the current training sample is classified correctly, no action is taken. Otherwise, if the sample is misclassified, the weight vector is corrected by adding (subtracting) an amount proportional to $\boldsymbol{x}_{(t)}$. The algorithm belongs to a more general algorithmic family known as *reward and punishment* schemes. If the classification is correct, the reward is that no action is taken. If the current vector is misclassified, the punishment is the cost of correction. It can be shown that this form of the perceptron algorithm also converges in a finite number of iteration steps (Problem 3.3).

The perceptron algorithm was originally proposed by Rosenblatt in the late 1950s. The algorithm was developed for training the *perceptron*, the basic unit used for modeling neurons of the brain. This was considered central in developing powerful models for machine learning [Rose 58, Min 88].

Example 3.2

Figure 3.4 shows four points in the two-dimensional space. Points $(-1,0),(0,1)$ belong to class ω_1, and points $(0,-1),(1,0)$ belong to class ω_2. The goal of this example is to design a linear classifier using the perceptron algorithm in its reward and punishment form. The parameter ρ is set equal to one, and the initial weight vector is chosen as $\boldsymbol{w}(0) = [0,0,0]^T$ in the extended three-dimensional space. According to (3.21)–(3.23), the following computations are in order:

Step 1.

$$\boldsymbol{w}^T(0)\begin{bmatrix}-1\\0\\1\end{bmatrix} = 0, \quad \boldsymbol{w}(1) = \boldsymbol{w}(0) + \begin{bmatrix}-1\\0\\1\end{bmatrix} = \begin{bmatrix}-1\\0\\1\end{bmatrix}$$

Step 2.

$$\boldsymbol{w}^T(1)\begin{bmatrix}0\\1\\1\end{bmatrix} = 1 > 0, \quad \boldsymbol{w}(2) = \boldsymbol{w}(1)$$

Step 3.

$$\boldsymbol{w}^T(2)\begin{bmatrix}0\\-1\\1\end{bmatrix} = 1 > 0, \quad \boldsymbol{w}(3) = \boldsymbol{w}(2) - \begin{bmatrix}0\\-1\\1\end{bmatrix} = \begin{bmatrix}-1\\1\\0\end{bmatrix}$$

Step 4.

$$\boldsymbol{w}^T(3)\begin{bmatrix}1\\0\\1\end{bmatrix} = -1 < 0, \quad \boldsymbol{w}(4) = \boldsymbol{w}(3)$$

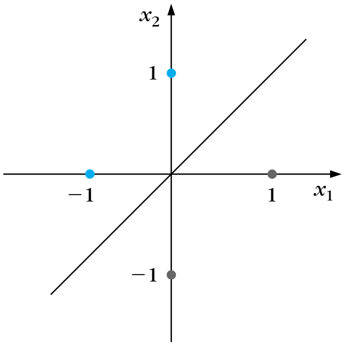

FIGURE 3.4

The setup for Example 3.2. The line $x_1 = x_2$ is the resulting solution.

Step 5.

$$\boldsymbol{w}^T(4)\begin{bmatrix}-1\\0\\1\end{bmatrix} = 1 > 0, \quad \boldsymbol{w}(5) = \boldsymbol{w}(4)$$

Step 6.

$$\boldsymbol{w}^T(5)\begin{bmatrix}0\\1\\1\end{bmatrix} = 1 > 0, \quad \boldsymbol{w}(6) = \boldsymbol{w}(5)$$

Step 7.

$$\boldsymbol{w}^T(6)\begin{bmatrix}0\\-1\\1\end{bmatrix} = -1 < 0, \quad \boldsymbol{w}(7) = \boldsymbol{w}(6)$$

Since for four consecutive steps no correction is needed, all points are correctly classified and the algorithm terminates. The solution is $\boldsymbol{w} = [-1, 1, 0]^T$. That is, the resulting linear classifier is $-x_1 + x_2 = 0$, and it is the line passing through the origin shown in Figure 3.4.

The Perceptron

Once the perceptron algorithm has converged to a weight vector $\boldsymbol{w}$ and a threshold w_0, our next goal is the classification of an unknown feature vector to either of the two classes. Classification is achieved via the simple rule

$$\begin{aligned}&\text{If} \quad \boldsymbol{w}^T\boldsymbol{x} + w_0 > 0 \quad \text{assign } \boldsymbol{x} \text{ to } \omega_1\\ &\text{If} \quad \boldsymbol{w}^T\boldsymbol{x} + w_0 < 0 \quad \text{assign } \boldsymbol{x} \text{ to } \omega_2\end{aligned} \tag{3.24}$$

A basic network unit that implements the operation is shown in Figure 3.5a.

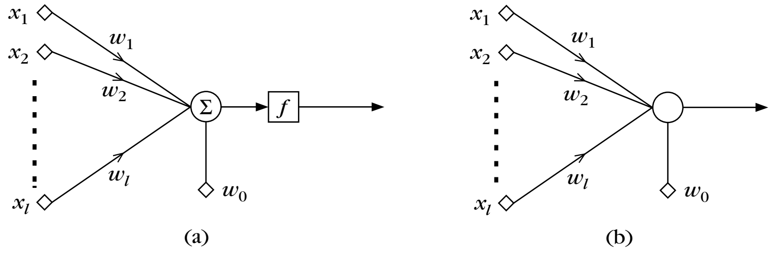

FIGURE 3.5

The basic perceptron model. (a) A linear combiner is followed by the activation function. (b) The combiner and the activation function are merged together.

The elements of the feature vector $x_1, x_2, \dots, x_l$ are applied to the *input nodes* of the network. Then each one is multiplied by the corresponding weights $w_i, i = 1, 2, \dots, l$. These are known as *synaptic weights* or simply *synapses*. The products are summed up together with the *threshold* value w_0. The result then goes through a nonlinear device, which implements the so-called *activation function*. A common choice is a hard limiter; that is, $f(\cdot)$ is the step function [$f(x) = -1$ if $x < 0$ and $f(x) = 1$ if $x > 0$]. The corresponding feature vector is classified in one of the classes depending on the sign of the output. Besides $+1$ and -1, other values (class labels) for the hard limiter are also possible. Another popular choice is 1 and 0, and it is achieved by choosing the two levels of the step function appropriately. This basic network is known as a *perceptron* or *neuron*. Perceptrons are simple examples of the so-called *learning machines*—that is, structures whose free parameters are updated by a *learning algorithm*, such as the perceptron algorithm, in order to "learn" a specific task, based on a set of training data. Later on we will use the perceptron as the basic building element for more complex learning networks. Figure 3.5b is a simplified graph of the neuron where the summer and nonlinear device have been merged for notational simplification. Sometimes a neuron with a hard limiter device is referred to as a McCulloch–Pitts neuron. Other types of neurons will be considered in Chapter 4.

The Pocket Algorithm

A basic requirement for the convergence of the perceptron algorithm is the linear separability of the classes. If this is not true, as is usually the case in practice, the perceptron algorithm does not converge. A variant of the perceptron algorithm was suggested in [Gal 90] that converges to an optimal solution even if the linear separability condition is not fulfilled. The algorithm is known as the *pocket* algorithm and consists of the following two steps

- Initialize the weight vector $\boldsymbol{w}(0)$ randomly. Define a stored (in the pocket!) vector $\boldsymbol{w}_s$. Set a history counter h_s of the $\boldsymbol{w}_s$ to zero.
- At the tth iteration step compute the update $\boldsymbol{w}(t+1)$, according to the perceptron rule. Use the updated weight vector to test the number h of training vectors that are classified correctly. If $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(t+1)$ and h_s with h. Continue the iterations.

It can be shown that this algorithm converges with probability one to the optimal solution, that is, the one that produces the minimum number of misclassifications [Gal 90, Muse 97]. Other related algorithms that find reasonably good solutions when the classes are not linearly separable are the *thermal perceptron algorithm* [Frea 92], the *loss minimization algorithm* [Hryc 92], and the barycentric correction procedure [Poul 95].

Kesler's Construction

So far we have dealt with the two-class case. The generalization to an *M*-class task is straightforward. A linear discriminant function $\boldsymbol{w}_i, i = 1, 2, \dots, M$, is defined for each of the classes. A feature vector $\boldsymbol{x}$ (in the $(l+1)$-dimensional space to account for the threshold) is classified in class ω_i if

$$\boldsymbol{w}_i^T \boldsymbol{x} > \boldsymbol{w}_j^T \boldsymbol{x}, \quad \forall j \neq i \tag{3.25}$$

This condition leads to the so-called Kesler's construction. For each of the training vectors from class $\omega_i, i = 1, 2, \ldots, M$, we construct $M-1$ vectors $\boldsymbol{x}_{ij} = [\boldsymbol{0}^T, \boldsymbol{0}^T, \ldots, \boldsymbol{x}^T, \ldots, -\boldsymbol{x}^T, \ldots, \boldsymbol{0}^T]^T$ of dimension $(l+1)M \times 1$. That is, they are block vectors having zeros everywhere except at the ith and jth block positions, where they have $\boldsymbol{x}$ and $-\boldsymbol{x}$, respectively, for $j \neq i$. We also construct the block vector $\boldsymbol{w} = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_M^T]^T$. If $\boldsymbol{x} \in \omega_i$, this imposes the requirement that $\boldsymbol{w}^T \boldsymbol{x}_{ij} > 0, \forall j = 1, 2, \ldots, M, j \neq i$. The task now is to design a linear classifier, in the extended $(l+1)M$-dimensional space, so that each of the $(M-1)N$ training vectors lies in its positive side. The perceptron algorithm will have no difficulty in solving this problem for us, provided that such a solution is possible—that is, if all the training vectors can be correctly classified using a set of linear discriminant functions.

Example 3.3

Let us consider a three-class problem in the two-dimensional space. The training vectors for each of the classes are the following:

$$\omega_1\colon [1, 1]^T, [2, 2]^T, [2, 1]^T$$

$$\omega_2\colon [1, -1]^T, [1, -2]^T, [2, -2]^T$$

$$\omega_3\colon [-1, 1]^T, [-1, 2]^T, [-2, 1]^T$$

This is obviously a linearly separable problem, since the vectors of different classes lie in different quadrants.

To compute the linear discriminant functions, we first extend the vectors to the three-dimensional space, and then we use Kesler's construction. For example,

For $[1, 1]^T$ we get $[1, 1, 1, -1, -1, -1, 0, 0, 0]^T$ and
$[1, 1, 1, 0, 0, 0, -1, -1, -1]^T$
For $[1, -2]^T$ we get $[-1, 2, -1, 1, -2, 1, 0, 0, 0]^T$ and
$[0, 0, 0, 1, -2, 1, -1, 2, -1]^T$
For $[-2, 1]^T$ we get $[2, -1, -1, 0, 0, 0, -2, 1, 1]^T$ and
$[0, 0, 0, 2, -1, -1, -2, 1, 1]^T$

Similarly, we obtain the other twelve vectors. To obtain the corresponding weight vectors

$$\boldsymbol{w}_1 = [w_{11}, w_{12}, w_{10}]^T$$

$$\boldsymbol{w}_2 = [w_{21}, w_{22}, w_{20}]^T$$

$$\boldsymbol{w}_3 = [w_{31}, w_{32}, w_{30}]^T$$

we can run the perceptron algorithm by requiring $\boldsymbol{w}^T\boldsymbol{x} > 0$, $\boldsymbol{w} = [\boldsymbol{w}_1^T, \boldsymbol{w}_2^T, \boldsymbol{w}_3^T]^T$, for each of the eighteen 9-dimensional vectors. That is, we require all the vectors to lie on the same side of the decision hyperplane. The initial vector of the algorithm $\boldsymbol{w}(0)$ is computed using the uniform pseudorandom sequence generator in $[0, 1]$. The learning sequence ρ_t was chosen to be constant and equal to 0.5. The algorithm converges after four iterations and gives

$$\boldsymbol{w}_1 = [5.13, 3.60, 1.00]^T$$

$$\boldsymbol{w}_2 = [-0.05, -3.16, -0.41]^T$$

$$\boldsymbol{w}_3 = [-3.84, 1.28, 0.69]^T$$

3.4 LEAST SQUARES METHODS

As we have already pointed out, the attractiveness of linear classifiers lies in their simplicity. Thus, in many cases, although we know that the classes are not linearly separable, we still wish to adopt a linear classifier, despite the fact that this will lead to *suboptimal* performance from the classification error probability point of view. The goal now is to compute the corresponding weight vector under a suitable optimality criterion. The least squares methods are familiar to us, in one way or another, from our early college courses. Let us then build upon them.

3.4.1 Mean Square Error Estimation

Let us once more focus on the two-class problem. In the previous section, we saw that the perceptron output was ±1, depending on the class ownership of $\boldsymbol{x}$. Since the classes were linearly separable, these outputs were correct for all the training feature vectors, after, of course, the perceptron algorithm's convergence. In this section we will attempt to design a linear classifier so that its *desired* output is again ±1, depending on the class ownership of the input vector. However, we will have to live with errors; that is, the true output will not always be equal to the desired one. Given a vector $\boldsymbol{x}$, the output of the classifier will be $\boldsymbol{w}^T\boldsymbol{x}$ (thresholds can be accommodated by vector extensions). The desired output will be denoted as $y(\boldsymbol{x}) \equiv y = \pm 1$. The weight vector will be computed so as to minimize the mean square error (MSE) between the desired and true outputs, that is,

$$J(\boldsymbol{w}) = E[|y - \boldsymbol{x}^T\boldsymbol{w}|^2] \tag{3.26}$$

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w}) \tag{3.27}$$

The reader can easily check that $J(\boldsymbol{w})$ is equal to

$$J(\boldsymbol{w}) = P(\omega_1)\int (1 - \boldsymbol{x}^T\boldsymbol{w})^2 p(\boldsymbol{x}|\omega_1)\,d\boldsymbol{x} + P(\omega_2)\int (1 + \boldsymbol{x}^T\boldsymbol{w})^2 p(\boldsymbol{x}|\omega_2)\,d\boldsymbol{x} \tag{3.28}$$

Minimizing (3.27) easily results in

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 2E[\boldsymbol{x}(y - \boldsymbol{x}^T\boldsymbol{w})] = \boldsymbol{0} \tag{3.29}$$

Then

$$\hat{\boldsymbol{w}} = R_x^{-1}E[\boldsymbol{x}y] \tag{3.30}$$

where

$$R_x \equiv E[\boldsymbol{x}\boldsymbol{x}^T] = \begin{bmatrix} E[x_1x_1] & \cdots & E[x_1x_l] \\ E[x_2x_1] & \cdots & E[x_2x_l] \\ \vdots & \vdots & \vdots \\ E[x_lx_1] & \cdots & E[x_lx_l] \end{bmatrix} \tag{3.31}$$

is known as the *correlation* or autocorrelation matrix and is equal to the covariance matrix, introduced in the previous chapter, if the respective mean values are zero. The vector

$$E[\boldsymbol{x}y] = E\left[\begin{bmatrix} x_1y \\ \vdots \\ x_ly \end{bmatrix}\right] \tag{3.32}$$

is known as the *cross-correlation* between the desired output and the (input) feature vectors. Thus, the mean square optimal weight vector results as the solution of a ***linear set of equations***, provided, of course, that the correlation matrix is invertible.

It is interesting to point out that there is a geometrical interpretation of this solution. Random variables can be considered as points in a vector space. It is straightforward to see that the expectation operation $E[xy]$ between two random variables satisfies the properties of the inner product. Indeed, $E[x^2] \geq 0, E[xy] = E[yx], E[x(c_1y + c_2z)] = c_1E[xy] + c_2E[xz]$. In such a vector space $\boldsymbol{w}^T\boldsymbol{x} = w_1x_1 + \cdots + w_lx_l$ is a linear combination of vectors, and thus it lies in the subspace defined by the x_i's.

This is illustrated by an example in Figure 3.6. Then, if we want to approximate y by this linear combination, the resulting error is $y - \boldsymbol{w}^T\boldsymbol{x}$. Equation (3.29) states that the minimum mean square error solution results if the error is orthogonal to each x_i; thus it is *orthogonal* to the vector subspace spanned by $x_i, i = 1, 2, \ldots, l$—in other words, if y is approximated by its *orthogonal projection* on the subspace (Figure 3.6). Equation (3.29) is also known as the *orthogonality condition*.

Multiclass Generalization

In the multiclass case, the task is to design the M linear discriminant functions $g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x}$ according to the MSE criterion. The corresponding desired output responses (i.e., class labels) are chosen so that $y_i = 1$ if $\boldsymbol{x} \in \omega_i$ and $y_i = 0$ otherwise. This is in agreement with the two-class case. Indeed, for such

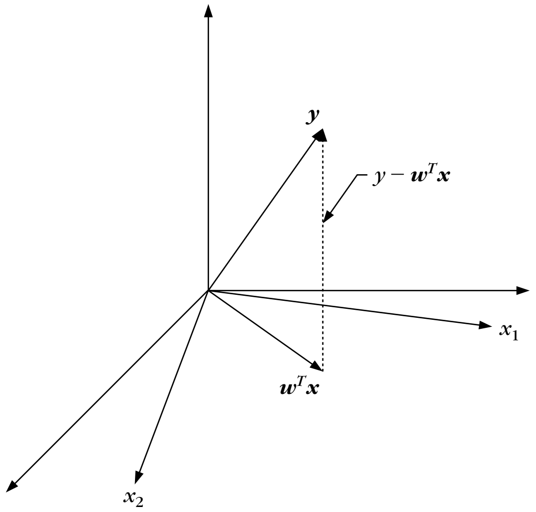

FIGURE 3.6

Interpretation of the MSE estimate as an orthogonal projection on the input vector elements' subspace.

a choice and if $M = 2$, the design of the decision hyperplane $\boldsymbol{w}^T\boldsymbol{x} \equiv (\boldsymbol{w}_1 - \boldsymbol{w}_2)^T\boldsymbol{x}$ corresponds to ±1 desired responses, depending on the respective class ownership.

Let us now define $\boldsymbol{y}^T = [y_1, \ldots, y_M]$, for a given vector $\boldsymbol{x}$, and $W = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_M]$. That is, matrix W has as columns the weight vectors $\boldsymbol{w}_i$. The MSE criterion in (3.27) can now be generalized to minimize the norm of the error vector $\boldsymbol{y} - W^T\boldsymbol{x}$, that is,

$$\hat{W} = \arg\min_W E[\|\boldsymbol{y} - W^T\boldsymbol{x}\|^2] = \arg\min_W E\left[\sum_{i=1}^{M}\left(y_i - \boldsymbol{w}_i^T\boldsymbol{x}\right)^2\right] \tag{3.33}$$

This is equivalent to M MSE independent minimization problems of the (3.27) type, with scalar desired responses. *In other words, in order to design the MSE optimal linear discriminant functions, it suffices to design each one of them so that its desired output is* 1 *for vectors belonging to the corresponding class and* 0 *for all the others*.

3.4.2 Stochastic Approximation and the LMS Algorithm

The solution of (3.30) requires the computation of the correlation matrix and cross-correlation vector. This presupposes knowledge of the underlying distributions, which in general are not known. After all, if they were known, why not then use

Bayesian classifiers? Thus, our major goal now becomes to see if it is possible to solve (3.29) without having this statistical information available. The answer has been provided by Robbins and Monro [Robb 51] in the more general context of stochastic approximation theory. Consider an equation of the form $E[F(\boldsymbol{x}_k, \boldsymbol{w})] = \mathbf{0}$, where $\boldsymbol{x}_k, k = 1, 2, \ldots,$ is a sequence of random vectors from the same distribution, $F(\cdot, \cdot)$ a function, and $\boldsymbol{w}$ the vector of the unknown parameters. Then adopt the iterative scheme

$$\hat{\boldsymbol{w}}(k) = \hat{\boldsymbol{w}}(k-1) + \rho_k F(\boldsymbol{x}_k, \hat{\boldsymbol{w}}(k-1)) \tag{3.34}$$

In other words, the place of the *mean value* (which cannot be computed due to lack of information) is taken by the samples of the random variables resulting from the experiments. It turns out that under mild conditions the iterative scheme converges in probability to the solution $\boldsymbol{w}$ of the original equation, provided that the sequence ρ_k satisfies the two conditions

$$\sum_{k=1}^{\infty} \rho_k \to \infty \tag{3.35}$$

$$\sum_{k=1}^{\infty} \rho_k^2 < \infty \tag{3.36}$$

and which implies that

$$\rho_k \to 0 \tag{3.37}$$

That is,

$$\lim_{k\to\infty} \text{prob}\{\hat{\boldsymbol{w}}(k) = \boldsymbol{w}\} = 1 \tag{3.38}$$

The stronger, in the mean square sense, convergence is also true

$$\lim_{k\to\infty} E[\|\hat{\boldsymbol{w}}(k) - \boldsymbol{w}\|^2] = 0 \tag{3.39}$$

Conditions (3.35), (3.36) have already been met before and guarantee that the corrections of the estimates in the iterations tend to zero. Thus, for large values of k (in theory at infinity) iterations freeze. However, this must not happen too early (first condition) to make sure that the iterations do not stop away from the solution. The second condition guarantees that the accumulated noise, due to the stochastic nature of the variables, remains finite and the algorithm can cope with it [Fuku 90]. The proof is beyond the scope of the present text. However, we will demonstrate its validity via an example.

Let us consider the simple equation $E[x_k - w] = 0$. For $\rho_k = 1/k$ the iteration becomes

$$\hat{w}(k) = \hat{w}(k-1) + \frac{1}{k}[x_k - \hat{w}(k-1)] = \frac{(k-1)}{k}\hat{w}(k-1) + \frac{1}{k}x_k$$

For large values of k it is easy to see that

$$\hat{w}(k) = \frac{1}{k}\sum_{r=1}^{k} x_r$$

That is, the solution is the sample mean of the measurements. Most natural!

Let us now return to our original problem and apply the iteration to solve (3.29). Then (3.34) becomes

$$\hat{\boldsymbol{w}}(k) = \hat{\boldsymbol{w}}(k-1) + \rho_k \boldsymbol{x}_k \big(y_k - \boldsymbol{x}_k^T \hat{\boldsymbol{w}}(k-1)\big) \tag{3.40}$$

where $(y_k, \boldsymbol{x}_k)$ are the desired output (± 1)-input training sample pairs, successively presented to the algorithm. The algorithm is known as the least mean squares (LMS) or Widrow–Hoff algorithm, after those who suggested it in the early 1960s [Widr 60, Widr 90]. The algorithm converges asymptotically to the MSE solution.

A number of variants of the LMS algorithm have been suggested and used. The interested reader may consult, for example, [Hayk 96, Kalou 93]. A common variant is to use a constant ρ in the place of ρ_k. However, in this case the algorithm does not converge to the MSE solution. It can be shown, for example, [Hayk 96], that if $0 < \rho < 2/\text{trace}\{R_x\}$ then

$$E[\hat{\boldsymbol{w}}(k)] \rightarrow \boldsymbol{w}_{\text{MSE}} \quad \text{and} \quad E[\|\hat{\boldsymbol{w}}(k) - \boldsymbol{w}_{\text{MSE}}\|^2] \rightarrow \text{constant} \tag{3.41}$$

where $\boldsymbol{w}_{\text{MSE}}$ denotes the MSE optimal estimate and trace$\{\cdot\}$ the trace of the matrix. That is, the mean value of the LMS estimate is equal to the MSE solution, and also the corresponding variance remains finite. It turns out that the smaller the ρ, the smaller the variance around the desired MSE solution. However, the smaller the ρ, the slower the convergence of the LMS algorithm. The reason for using Constant ρ in place of a vanishing sequence is to keep the algorithm "alert" to track variations when the statistics are not stationary but are *slowly* varying, that is, when the underlying distributions are time dependent.

Remarks

- Observe that in the case of the LMS, the parameters' update iteration step, k, coincides with the index of the current input sample $\boldsymbol{x}_k$. In case k is a time index, LMS is a time-*adaptive* scheme, which adapts to the solution as successive samples become available to the system.

- Observe that Eq. (3.40) can be seen as the training algorithm of a *linear neuron*, that is, a neuron without the nonlinear activation function. This type of training, which neglects the nonlinearity *during training* and applies the desired response just after the adder of the linear combiner part of the neuron (Figure 3.5a), was used by Widrow and Hoff. The resulting neuron architecture is known as *adaline* (adaptive linear element). After training and once the weights have been fixed, the model is the same as in Figure 3.5, with the hard limiter following the linear combiner. In other words, the adaline is a neuron that is trained according to the LMS instead of the perceptron algorithm.

3.4.3 Sum of Error Squares Estimation

A criterion closely related to the MSE is the *sum of error squares* or simply the *least squares* (LS) criterion defined as

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{w}\right)^2 \equiv \sum_{i=1}^{N} e_i^2 \tag{3.42}$$

In other words, the errors between the desired output of the classifier (±1 in the two-class case) and the true output are summed up over all the available training feature vectors, instead of averaging them out. In this way, we overcome the need for explicit knowledge of the underlying pdfs. Minimizing (3.42) with respect to $\boldsymbol{w}$ results in

$$\sum_{i=1}^{N} \boldsymbol{x}_i\left(y_i - \boldsymbol{x}_i^T \hat{\boldsymbol{w}}\right) = 0 \Rightarrow \left(\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T\right) \hat{\boldsymbol{w}} = \sum_{i=1}^{N} (\boldsymbol{x}_i y_i) \tag{3.43}$$

For the sake of mathematical formulation let us define

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1l} \\ x_{21} & x_{22} & \dots & x_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nl} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.44}$$

That is, X is an $N \times l$ matrix whose rows are the available training feature vectors, and $\boldsymbol{y}$ is a vector consisting of the corresponding desired responses. Then $\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T = X^T X$ and also $\sum_{i=1}^{N} \boldsymbol{x}_i y_i = X^T \boldsymbol{y}$. Hence, (3.43) can now be written as

$$(X^T X)\hat{\boldsymbol{w}} = X^T \boldsymbol{y} \Rightarrow \hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y} \tag{3.45}$$

Thus, the optimal weight vector is again provided as the solution of a linear set of equations. Matrix $X^T X$ is known as the *sample correlation matrix*. Matrix $X^+ \equiv (X^T X)^{-1} X^T$ is known as the *pseudoinverse* of X, and it is meaningful only if $X^T X$ is invertible, that is, X is of rank l. X^+ is a generalization of the inverse of an invertible square matrix. Indeed, if X is an $l \times l$ square and invertible matrix, then it is straightforward to see that $X^+ = X^{-1}$. In such a case the estimated weight vector is the solution of the linear system $X\hat{\boldsymbol{w}} = \boldsymbol{y}$. If, however, there are more equations than unknowns, $N > l$, as is the usual case in pattern recognition, there is not, in general, a solution. The solution obtained by the pseudoinverse is the vector that minimizes the sum of error squares. It is easy to show that (under mild assumptions) the sum of error squares tends to the MSE solution for large values of N (Problem 3.8).

Remarks

- So far we have restricted the desired output values to be ± 1. Of course, this is not necessary. All we actually need is a desired response positive for ω_1 and negative for ω_2. Thus, in place of ± 1 in the $\boldsymbol{y}$ vector we could have any positive (negative) values. Obviously, all we have said so far is still applicable. However, the interesting aspect of this generalization would be to compute these desired values in an optimal way, in order to obtain a better solution. The Ho–Kashyap algorithm is such a scheme solving for both the optimal $\boldsymbol{w}$ and optimal desired values y_i. The interested reader may consult [Ho 65, Tou 74].
- Generalization to the multi-class case follows the same concept as that introduced for the MSE cost, and it is easily shown that it reduces to M equivalent problems of scalar desired responses, one for each discriminant function (Problem 3.10).

Example 3.4

Class ω_1 consists of the two-dimensional vectors $[0.2, 0.7]^T$, $[0.3, 0.3]^T$, $[0.4, 0.5]^T$, $[0.6, 0.5]^T$, $[0.1, 0.4]^T$ and class ω_2 of $[0.4, 0.6]^T$, $[0.6, 0.2]^T$, $[0.7, 0.4]^T$, $[0.8, 0.6]^T$, $[0.7, 0.5]^T$. Design the sum of error squares optimal linear classifier $w_1x_1 + w_2x_2 + w_0 = 0$.

We first extend the given vectors by using 1 as their third dimension and form the 10×3 matrix X, which has as rows the transposes of these vectors. The resulting sample correlation 3×3 matrix X^TX is equal to

$$X^TX = \begin{bmatrix} 2.8 & 2.24 & 4.8 \\ 2.24 & 2.41 & 4.7 \\ 4.8 & 4.7 & 10 \end{bmatrix}$$

The corresponding $\boldsymbol{y}$ consists of five 1's and then five -1's and

$$X^T\boldsymbol{y} = \begin{bmatrix} -1.6 \\ 0.1 \\ 0.0 \end{bmatrix}$$

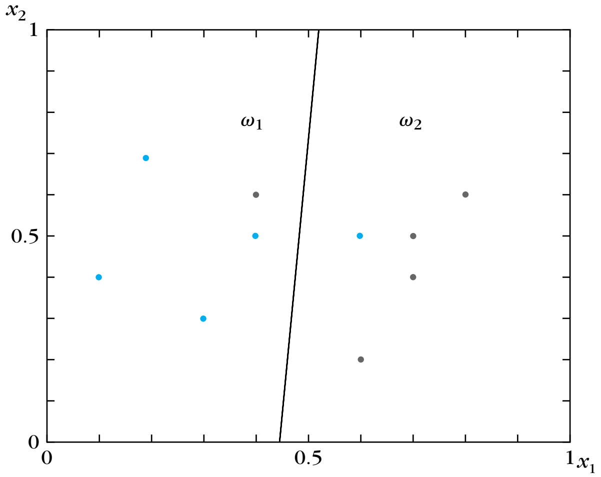

FIGURE 3.7

Least sum of error squares linear classifier. The task is not linearly separable. The linear LS classifier classifies some of the points in the wrong class. However, the resulting sum of error squares is minimum.

Solving the corresponding set of equations results in $[w_1, w_2, w_0] = [-3.218, 0.241, 1.431]$. Figure 3.7 shows the resulting geometry.

3.5 MEAN SQUARE ESTIMATION REVISITED

3.5.1 Mean Square Error Regression

In this subsection, we will approach the MSE task from a slightly different perspective and in a more general framework.

Let $\boldsymbol{y}$, $\boldsymbol{x}$ be two random vector variables of dimensions $M \times 1$ and $l \times 1$, respectively, and assume that they are described by the joint pdf $p(\boldsymbol{y}, \boldsymbol{x})$. The task of interest is to estimate the value of $\boldsymbol{y}$, given the value of $\boldsymbol{x}$ that is obtained from an experiment. No doubt the classification task falls under this more general formulation. For example, when we are given a feature vector $\boldsymbol{x}$, our goal is to estimate the value of the class label y, which is ± 1 in the two-class case. In a more general setting, the values of $\boldsymbol{y}$ may not be discrete. Take, as an example, the case where $y \in \mathcal{R}$ is generated by an unknown rule, i.e.,

$$y = f(\boldsymbol{x}) + \epsilon$$

where $f(\cdot)$ is some unknown function and ϵ is a noise source. The task now is to estimate (predict) the value of y, given the value of $\boldsymbol{x}$. Once more, this is a problem of designing a function $g(\boldsymbol{x})$, based on a set of training data points $(y_i, \boldsymbol{x}_i)$, $i = 1, 2, \ldots, N$, so that the predicted value

$$\hat{y} = g(\boldsymbol{x})$$

to be as close as possible to the true value y in some optimal sense. This type of problem is known as a *regression* task. One of the most popular optimality criteria for regression is the mean square error (MSE). In this section, we will focus on the MSE regression and highlight some of its properties.

The mean square estimate $\hat{\boldsymbol{y}}$ of the random vector $\boldsymbol{y}$, given the value $\boldsymbol{x}$, is defined as

$$\hat{\boldsymbol{y}} = \arg\min_{\tilde{\boldsymbol{y}}} E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] \tag{3.46}$$

Note that the mean value here is with respect to the conditional pdf $p(\boldsymbol{y}|\boldsymbol{x})$. We will show that the optimal estimate is the mean value of $\boldsymbol{y}$, that is,

$$\hat{\boldsymbol{y}} = E[\boldsymbol{y}|\boldsymbol{x}] \equiv \int_{-\infty}^{\infty} \boldsymbol{y}p(\boldsymbol{y}|\boldsymbol{x})\,d\boldsymbol{y} \tag{3.47}$$

Proof. Let $\tilde{\boldsymbol{y}}$ be another estimate. It will be shown that it results in higher mean square error. Indeed,

$$\begin{aligned} E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] = E[\|\boldsymbol{y} - \hat{\boldsymbol{y}} + \hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}\|^2] = E[\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2] \\ + E[\|\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}\|^2] + 2E[(\boldsymbol{y} - \hat{\boldsymbol{y}})^T(\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}})] \end{aligned} \tag{3.48}$$

where the dependence on $\boldsymbol{x}$ has been omitted for notational convenience. Note now that $\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}$ is a constant. Thus,

$$E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] \geq E[\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2] + 2E[(\boldsymbol{y} - \hat{\boldsymbol{y}})^T](\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}) \tag{3.49}$$

and from the definition of $\hat{\boldsymbol{y}} = E[\boldsymbol{y}]$ it follows that

$$E[\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|^2] \geq E[\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2] \tag{3.50}$$

□

Remark

- This is a very elegant result. Given a measured value of $\boldsymbol{x}$, the best (in the MSE sense) estimate of $\boldsymbol{y}$ is given by the function $\boldsymbol{y}(\boldsymbol{x}) \equiv E[\boldsymbol{y}|\boldsymbol{x}]$. In general,this is a *nonlinear* vector-valued function of $\boldsymbol{x}$ (i.e., $\boldsymbol{g}(\cdot) \equiv [g_1(\cdot), \ldots, g_M(\cdot)]^T$), and it is known as the *regression* of $\boldsymbol{y}$ *conditioned* on $\boldsymbol{x}$. It can be shown (Problem 3.11) that if $(\boldsymbol{y}, \boldsymbol{x})$ are jointly Gaussian, then the MSE optimal regressor is a linear function.

3.5.2 MSE Estimates Posterior Class Probabilities

In the beginning of the chapter we promised to “emancipate” ourselves from the Bayesian classification. However, the nice surprise is that a Bayesian flavor still remains, although in a disguised form. Let us reveal it—it can only be beneficial.

We will consider the multiclass case. Given $\boldsymbol{x}$, we want to estimate its class label. Let $g_i(\boldsymbol{x})$ be the discriminant functions to be designed. The cost function in Eq. (3.33) now becomes

$$J = E\left[\sum_{i=1}^{M}(g_i(\boldsymbol{x}) - y_i)^2\right] \equiv E[\|\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{y}\|^2] \tag{3.51}$$

where the vector $\boldsymbol{y}$ consists of zeros and a single 1 at the appropriate place. Note that each $g_i(\boldsymbol{x})$ depends *only* on $\boldsymbol{x}$, whereas the y_i’s depend on the specific class to which $\boldsymbol{x}$ belongs. Let $p(\boldsymbol{x}, \omega_i)$ be the joint probability density of the feature vector belonging to the ith class. Then (3.51) is written as

$$J = \int_{-\infty}^{+\infty} \sum_{j=1}^{M} \left\{ \sum_{i=1}^{M} (g_i(\boldsymbol{x}) - y_i)^2 \right\} p(\boldsymbol{x}, \omega_j)\, d\boldsymbol{x} \tag{3.52}$$

Taking into account that $p(\boldsymbol{x}, \omega_j) = P(\omega_j|\boldsymbol{x})p(\boldsymbol{x})$, (3.52) becomes

$$\begin{aligned} J &= \int_{-\infty}^{\infty} \left\{ \sum_{j=1}^{M} \sum_{i=1}^{M} \left(g_i(\boldsymbol{x}) - y_i\right)^2 P(\omega_j|\boldsymbol{x}) \right\} p(\boldsymbol{x})\, d\boldsymbol{x} \\ &= E\left[\sum_{j=1}^{M} \sum_{i=1}^{M} \left(g_i(\boldsymbol{x}) - y_i\right)^2 P(\omega_j|\boldsymbol{x}) \right] \end{aligned} \tag{3.53}$$

where the mean is taken with respect to $\boldsymbol{x}$. Expanding this, we get

$$J = E\left[\sum_{j=1}^{M} \sum_{i=1}^{M} \left(g_i^2(\boldsymbol{x})P(\omega_j|\boldsymbol{x}) - 2g_i(\boldsymbol{x})y_iP(\omega_j|\boldsymbol{x}) + y_i^2P(\omega_j|\boldsymbol{x})\right) \right] \tag{3.54}$$

Exploiting the fact that $g_i(\boldsymbol{x})$ is a function of $\boldsymbol{x}$ only and $\sum_{j=1}^{M} P(\omega_j|\boldsymbol{x}) = 1$, (3.54) becomes

$$\begin{aligned} J &= E\left[\sum_{i=1}^{M} \left(g_i^2(\boldsymbol{x}) - 2g_i(\boldsymbol{x}) \sum_{j=1}^{M} y_iP(\omega_j|\boldsymbol{x}) + \sum_{j=1}^{M} y_i^2P(\omega_j|\boldsymbol{x}) \right) \right] \\ &= E\left[\sum_{i=1}^{M} \left(g_i^2(\boldsymbol{x}) - 2g_i(\boldsymbol{x})E[\,y_i|\boldsymbol{x}] + E\left[y_i^2|\boldsymbol{x}\right]\right) \right] \end{aligned} \tag{3.55}$$

where $E[y_i|\boldsymbol{x}]$ and $E[y_i^2|\boldsymbol{x}]$ are the respective mean values conditioned on $\boldsymbol{x}$. Adding and subtracting $(E[y_i|\boldsymbol{x}])^2$, Eq. (3.55) becomes

$$J = E\left[\sum_{i=1}^{M}\left(g_i(\boldsymbol{x}) - E[y_i|\boldsymbol{x}]\right)^2\right] + E\left[\sum_{i=1}^{M}\left(E\left[y_i^2|\boldsymbol{x}\right] - (E[y_i|\boldsymbol{x}])^2\right)\right] \tag{3.56}$$

The second term in (3.56) does not depend on the functions $g_i(\boldsymbol{x})$, $i = 1, 2, \ldots, M$. Thus, minimization of J with respect to (the parameters of) $g_i(\cdot)$ affects only the first of the two terms. Let us concentrate and look at it more carefully. Each of the M summands involves two terms: the unknown discriminant function $g_i(\cdot)$ and the conditional mean of the corresponding desired response. Let us now write $g_i(\cdot) = g_i(\cdot; \boldsymbol{w}_i)$, to state explicitly that the functions are defined in terms of a set of parameters, to be determined optimally during training. Minimizing J with respect to $\boldsymbol{w}_i$, $i = 1, 2, \ldots, M$, results in *the mean square estimates of the unknown parameters, $\hat{\boldsymbol{w}}_i$, so that the discriminant functions approximate optimally the corresponding conditional means—that is, the regressions of y_i conditioned on $\boldsymbol{x}$*. Moreover, for the M-class problem and the preceding definitions we have

$$E[y_i|\boldsymbol{x}] \equiv \sum_{j=1}^{M} y_i P(\omega_j|\boldsymbol{x}) \tag{3.57}$$

However $y_i = 1(0)$ if $\boldsymbol{x} \in \omega_i (\boldsymbol{x} \in \omega_j, j \neq i)$. Hence

$$g_i(\boldsymbol{x}, \hat{\boldsymbol{w}}_i) \text{ is the MSE estimate of } P(\omega_i|\boldsymbol{x}) \tag{3.58}$$

This is an important result. Training the discriminant functions g_i with desired outputs 1 *or* 0 *in the MSE sense, Eq. (3.51) is equivalent to obtaining the MSE estimates of the class posterior probabilities, without using any statistical information or pdf modeling!* It suffices to say that these estimates may in turn be used for Bayesian classification. An important issue here is to assess *how good the resulting estimates are. It all depends on how well the adopted functions $g_i(\cdot; \boldsymbol{w}_i)$ can model the desired (in general) nonlinear functions $P(\omega_i|\boldsymbol{x})$*. If, for example, we adopt linear models, as was the case in Eq. (3.33), and $P(\omega_i|\boldsymbol{x})$ is highly nonlinear, the resulting MSE optimal approximation will be a bad one. Our focus in the next chapter will be on developing modeling techniques for nonlinear functions.

Finally, it must be emphasized that the *conclusion above is an implication of the cost function itself and not of the specific model function used*. The latter plays its part when the approximation accuracy issue comes into the scene. MSE cost is just one of the costs that have this important property. Other cost functions share this property too, see, for example, [Rich 91, Bish 95, Pear 90, Cid 99]. In [Guer 04] a procedure is developed to design cost functions that provide more accurate estimates of the probability values, taking into account the characteristics of each classification problem.

3.5.3 The Bias–Variance Dilemma

So far we have touched on some very important issues concerning the interpretation of the output of an optimally designed classifier. Also, we saw that a regressor or a classifier can be viewed as *learning machines* realizing a function or a set of functions $g(\boldsymbol{x})$, which attempt to estimate the corresponding value or class label y and make a decision based on these estimates. In practice, the functions $g(\cdot)$ are estimated using a *finite* training data set $\mathcal{D} = \{(y_i, \boldsymbol{x}_i), i = 1, 2, \ldots, N\}$ and a suitable methodology (e.g., mean square error, sum of error squares, LMS). To emphasize the explicit dependence on $\mathcal{D}$, we write $g(\boldsymbol{x}; \mathcal{D})$. This subsection focuses on the capabilities of $g(\boldsymbol{x}; \mathcal{D})$ to approximate the MSE optimal regressor $E[y|\boldsymbol{x}]$ and on how this is affected by the *finite* size, N, of the training data set.

The key factor here is the dependence of the approximation on $\mathcal{D}$. The approximation may be very good for a specific training data set but very bad for another. The effectiveness of an estimator can be evaluated by computing its mean square deviation from the desired optimal value. This is achieved by averaging over all possible sets $\mathcal{D}$ of size N, that is,

$$E_{\mathcal{D}}\left[\left(g(\boldsymbol{x}; \mathcal{D}) - E[y|\boldsymbol{x}]\right)^2\right] \tag{3.59}$$

If we add and subtract $E_{\mathcal{D}}[g(\boldsymbol{x}; \mathcal{D})]$ and follow a procedure similar to that in the proof of (3.47), we easily obtain

$$\begin{aligned} E_{\mathcal{D}}\left[\left(g(\boldsymbol{x}; \mathcal{D}) - E[y|\boldsymbol{x}]\right)^2\right] &= (E_{\mathcal{D}}[g(\boldsymbol{x}; \mathcal{D})] - E[y|\boldsymbol{x}])^2 \\ &\quad + E_{\mathcal{D}}\left[(g(\boldsymbol{x}; \mathcal{D}) - E_{\mathcal{D}}[g(\boldsymbol{x}; \mathcal{D})])^2\right] \end{aligned} \tag{3.60}$$

The first term is the contribution of the *bias* and the second that of the *variance*. In other words, even if the estimator is unbiased, it can still result in a large mean square error due to a large variance term. For a finite data set, it turns out that there is a trade-off between these two terms. Increasing the bias decreases the variance and vice versa. This is known as the *bias–variance dilemma*. This behavior is reasonable. The problem at hand is similar to that of a curve fitting through a given data set. If, for example, the adopted model is complex (many parameters involved) with respect to the number N, the model will fit the idiosyncrasies of the specific data set. *Thus, it will result in low bias but will yield high variance, as we change from one data set to another*. The major issue now is to seek ways to make both bias and variance low at the same time. It turns out that this may be possible only asymptotically, as the number N grows to infinity. Moreover, N has to grow in such a way as *to allow more complex models, g, to be fitted (which reduces bias) and at the same time to ensure low variance*. However, in practice N is finite, and one should aim at the best compromise. If, on the other hand, some *a priori* knowledge is available, this must be exploited in the form of constraints that the classifier/regressor has to satisfy. This can lead to lower values of both the variance and the bias, compared with a more general type of classifier/regressor. This is natural, because one takes advantage of the available information and helps the optimization process.

Let us now use two simplified "extreme" example cases, which will help us grasp the meaning of the bias–variance dilemma using common-sense reasoning. Let us assume that our data are generated by the following mechanism

$$y = f(x) + \epsilon$$

where $f(\cdot)$ is an unknown function and ϵ a noise source of zero mean and known variance equal to, say, σ_ϵ^2. Obviously, for any x, the optimum MSE regressor is $E[y|x] = f(x)$. To make our point easier to understand, let us further assume that the randomness in the different training sets, $\mathcal{D}$, is due to the y_i's (whose values are affected by the noise), while the respective points, x_i, are fixed. Such an assumption is not an unreasonable one. Since our goal is to obtain an estimate of $f(\cdot)$, it is sensible for one to divide the interval $[x_1, x_2]$, in which x lies, in equally spaced points. For example, one can choose $x_i = x_1 + \frac{x_2-x_1}{N-1}(i-1),\ i = 1, 2, \ldots, N$.

- *Case 1.* Choose the estimate of $f(x)$, $g(x;\mathcal{D})$, to be independent of $\mathcal{D}$, for example,

  $$g(x) = w_1x + w_0$$

  for some fixed values of w_1 and w_0. Figure 3.8 illustrates this setup showing a line $g(x)$ and $N = 11$ training pairs (y_i, x_i), which spread around $f(x), x \in [0, 1]$. Since $g(x)$ is fixed and does not change, we have $E_{\mathcal{D}}[g(x;\mathcal{D})] = g(x;\mathcal{D}) \equiv g(x)$, and the variance term in (3.60) is zero. On the other hand, since $g(x)$ has been chosen arbitrarily, in general, one expects the bias term to be large.

- *Case 2.* In contrast to $g(x)$, the function $g_1(x)$, shown in Figure 3.8, corresponds to a polynomial of high degree and with a large enough number of free parameters so that, for each one of the different training sets $\mathcal{D}$, the respective graphs of $g_1(x)$ pass through the training points $(y_i, x_i), i = 1, 2, \ldots, 11$. For this case, due to the zero mean of the noise source, we have that $E_{\mathcal{D}}[g_1(x;\mathcal{D})] = f(x) = E[y|x]$, for any $x = x_i$. That is, at the training points, the bias is zero. Due to the continuity of $f(x)$ and $g_1(x)$, one expects similar behavior and at the points that lie in the vicinity of the training points x_i. Thus, if N is large enough we can expect the bias to be small for all the points in the interval $[0, 1]$. However, now the variance increases. Indeed, for this case we have that

  $$E_{\mathcal{D}}\left[\left(g_1(x;\mathcal{D}) - E_{\mathcal{D}}[g_1(x;\mathcal{D})]\right)^2\right] = E_{\mathcal{D}}\left[\left(f(x) + \epsilon - f(x)\right)^2\right]$$
  $$= \sigma_\epsilon^2, \text{ for } x = x_i,\ i = 1, 2, \ldots, N$$

  In other words, the bias becomes zero (or approximately zero) but the variance is now equal to the variance of the noise source.

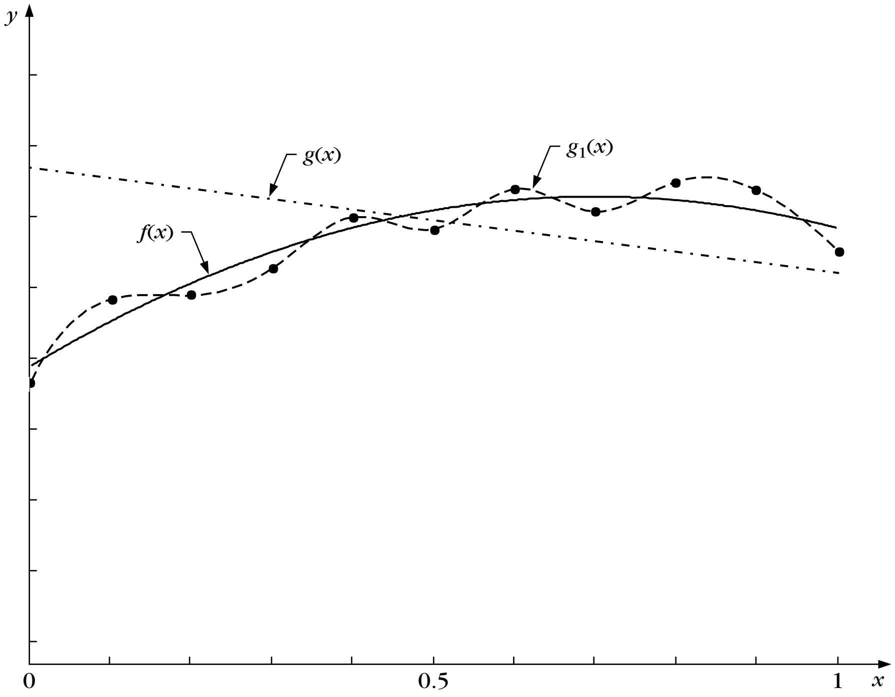

FIGURE 3.8

The data points are spread around the $f(x)$ curve. The line $g(x) = 0$ exhibits zero variance but high bias. The high degree polynomial curve, $g_1(x) = 0$, always passes through the training points and leads to low bias (zero bias at the training points) but to high variance.

The reader will notice that everything that has been said so far applies to both the regression and the classification tasks. The reason that we talked only for regression is that the mean square error is not the best criterion to validate the performance of a classifier. After all, we may have a classifier that results in high mean square error, yet its error performance can be very good. Take, as an example, the case of a classifier $g(x)$ resulting in relatively high mean square error, but predicts the correct class label y for most of the values of x. That is, for all points originating from class ω_1 (ω_2) the predicted values lie, for most of the cases, on the correct side of the classifier, albeit with a lot of variation (for the different training sets) and away from the desired values of ± 1. From the classification point of view, such a designed classifier would be perfectly acceptable. Concerning the preceding theory, in order to get more meaningful results, for the classification task, one has to rework the previous theory in terms of the probability of error. However, now the algebra gets a bit more involved, and some further assumptions need to be adopted (e.g., Gaussian data), in order to make the algebra more tractable. We will not delve into that, since more recent and elegant theories are now available, which study the trade-off between model complexity and the accuracy of the resulting classifier for finite data sets in a generalized framework (see Chapter 5).

A simple and excellent treatment of the bias–variance dilemma task can be found in [Gema 92]. As for ourselves, this was only the beginning. We will come to the finite data set issue and its implications many times throughout this book and from different points of view.

3.6 LOGISTIC DISCRIMINATION

In *logistic discrimination* the logarithm of the likelihood ratios [Eq. (2.20)] is modeled via linear functions. That is,

$$\ln \frac{P(\omega_i|\boldsymbol{x})}{P(\omega_M|\boldsymbol{x})} = w_{i,0} + \boldsymbol{w}_i^T \boldsymbol{x}, \ i = 1, 2, \ldots, M-1 \tag{3.61}$$

In the denominator, any class other than ω_M can also be used. The unknown parameters, $w_{i,0}, \boldsymbol{w}_i, \ i = 1, 2, \ldots, M-1$, must be chosen to ensure that probabilities add to one. That is,

$$\sum_{i=1}^{M} P(\omega_i|\boldsymbol{x}) = 1 \tag{3.62}$$

Combining (3.61) and (3.62), it is straightforward to see that this type of linear modeling is equivalent to an exponential modeling of the *a posteriori* probabilities

$$P(\omega_M|\boldsymbol{x}) = \frac{1}{1 + \sum_{i=1}^{M-1} \exp\left(w_{i,0} + \boldsymbol{w}_i^T \boldsymbol{x}\right)} \tag{3.63}$$

$$P(\omega_i|\boldsymbol{x}) = \frac{\exp\left(w_{i,0} + \boldsymbol{w}_i^T \boldsymbol{x}\right)}{1 + \sum_{i=1}^{M-1} \exp\left(w_{i,0} + \boldsymbol{w}_i^T \boldsymbol{x}\right)}, \ i = 1, 2, \ldots M-1 \tag{3.64}$$

For the two-class case, the previous equations are simplified to

$$P(\omega_2|\boldsymbol{x}) = \frac{1}{1 + \exp(w_0 + \boldsymbol{w}^T \boldsymbol{x})} \tag{3.65}$$

$$P(\omega_1|\boldsymbol{x}) = \frac{\exp(w_0 + \boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(w_0 + \boldsymbol{w}^T \boldsymbol{x})} \tag{3.66}$$

To estimate the set of the unknown parameters, a maximum likelihood approach is usually employed. Optimization is performed with respect to all parameters, which we can think of as the components of a parameter vector $\boldsymbol{\theta}$. Let $\boldsymbol{x}_k, \ k = 1, 2, \ldots, N$, be the training feature vectors with known class labels. Let us denote by $\boldsymbol{x}_k^{(m)}$, $k = 1, 2, \ldots, N_m$, the vectors originating from class $m = 1, 2, \ldots, M$. Obviously, $\sum_m N_m = N$. The log-likelihood function to be optimized is given by

$$L(\boldsymbol{\theta}) = \ln \left\{ \prod_{k=1}^{N_1} p(\boldsymbol{x}_k^{(1)}|\omega_1; \boldsymbol{\theta}) \prod_{k=1}^{N_2} p(\boldsymbol{x}_k^{(2)}|\omega_2; \boldsymbol{\theta}) \ldots \prod_{k=1}^{N_M} p(\boldsymbol{x}_k^{(M)}|\omega_M; \boldsymbol{\theta}) \right\} \tag{3.67}$$

Taking into account that

$$p(\boldsymbol{x}_k^{(m)}|\omega_m; \boldsymbol{\theta}) = \frac{p(\boldsymbol{x}_k^{(m)})P(\omega_m|\boldsymbol{x}_k^{(m)}; \boldsymbol{\theta})}{P(\omega_m)} \tag{3.68}$$

(3.67) becomes

$$L(\boldsymbol{\theta}) = \sum_{k=1}^{N_1} \ln P(\omega_1|\boldsymbol{x}_k^{(1)}) + \sum_{k=1}^{N_2} \ln P(\omega_2|\boldsymbol{x}_k^{(2)}) + \ldots + \sum_{k=1}^{N_M} \ln P(\omega_M|\boldsymbol{x}_k^{(M)}) + C \tag{3.69}$$

where the explicit dependence on $\boldsymbol{\theta}$ has been suppressed for notational simplicity and C is a parameter independent on $\boldsymbol{\theta}$ equal to

$$C = \ln \frac{\prod_{k=1}^{N} p(\boldsymbol{x}_k)}{\prod_{m=1}^{M} P(\omega_m)^{N_m}} \tag{3.70}$$

Inserting Eqs. (3.63) and (3.64) in (3.69), any optimization algorithm can then be used to perform the required maximization (Appendix C). More on the optimization task and the properties of the obtained solution can be found in, for example, [Ande 82, McLa 92].

There is a close relationship between the method of logistic discrimination and the LDA method, discussed in Chapter 2. It does not take much thought to realize that under the Gaussian assumption and for equal covariance matrices across all classes the following holds true.

$$\begin{aligned}\ln \frac{P(\omega_1|\boldsymbol{x})}{P(\omega_2|\boldsymbol{x})} &= \frac{1}{2}(\boldsymbol{\mu}_2^T \Sigma^{-1} \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1 \Sigma^{-1} \boldsymbol{\mu}_1) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} \boldsymbol{x} \\ &\equiv w_0 + \boldsymbol{w}^T \boldsymbol{x}\end{aligned}$$

Here, the equiprobable two-class case was considered for simplicity. However, LDA and logistic discrimination are not identical methods. Their (subtle) difference lies in the way the unknown parameters are estimated. In LDA, the class probability densities are assumed to be Gaussian and the unknown parameters are, basically, estimated by maximizing (3.67) directly. In this maximization, the marginal probability densities ($p(\boldsymbol{x}_k)$) play their own part, since they enter implicitly into the game. However, in the case of logistic discrimination, marginal densities contribute to C and do not affect the solution. Thus, if the Gaussian assumption is a reasonable one for the problem at hand, LDA is the natural approach since it exploits all available information. On the other hand, if this is not a good assumption, then logistic discrimination seems to be a better candidate, since it relies on fewer assumptions. However, in practice it has been reported [Hast 01] that there is little difference between the results obtained by the two methods. Generalizations of the logistic discrimination method to include nonlinear models have also been suggested. See, for example, [Yee 96, Hast 01].

3.7 SUPPORT VECTOR MACHINES

3.7.1 Separable Classes

In this section, an alternative rationale for designing linear classifiers will be adopted. We will start with the two-class linearly separable task, and then we will extend the method to more general cases where data are not separable.

Let $\boldsymbol{x}_i, i = 1, 2, \ldots, N$, be the feature vectors of the training set, X. These belong to either of two classes, ω_1, ω_2, which are assumed to be linearly separable. The goal, once more, is to design a hyperplane

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0 \tag{3.71}$$

that classifies correctly all the training vectors. As we have already discussed in Section 3.3, such a hyperplane is not unique. The perceptron algorithm may converge to any one of the possible solutions. Having gained in experience, this time we will be more demanding. Figure 3.9 illustrates the classification task with

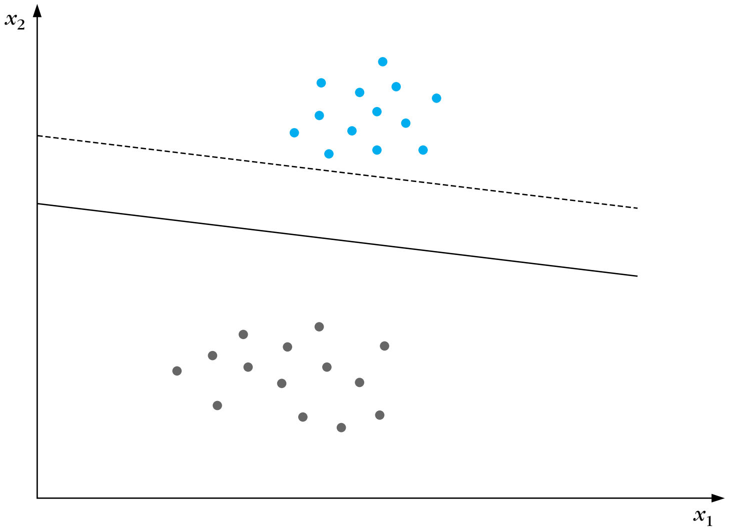

FIGURE 3.9

An example of a linearly separable two-class problem with two possible linear classifiers.

two possible hyperplane[1] solutions. Both hyperplanes do the job for the training set. However, which one of the two would any sensible engineer choose as the classifier for operation in practice, where data outside the training set will be fed to it? No doubt the answer is: the full-line one. The reason is that this hyperplane leaves more "room" on either side, so that data in both classes can move a bit more freely, with less risk of causing an error. Thus such a hyperplane can be trusted more, when it is faced with the challenge of operating with unknown data. Here we have touched a very important issue in the classifier design stage. It is known as the *generalization performance of the classifier*. This refers to the capability of the classifier, designed using the training data set, to operate satisfactorily with data outside this set. We will come to this issue over and over again.

After the above brief discussion, we are ready to accept that a very sensible choice for the hyperplane classifier would be the one that leaves the maximum margin from both classes. Later on, at the end of Chapter 5, we will see that this sensible choice has a deeper justification, springing from the elegant mathematical formulation that Vapnik and Chervonenkis have offered to us.

Let us now quantify the term *margin* that a hyperplane leaves from both classes. Every hyperplane is characterized by its direction (determined by $\boldsymbol{w}$) and its exact position in space (determined by w_0). Since we want to give no preference to either of the classes, then it is reasonable for each direction to select that hyperplane which has the same distance from the respective nearest points in ω_1 and ω_2. This is illustrated in Figure 3.10. The hyperplanes shown with dark lines are the selected ones from the infinite set in the respective direction. The margin for direction "1" is $2z_1$, and the margin for direction "2" is $2z_2$. *Our goal is to search for the direction that gives the maximum possible margin*. However, each hyperplane is determined within a scaling factor. We will free ourselves from it, by appropriate scaling of all the candidate hyperplanes. Recall from Section 3.2 that the distance of a point from a hyperplane is given by

$$z = \frac{|g(\boldsymbol{x})|}{\|\boldsymbol{w}\|}$$

We can now scale $\boldsymbol{w}, w_0$ so that the value of $g(\boldsymbol{x})$, at the nearest points in ω_1, ω_2 (circled in Figure 3.10), is equal to 1 for ω_1 and, thus, equal to -1 for ω_2. This is equivalent with

1. Having a margin of $\frac{1}{\|\boldsymbol{w}\|} + \frac{1}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$
2. Requiring that

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 1, \qquad \forall \boldsymbol{x} \in \omega_1$$
$$\boldsymbol{w}^T\boldsymbol{x} + w_0 \leq -1, \qquad \forall \boldsymbol{x} \in \omega_2$$

[1] We will refer to lines as hyperplanes to cover the general case.

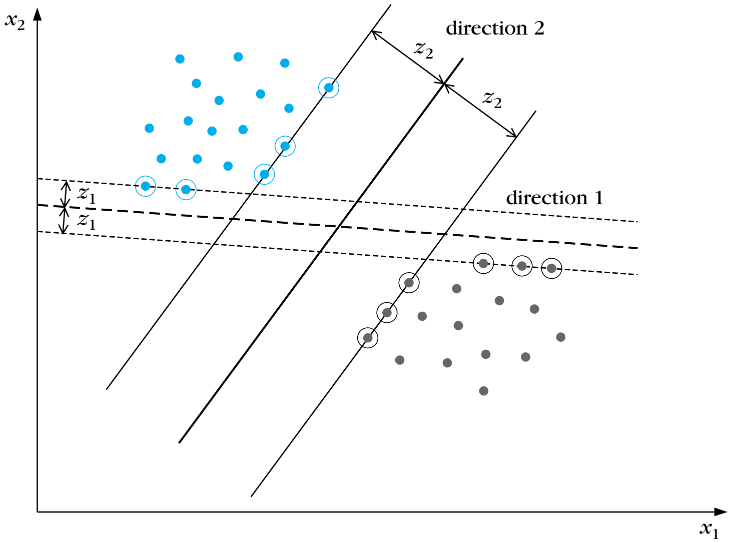

FIGURE 3.10

An example of a linearly separable two-class problem with two possible linear classifiers.

We have now reached the point where mathematics will take over. For each $\boldsymbol{x}_i$, we denote the corresponding class indicator by y_i ($+1$ for ω_1, -1 for ω_2.) Our task can now be summarized as: Compute the parameters $\boldsymbol{w}, w_0$ of the hyperplane so that to:

$$\text{minimize} \quad J(\boldsymbol{w}, w_0) \equiv \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{3.72}$$

$$\text{subject to} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) \geq 1, \quad i = 1, 2, \ldots, N \tag{3.73}$$

Obviously, minimizing the norm makes the margin maximum. This is a nonlinear (quadratic) optimization task subject to a set of linear inequality constraints. The Karush–Kuhn–Tucker (KKT) conditions (Appendix C) that the minimizer of (3.72), (3.73) has to satisfy are

$$\frac{\partial}{\partial \boldsymbol{w}}\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = \boldsymbol{0} \tag{3.74}$$

$$\frac{\partial}{\partial w_0}\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = 0 \tag{3.75}$$

$$\lambda_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.76}$$

$$\lambda_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - 1] = 0, \quad i = 1, 2, \ldots, N \tag{3.77}$$

where $\boldsymbol{\lambda}$ is the vector of the Lagrange multipliers, λ_i, and $\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda})$ is the Lagrangian function defined as

$$\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{i=1}^{N} \lambda_i [y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - 1] \tag{3.78}$$

Combining (3.78) with (3.74) and (3.75) results in

$$\boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i \tag{3.79}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.80}$$

Remarks

- The Lagrange multipliers can be either zero or positive (Appendix C). Thus, the vector parameter $\boldsymbol{w}$ of the optimal solution is a linear combination of $N_s \leq N$ feature vectors that are associated with $\lambda_i \neq 0$. That is,

$$\boldsymbol{w} = \sum_{i=1}^{N_s} \lambda_i y_i \boldsymbol{x}_i \tag{3.81}$$

These are known as *support vectors* and the optimum hyperplane classifier as a *support vector machine* (SVM). As it is pointed out in Appendix C, a nonzero Lagrange multiplier corresponds to a so called active constraint. Hence, as the set of constraints in (3.77) suggests for $\lambda_i \neq 0$, *the support vectors lie on either of the two hyperplanes*, that is,

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = \pm 1 \tag{3.82}$$

In other words, they are the training vectors that are closest to the linear classifier, and they constitute the *critical elements of the training set*. Feature vectors corresponding to $\lambda_i = 0$ can either lie outside the "class separation band," defined as the region between the two hyperplanes given in (3.82), or they can also lie on one of these hyperplanes (degenerate case, Appendix C). The resulting hyperplane classifier is insensitive to the number and position of such feature vectors, provided they do not cross the class separation band.

- Although $\boldsymbol{w}$ is explicitly given, w_0 can be implicitly obtained by any of the *(complementary slackness)* conditions (3.77), satisfying strict complementarity (i.e., $\lambda_i \neq 0$, Appendix C). In practice, w_0 is computed as an average value obtained using all conditions of this type.

- The cost function in (3.72) is a strict convex one (Appendix C), a property that is guaranteed by the fact that the corresponding Hessian matrix is positive definite [Flet 87]. Furthermore, the inequality constraints consist of linear functions. As discussed in Appendix C, these two conditions guarantee that any local minimum is also global and unique. This is most welcome. *The optimal hyperplane classifier of a support vector machine is unique.*

Having stated all these very interesting properties of the optimal hyperplane of a support vector machine, we next need to compute the involved parameters. From a computational point of view this is not always an easy task, and a number of algorithms exist, for example, [Baza 79]. We will move to a path, which is suggested to us by the special nature of our optimization task, given in (3.72) and (3.73). It belongs to the *convex programming* family of problems, since the cost function is convex and the constraints are linear and define a convex set of feasible solutions. As we discuss in Appendix C, such problems can be solved by considering the so-called *Lagrangian duality*. The problem can be stated equivalently by its Wolfe dual representation form, that is,

$$\text{maximize} \quad \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) \tag{3.83}$$

$$\text{subject to} \quad \boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i \tag{3.84}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.85}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0} \tag{3.86}$$

The two equality constraints are the result of equating to zero the gradient of the Lagrangian, with respect to $\boldsymbol{w}, w_0$. We have already gained something. The training feature vectors enter into the problem via equality constraints and not inequality ones, which can be easier to handle. Substituting (3.84) and (3.85) into (3.83) and after a bit of algebra we end up with the equivalent optimization task

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right) \tag{3.87}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.88}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0} \tag{3.89}$$

Once the optimal Lagrange multipliers have been computed, by maximizing (3.87), the optimal hyperplane is obtained via (3.84), and w_0 via the complementary slackness conditions, as before.

Remarks

- Besides the more attractive setting of the involved constraints in (3.87), (3.88), there is another important reason that makes this formulation popular. The training vectors enter into the game in pairs, in the form of inner products. This is most interesting. *The cost function does not depend explicitly on the dimensionality of the input space!* This property allows for efficient generalizations in the case of nonlinearly separable classes. We will return to this at the end of Chapter 4.
- Although the resulting optimal hyperplane is unique, there is no guarantee about the uniqueness of the associated Lagrange multipliers λ_i. In words, the expansion of $\boldsymbol{w}$ in terms of support vectors in (3.84) may not be unique, although the final result is unique (Example 3.5).

3.7.2 Nonseparable Classes

When the classes are not separable, the above setup is no longer valid. Figure 3.11 illustrates the case in which the two classes are not separable. Any attempt to draw a hyperplane will never end up with a class separation band with no

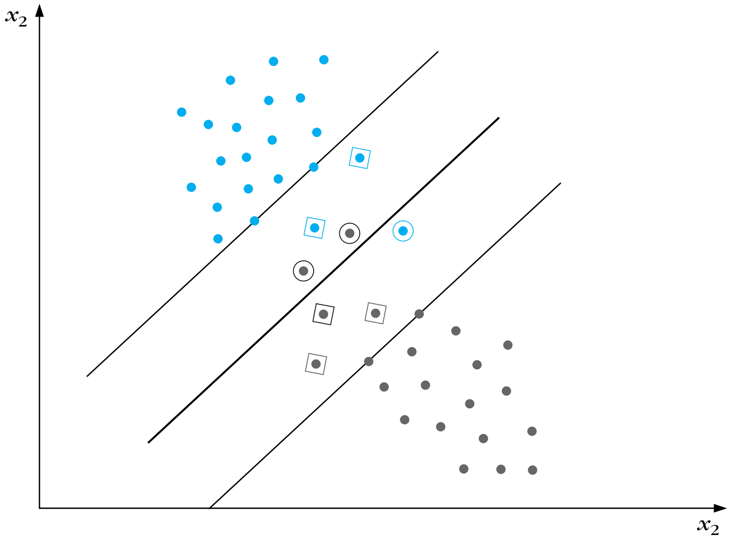

FIGURE 3.11

In the nonseparable class case, points fall inside the class separation band.

data points inside it, as was the case in the linearly separable task. Recall that the margin is defined as the distance between the pair of parallel hyperplanes described by

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = \pm\, 1$$

The training feature vectors now belong to one of the following three categories:

- Vectors that fall outside the band and are correctly classified. These vectors comply with the constraints in (3.73).
- Vectors falling inside the band and are correctly classified. These are the points placed in squares in Figure 3.11, and they satisfy the inequality

$$0 \leq y_i(\boldsymbol{w}^T \boldsymbol{x} + w_0) < 1$$

- Vectors that are misclassified. They are enclosed by circles and obey the inequality

$$y_i(\boldsymbol{w}^T \boldsymbol{x} + w_0) < 0$$

All three cases can be treated under a single type of constraints by introducing a new set of variables, namely,

$$y_i[\boldsymbol{w}^T \boldsymbol{x} + w_0] \geq 1 - \xi_i \tag{3.90}$$

The first category of data corresponds to $\xi_i = 0$, the second to $0 < \xi_i \leq 1$, and the third to $\xi_i > 1$. The variables ξ_i are known as *slack variables*. The optimizing task becomes more involved, yet it falls under the same rationale as before. The goal now is to make the margin as large as possible but at the same time to keep the number of points with $\xi > 0$ as small as possible. In mathematical terms, this is equivalent to adopting to minimize the cost function

$$J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i) \tag{3.91}$$

where $\boldsymbol{\xi}$ is the vector of the parameters ξ_i and

$$I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases} \tag{3.92}$$

The parameter C is a positive constant that controls the relative influence of the two competing terms. However, optimization of the above is difficult since it involves a discontinuous function $I(\cdot)$. As it is common in such cases, we choose to optimize a closely related cost function, and the goal becomes

$$\text{minimize} \quad J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i \tag{3.93}$$

$$\text{subject to} \quad y_i[\boldsymbol{w}^T\boldsymbol{x}_i + w_0] \geq 1 - \xi_i, \quad i = 1, 2, \ldots, N \tag{3.94}$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.95}$$

The problem is again a convex programming one, and the corresponding Lagrangian is given by

$$\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\lambda_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - 1 + \xi_i] \tag{3.96}$$

The corresponding Karush–Kuhn–Tucker conditions are

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} = \boldsymbol{0} \quad \text{or} \quad \boldsymbol{w} = \sum_{i=1}^{N}\lambda_i y_i \boldsymbol{x}_i \tag{3.97}$$

$$\frac{\partial\mathcal{L}}{\partial w_0} = 0 \quad \text{or} \quad \sum_{i=1}^{N}\lambda_i y_i = 0 \tag{3.98}$$

$$\frac{\partial\mathcal{L}}{\partial\xi_i} = 0 \quad \text{or} \quad C - \mu_i - \lambda_i = 0, \quad i = 1, 2, \ldots, N \tag{3.99}$$

$$\lambda_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - 1 + \xi_i] = 0, \quad i = 1, 2, \ldots, N \tag{3.100}$$

$$\mu_i\xi_i = 0, \quad i = 1, 2, \ldots, N \tag{3.101}$$

$$\mu_i \geq 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.102}$$

The associated Wolfe dual representation now becomes

$$\begin{aligned}
\text{maximize} \quad & \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}) \\
\text{subject to} \quad & \boldsymbol{w} = \sum_{i=1}^{N}\lambda_i y_i \boldsymbol{x}_i \\
& \sum_{i=1}^{N}\lambda_i y_i = 0 \\
& C - \mu_i - \lambda_i = 0, \quad i = 1, 2, \ldots, N \\
& \lambda_i \geq 0, \ \mu_i \geq 0, \quad i = 1, 2, \ldots, N
\end{aligned}$$

Substituting the above equality constraints into the Lagrangian, we end up with

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right) \tag{3.103}$$

$$\text{subject to} \quad 0 \leq \lambda_i \leq C, \quad i = 1, 2, \ldots, N \tag{3.104}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.105}$$

Note that the Lagrange multipliers corresponding to the points residing either within the margin or on the wrong side of the classifier, that is, $\xi_i > 0$, are all equal to the maximum allowable value C. Indeed, at the solution, for $\xi_i \neq 0$ the KKT conditions give $\mu_i = 0$ leading to $\lambda_i = C$. In other words, these points have the largest possible "share" in the final solution $\boldsymbol{w}$.

3.7.3 The Multiclass Case

In all our discussions, so far, we have been involved with the two-class classification task. In an M-class problem, a straightforward extension is to consider it as a set of M two-class problems (*one-against-all*). For each one of the classes, we seek to design an optimal discriminant function, $g_i(\boldsymbol{x})$, $i = 1, 2, \ldots, M$, so that $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$, $\forall j \neq i$, if $\boldsymbol{x} \in \omega_i$. Adopting the SVM methodology, we can design the discriminant functions so that $g_i(\boldsymbol{x}) = 0$ to be the optimal hyperplane separating class ω_i from all the others. Thus, each classifier is designed to give $g_i(\boldsymbol{x}) > 0$ for $\boldsymbol{x} \in \omega_i$ and $g_i(\boldsymbol{x}) < 0$ otherwise. Classification is then achieved according to the following rule:

$$\text{assign } \boldsymbol{x} \text{ in } \omega_i \text{ if } i = \arg\max_k \{g_k(\boldsymbol{x})\}$$

This technique, however, may lead to indeterminate regions, where more than one $g_i(\boldsymbol{x})$ is positive (Problem 3.15). Another drawback of the technique is that each binary classifier deals with a rather asymmetric problem in the sense that training is carried out with many more negative than positive examples. This becomes more serious when the number of classes is relatively large.

An alternative technique is the *one-against-one*. In this case, $M(M-1)/2$ binary classifiers are trained and each classifier separates a pair of classes. The decision is made on the basis of a majority vote. The obvious disadvantage of the technique is that a relatively large number of binary classifiers has to be trained. In [Plat 00] a methodology is suggested that may speed up the procedure.

A different and very interesting rationale has been adopted in [Diet 95]. The multiclass task is treated in the context of error correcting coding, inspired by the coding schemes used in communications. For a M-class problem a number of, say, L binary classifiers are used, where L is appropriately chosen by the designer. Each class is now represented by a binary code word of length L. During training, for the ith classifier, $i = 1, 2, \ldots, L$, the desired labels, y, for each class are chosen to be either $+1$ or -1. For each class, the desired labels may be different for the various classifiers. This is equivalent to constructing a matrix $M \times L$ of desired labels. For example, if $M = 4$ and $L = 6$, such a matrix can be

$$\begin{bmatrix} -1 & -1 & -1 & +1 & -1 & +1 \\ +1 & -1 & +1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1 & -1 & +1 \\ -1 & -1 & +1 & -1 & +1 & +1 \end{bmatrix} \tag{3.106}$$

In words, during training, the first classifier (corresponding to the first column of the previous matrix) is designed in order to respond (−1,+1,+1,−1) for patterns originating from classes ω_1, ω_2, ω_3, ω_4, respectively. The second classifier will be trained to respond (−1,−1,+1,−1), and so on. The procedure is equivalent to grouping the classes into L different pairs, and, for each pair, we train a binary classifier accordingly. For the case of our example and for the first binary classifier, class ω_1 has been grouped together with ω_4 and class ω_2 with class ω_3. Each row must be distinct and corresponds to a class. For our example, and in the absence of errors, the outputs of the L classifiers for a pattern from class ω_1 will result in the code word (−1,−1,−1,+1,−1,+1), and so on. When an unknown pattern is presented, the output of each one of the binary classifiers is recorded, resulting in a code word. Then, the Hamming distance (number of places where two code words differ) of this code word is measured against the M code words, and the pattern is classified to the class corresponding to the smallest distance.

Here in lies the power of the technique. If the code words are designed so that the minimum Hamming distance between any pair of them is, say, d, then a correct decision will still be reached even if the decisions of at most $\left\lfloor \frac{d-1}{2} \right\rfloor$, out of the L, classifiers are wrong, where $\lfloor \cdot \rfloor$ is the floor operation. For the matrix in (3.106) the minimum Hamming distance, between any pair, is equal to three. In [Diet 95], the method has been applied for numerical digit classification, and the grouping of the ten classes is done in such a way as to be meaningful. For example, one grouping is based on the existence in the numeric digits of a horizontal line (e.g., “4” and “2”), or the existence of a vertical line (e.g., “1” and “4”), and so on. An extension of this method, which is proposed in [Allw 00], takes into consideration the resulting values of the margin (when an SVM or another type of margin classifier, e.g., boosting classifiers discussed in Chapter 4, is used). In [Zhou 08], the composition of the individual binary problems and their number (code word length, L) is the result of a data-adaptive procedure that designs the code words by taking into account the inherent structure of the training data.

All previous techniques are appropriate for any classifier. Another alternative, specific for SVMs, is to extentd the two class SVM mathematical formulation to the M-class problem, see, for example, [Vapn 98, Liu 06]. Comparative studies of the various methods for multiclass SVM classification can be found in [Rifk 04, Hsu 02, Fei 06].

Remarks

- The only difference between the linearly separable and nonseparable cases lies in the fact that for the latter one the Lagrange multipliers need to be bounded above by C. The linearly separable case corresponds to $C \rightarrow \infty$, see Eqs. (3.104) and (3.89). The slack variables, ξ_i, and their associated Lagrange multipliers, μ_i, do not enter into the problem explicitly. Their presence is indirectly reflected through C.

- A major limitation of support vector machines is the high computational burden required, both during training and in the test phase. A naive implementation of a quadratic programming (QP) solver takes $O(N^3)$ operations, and its memory requirements are of the order of $O(N^2)$. For problems with a relatively small number of training data, any general purpose optimization algorithm can be used. However, for a large number of training points (e.g., of the order of a few thousands), a naive QP implementation does not scale up well, and a special treatment is required. Training of SVM is usually performed in batch mode. For large problems this sets high demands on computer memory requirements. To attack such problems, a number of procedures have been devised. Their philosophy relies on the decomposition, in one way or another, of the optimization problem into a sequence of smaller ones, for example, [Bose 92, Osun 97, Chan 00]. The main rationale behind such algorithms is to start with an arbitrary data subset (chunk of data, working set) that can fit in the computer memory. Optimization is, then, performed on this subset via a general optimizer. Support vectors remain in the working set while others are replaced by new ones, outside the current working set, that violate severely the KKT conditions. It can be shown that this iterative procedure guarantees that the cost function is decreasing at each iteration step.

  In [Plat 99, Matt 99], the so called *Sequential Minimal Optimization* (SMO) algorithm is proposed where the idea of decomposition is pushed to its extreme and each working set consists of only two points. Its great advantage is that the optimization can now be performed analytically. In [Keer 01], a set of heuristics is used for the choice of the pair of points that constitute the working set. To this end, it is suggested that the use of two thresholded parameters can lead to considerable speedups. As suggested in [Plat 99, Platt 98], efficient implementations of such a scheme have an empirical training time complexity that scales between $O(N)$ and $O(N^{2.3})$.

  Theoretical issues related to the algorithm, such as convergence, are addressed in [Chen 06] and the references therein. The parallel implementation of the algorithm is considered in [Cao 06]. In [Joac 98] the working set is the result of a search for the steepest feasible direction. More recently, [Dong 05] suggested a technique to quickly remove most of the nonsupport vectors, using a parallel optimization step, and the original problem can be split into many subproblems that can be solved more efficiently. In [Mavr 06], the geometric interpretation of SVMs (Section 3.7.5) is exploited, and the optimization task is treated as a minimum distance points search between convex sets. It is reported that substantial computational savings can be obtained compared to the SMO algorithm. A sequential algorithm, which operates on the primal problem formulation, has been proposed in [Navi 01], where an iterative reweighted least squares procedure is employed and alternates weight optimization with constraint forcing. An advantage of the latter technique is that it naturally leads to online implementations. Another trend is to employ an algorithm that aims at an approximate solution to the problem. In [Fine 01] a low-rank approximation is used in place of the the so-called kernel matrix, which is involved in the computations. In [Tsan 06, Hush 06] the issues of complexity and accuracy of the approximation are considered together. For example, in [Hush 06] polynomial-time algorithms are derived that produce approximate solutions with a guaranteed accuracy for a class of QP problems that include the SVM classifiers.

For large problems, the test phase can also be quite demanding, if the number of support vectors is excessively high. Methods that speed up computations have also been suggested, for example, [Burg 97, Nguy 06].

Example 3.5

Consider the two-class classification task that consists of the following points:

$$w_1\colon [1,1]^T, [1,-1]^T$$

$$w_2\colon [-1,1]^T, [-1,-1]$$

Using the SVM approach, we will demonstrate that the optimal separating hyperplane (line) is $x_1 = 0$ and that this is obtained via different sets of Lagrange multipliers.

The points lie on the corners of a square, as shown in Figure 3.12. The simple geometry of the problem allows for a straightforward computation of the SVM linear classifier. Indeed, a careful observation of Figure 3.12 suggests that the optimal line

$$g(\boldsymbol{x}) = w_1x_1 + w_2x_2 + w_0 = 0$$

is obtained for $w_2 = w_0 = 0$ and $w_1 = 1$, that is,

$$g(\boldsymbol{x}) = x_1 = 0$$

Hence for this case, all four points become support vectors, and the margin of the separating line from both classes is equal to 1. For any other direction, e.g., $g_1(\boldsymbol{x}) = 0$, the margin is smaller. It must be pointed out that the same solution is obtained if one solves the associated KKT conditions (Problem 3.16.)

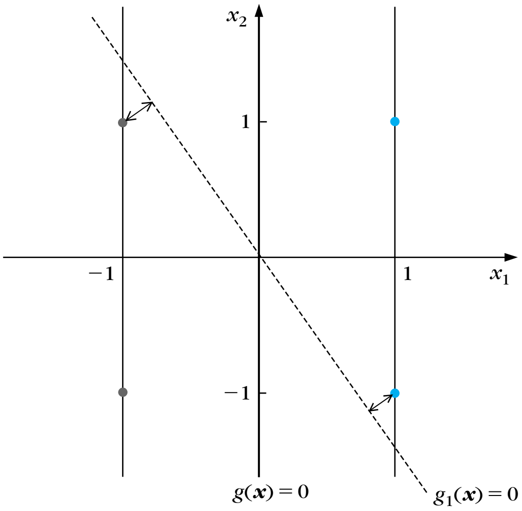

FIGURE 3.12

In this example all four points are support vectors. The margin associated with $g_1(\boldsymbol{x}) = 0$ is smaller compared to the margin defined by the optimal $g(\boldsymbol{x}) = 0$.

Let us now consider the mathematical formulation of our problem. The linear inequality constraints are

$$w_1 + w_2 + w_0 - 1 \geq 0$$
$$w_1 - w_2 + w_0 - 1 \geq 0$$
$$w_1 - w_2 - w_0 - 1 \geq 0$$
$$w_1 + w_2 - w_0 - 1 \geq 0$$

and the associated Lagrangian function becomes

$$\begin{aligned}\mathcal{L}(w_2, w_1, w_0, \boldsymbol{\lambda}) = \frac{w_1^2 + w_2^2}{2} &- \lambda_1(w_1 + w_2 + w_0 - 1)\\ &- \lambda_2(w_1 - w_2 + w_0 - 1)\\ &- \lambda_3(w_1 - w_2 - w_0 - 1)\\ &- \lambda_4(w_1 + w_2 - w_0 - 1)\end{aligned}$$

The KKT conditions are given by

$$\frac{\partial \mathcal{L}}{\partial w_1} = 0 \ \Rightarrow \ w_1 = \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 \tag{3.107}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = 0 \ \Rightarrow \ w_2 = \lambda_1 + \lambda_4 - \lambda_2 - \lambda_3 \tag{3.108}$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = 0 \ \Rightarrow \ \lambda_1 + \lambda_2 - \lambda_3 - \lambda_4 = 0 \tag{3.109}$$

$$\lambda_1(w_1 + w_2 + w_0 - 1) = 0 \tag{3.110}$$

$$\lambda_2(w_1 - w_2 + w_0 - 1) = 0 \tag{3.111}$$

$$\lambda_3(w_1 - w_2 - w_0 - 1) = 0 \tag{3.112}$$

$$\lambda_4(w_1 + w_2 - w_0 - 1) = 0 \tag{3.113}$$

$$\lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0 \tag{3.114}$$

Since we know that the solution for $\boldsymbol{w}, w_0$ is unique, we can substitute the solution $w_1 = 1, w_2 = w_0 = 0$ into the above equations. Then we are left with a linear system of three equations with four unknowns, that is,

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \tag{3.115}$$

$$\lambda_1 + \lambda_4 - \lambda_2 - \lambda_3 = 0 \tag{3.116}$$

$$\lambda_1 + \lambda_2 - \lambda_3 - \lambda_4 = 0 \tag{3.117}$$

which obviously has more than one solution. However, all of them lead to the unique optimal separating line.

Example 3.6

Figure 3.13 shows a set of training data points residing in the two-dimensional space and divided into two nonseparable classes. The full line in Figure 3.13a is the resulting hyperplane using Platt's algorithm and corresponds to the value $C = 0.2$. Dotted lines meet the conditions given in (3.82) and define the margin that separates the two classes, for those points with

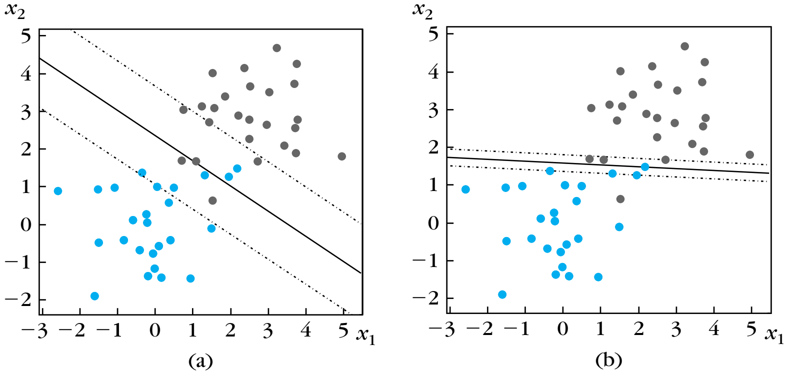

FIGURE 3.13

An example of two nonseparable classes and the resulting SVM linear classifier (full line) with the associated margin (dotted lines) for the values (a) $C = 0.2$ and (b) $C = 1000$. In the latter case, the location and direction of the classifier as well as the width of the margin have changed in order to include a smaller number of points inside the margin.

$\xi_i = 0$. The setting in Figure 3.13b corresponds to $C = 1000$ and has been obtained with the same algorithm and the same set of trimming parameters (e.g., stopping criteria).

It is readily observed that the margin associated with the classifier corresponding to the larger value of C is smaller. This is because the second term in (3.91) has now more influence in the cost, and the optimization process tries to satisfy this demand by reducing the margin and consequently the number of points with $\xi_i > 0$. In other words, the width of the margin does not depend entirely on the data distribution, as was the case with the separable class case, but is heavily affected by the choice of C. This is the reason SVM classifiers, defined by (3.91), are also known as *soft margin classifiers*.

3.7.4 ν-SVM

Example 3.6 demonstrated the close relation that exists between the parameter C and the width of the margin obtained as a result of the optimization process. However, since the margin is such an important entity in the design of SVM (after all, the essence of the SVM methodology is to maximize it), a natural question that arises is why not involve it in a more direct way in the cost function, instead of leaving its control to a parameter (i.e., C) whose relation with the margin, although strong, is not transparent to us. To this end, in [Scho 00] a variant of the soft margin SVM was introduced. The margin is defined by the pair of hyperplanes

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = \pm\ \rho \tag{3.118}$$

and $\rho \geq 0$ is left as a free variable to be optimized. Under this new setting, the primal problem given in (3.93)–(3.95) can now be cast as

$$\text{minimize} \quad J(\boldsymbol{w}, w_0, \boldsymbol{\xi}, \rho) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \nu\rho + \frac{1}{N}\sum_{i=1}^{N}\xi_i \tag{3.119}$$

$$\text{subject to} \quad y_i[\boldsymbol{w}^T\boldsymbol{x}_i + w_0] \geq \rho - \xi_i, \quad i = 1, 2, \ldots, N \tag{3.120}$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.121}$$

$$\rho \geq 0 \tag{3.122}$$

To understand the role of ρ, note that for $\xi_i = 0$ the constraints in (3.120) state that the margin separating the two classes is equal to $\frac{2\rho}{\|\boldsymbol{w}\|}$. In the previous formulation, also known as ν-SVM, we simply count and average the number of points with $\xi_i > 0$, whose number is now controlled by the margin variable ρ. The larger the ρ the wider the margin and the higher the number of points within the margin, for a specific direction $\boldsymbol{w}$. The parameter ν controls the influence of the second term in the cost function, and its value lies in the range $[0, 1]$. (We will revisit this issue later on.)

The Lagrangian function associated with the task (3.119)–(3.122) is given by

$$\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}, \rho, \delta) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \nu\rho + \frac{1}{N}\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i$$
$$- \sum_{i=1}^{N}\lambda_i\left[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - \rho + \xi_i\right] - \delta\rho \qquad (3.123)$$

Adopting similar steps as in Section 3.7.2, the following KKT conditions result:

$$\boldsymbol{w} = \sum_{i=1}^{N}\lambda_i y_i \boldsymbol{x}_i \qquad (3.124)$$

$$\sum_{i=1}^{N}\lambda_i y_i = 0 \qquad (3.125)$$

$$\mu_i + \lambda_i = \frac{1}{N}, \quad i = 1, 2, \ldots, N \qquad (3.126)$$

$$\sum_{i=1}^{N}\lambda_i - \delta = \nu \qquad (3.127)$$

$$\lambda_i\left[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) - \rho + \xi_i\right] = 0, \quad i = 1, 2, \ldots, N \qquad (3.128)$$

$$\mu_i\xi_i = 0, \quad i = 1, 2, \ldots, N \qquad (3.129)$$

$$\delta\rho = 0 \qquad (3.130)$$

$$\mu_i \geq 0, \quad \lambda_i \geq 0, \quad \delta \geq 0, \quad i = 1, 2, \ldots, N \qquad (3.131)$$

The associated Wolfe dual representation is easily shown to be

$$\text{maximize} \quad \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}, \delta) \qquad (3.132)$$

$$\text{subject to} \quad \boldsymbol{w} = \sum_{i=1}^{N}\lambda_i y_i \boldsymbol{x}_i \qquad (3.133)$$

$$\sum_{i=1}^{N}\lambda_i y_i = 0 \qquad (3.134)$$

$$\mu_i + \lambda_i = \frac{1}{N}, \quad i = 1, 2, \ldots, N \qquad (3.135)$$

$$\sum_{i=1}^{N}\lambda_i - \delta = \nu \qquad (3.136)$$

$$\lambda_i \geq 0,\ \mu_i \geq 0,\ \delta \geq 0, \quad i = 1, 2, \ldots, N \qquad (3.137)$$

If we substitute the equality constraints (3.133)–(3.136) in the Lagrangian, the dual problem becomes equivalent to (Problem 3.17)

$$\max_{\boldsymbol{\lambda}} \left(-\frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right) \tag{3.138}$$

$$\text{subject to} \quad 0 \leq \lambda_i \leq \frac{1}{N}, \quad i = 1, 2, \ldots, N \tag{3.139}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.140}$$

$$\sum_{i=1}^{N} \lambda_i \geq \nu \tag{3.141}$$

Once more, only the Lagrange multipliers $\boldsymbol{\lambda}$ enter into the problem explicitly, and ρ and the slack variables, ξ_i, make their presence felt through the bounds appearing in the constraints. Observe that in contrast to (3.103) the cost function is now quadratically homogeneous and the linear term $\sum_{i=1}^{N} \lambda_i$ is not present. Also, the new formulation has an extra constraint.

Remarks

- [Chan 01] shows that the ν-SVM and the more standard SVM formulation [(3.103)–(3.105)], sometimes referred to as C-SVM, lead to the same solution for appropriate values of C and ν. Also, it is shown that in order for the optimization problem to be feasible, the constant ν must lie in a range $0 \leq \nu_{\min} \leq \nu \leq \nu_{\max} \leq 1$.

- Although both SVM formulations result in the same solution, for appropriate choices of ν and C the ν-SVM offers certain advantages to the designer. As we will see in the next section, it leads to a geometric interpretation of the SVM task for nonseparable classes. Furthermore, the constant ν, controlled by the designer, offers itself to serve two important bounds concerning (a) the error rate and (b) the number of the resulting support vectors.

  At the solution, the points lying either within the margin or outside it but on the wrong side of the separating hyperplane correspond to $\xi_i > 0$ and hence to $\mu_i = 0$ [Eq. (3.129)], forcing the respective Lagrange multipliers to be $\lambda_i = \frac{1}{N}$ [Eq. (3.126)]. Also, since at the solution, for $\rho > 0$, $\delta = 0$ [Eq. (3.130)], it turns out that $\sum_{i=1}^{N} \lambda_i = \nu$ [Eq. (3.127)]. Combining these and taking into account that all points that lie in the wrong side of the classifier correspond to $\xi_i > 0$, the total number of errors can, at most, be equal to $N\nu$. Thus, the error rate, P_e, on the training set is upper-bounded as

$$P_e \leq \nu. \tag{3.142}$$

Also, at the solution, from the constraints (3.127) and (3.126) we have that

$$\nu = \sum_{i=1}^{N} \lambda_i = \sum_{i=1}^{N_s} \lambda_i \leq \sum_{i=1}^{N_s} \frac{1}{N} \tag{3.143}$$

or

$$N\nu \leq N_s \tag{3.144}$$

Thus, the designer, by controlling the value of ν, may have a feeling for both the error rate on the training set and the number of the support vectors to result from the optimization process. The number of the support vectors, N_s, is very important for the performance of the classifier in practice. First, as we have already commented, it directly affects the computational load, since large N_s means that a large number of inner products are to be computed for classifying an unknown pattern. Second, as we will see at the end of Section 5.10, a large number of support vectors can limit the error performance of the SVM classifier when it is fed with data outside the training set (this is also known as the generalization performance of the classifier). For more on the ν-SVM, the interested reader can consult [Scho 00, Chan 01, Chen 03], where implementation issues are also discussed.

3.7.5 Support Vector Machines: A Geometric Viewpoint

In this section, we will close the circle around the SVM design task via a path that is very close to what we call common sense. Figure 3.14a illustrates the case of two separable data classes together with their respective convex hulls. The convex hull of a data set X is denoted as conv$\{X\}$ and is defined as the intersection of all convex sets (see Appendix C.4) containing X. It can be shown (e.g., [Luen 69]) that conv$\{X\}$ consists of all the convex combinations of the N elements of X. That is,

$$\text{conv}\{X\} = \Bigg\{ \boldsymbol{y} : \ \boldsymbol{y} = \sum_{i=1}^{N} \lambda_i \boldsymbol{x}_i : \ \boldsymbol{x}_i \in X, \\ \sum_{i=1}^{N} \lambda_i = 1, \ 0 \leq \lambda_i \leq 1, \ i = 1, 2, \ldots, N \Bigg\} \tag{3.145}$$

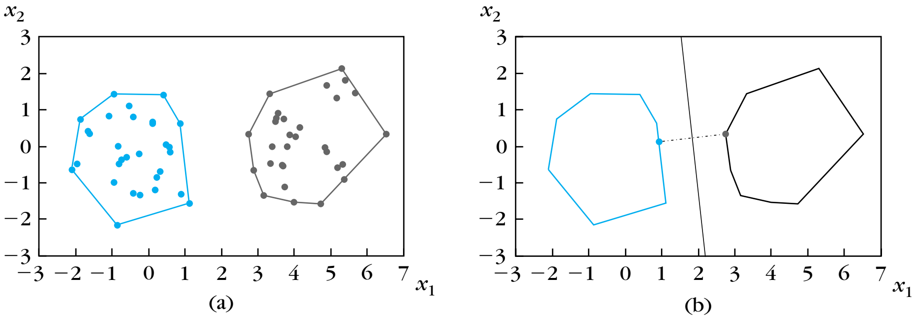

FIGURE 3.14

(a) A data set for two separable classes with the respective convex hulls. (b) The SVM optimal hyperplane bisects the segment joining the two nearest points between the convex hulls.

It turns out that solving the dual optimization problem in (3.87)–(3.89) for the linearly separable task results in the hyperplane that bisects the linear segment joining two *nearest points between the convex hulls* of the data classes [Figure 3.14b]. In other words, *searching for the maximum margin hyperplane is equivalent to searching for two nearest points between the corresponding convex hulls!* Let us investigate this a bit further.

Denote the convex hull of the vectors in class ω_1 as conv$\{X^+\}$ and the convex hull corresponding to class ω_2 as conv$\{X^-\}$. Following our familiar notation, any point in conv$\{X^+\}$, being a convex combination of all the points in ω_1, can be written as $\sum_{i:y_i=1}\lambda_i\boldsymbol{x}_i$, and any point in conv$\{X^-\}$ as $\sum_{i:y_i=-1}\lambda_i\boldsymbol{x}_i$, provided that λ_i fulfill the convexity constraints in (3.145). Searching for the closest points, it suffices to find the specific values of $\lambda_i,\ i = 1, 2, \ldots N$, such that

$$\min_{\boldsymbol{\lambda}} \ \|\sum_{i:y_i=1}\lambda_i\boldsymbol{x}_i - \sum_{i:y_i=-1}\lambda_i\boldsymbol{x}_i\|^2 \tag{3.146}$$

$$\text{subject to} \quad \sum_{i:y_i=1}\lambda_i = 1, \quad \sum_{i:y_i=-1}\lambda_i = 1 \tag{3.147}$$

$$\lambda_i \geq 0, \quad i = 1, 2, \ldots N \tag{3.148}$$

Elaborating the norm in (3.146) and reshaping the constraints in (3.147), we end up with the following equivalent formulation.

$$\text{minimize} \quad \sum_{i,j} y_i y_j \lambda_i \lambda_j \boldsymbol{x}_i^T\boldsymbol{x}_j \tag{3.149}$$

$$\text{subject to} \quad \sum_{i=1}^{N} y_i\lambda_i = 0, \quad \sum_{i=1}^{N}\lambda_i = 2 \tag{3.150}$$

$$\lambda_i \geq 0, \quad i = 1, 2, \ldots N \tag{3.151}$$

It takes a few lines of algebra to show that the optimization task in (3.87)–(3.89) results in the same solution as the task given in (3.149)–(3.151) ([Keer 00] and Problem 3.18). Having established the geometric interpretation of the SVM optimization task, any algorithm that has been developed to search for nearest points between convex hulls (e.g., [Gilb 66, Mitc 74, Fran 03]) can now, in principle, be mobilized to compute the maximum margin linear classifier.

It is now the turn of the nonseparable class problem to enter into the game, which, at this point becomes more exciting. Let us return to the ν-SVM formulation and reparameterize the primal problem in (3.119)–(3.122) by dividing the cost function by $\frac{\nu^2}{2}$ and the set of constraints by ν ([Crisp 99]). Obviously, this has no effect on the solution. The optimization task now becomes

$$\text{minimize} \quad J(\boldsymbol{w}, w_0, \boldsymbol{\xi}, \rho) = \|\boldsymbol{w}\|^2 - 2\rho + \mu \sum_{i=1}^{N} \xi_i \tag{3.152}$$

$$\text{subject to} \quad y_i[\boldsymbol{w}^T \boldsymbol{x}_i + w_0] \geq \rho - \xi_i, \quad i = 1, 2, \ldots, N \tag{3.153}$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, N \tag{3.154}$$

$$\rho \geq 0 \tag{3.155}$$

where $\mu = \frac{2}{\nu N}$ and we have kept, for economy, the same notation, although the parameters in (3.152)–(3.155) are scaled versions of those in (3.119)–(3.122). That is, $\boldsymbol{w} \rightarrow \frac{\boldsymbol{w}}{\nu}$, $w_0 \rightarrow \frac{w_0}{\nu}$, $\rho \rightarrow \frac{\rho}{\nu}$, $\xi_i \rightarrow \frac{\xi_i}{\nu}$. Hence, the solution obtained via (3.152)–(3.155) is a scaled version of the solution resulting via (3.119)–(3.122). The Wolfe dual representation of the primal problem in (3.152)–(3.155) is easily shown to be equivalent to

$$\text{minimize} \quad \sum_{i,j} y_i y_j \lambda_i \lambda_j \boldsymbol{x}_i^T \boldsymbol{x}_j \tag{3.156}$$

$$\text{subject to} \quad \sum_i y_i \lambda_i = 0, \ \sum_i \lambda_i = 2 \tag{3.157}$$

$$0 \leq \lambda_i \leq \mu, \ i = 1, 2, \ldots N \tag{3.158}$$

This set of relations is almost the same as those defining the nearest points between the convex hulls in the separable class case, (3.149)–(3.151), with a small, yet significant, difference. The Lagrange multipliers are bounded by μ, and for $\mu < 1$ they are not permitted to span their entire allowable range (i.e., [0, 1]).

3.7.6 Reduced Convex Hulls

The *reduced convex hull* (RCH) of a (finite) vector set, X, is denoted as $R(X, \mu)$ and is defined as the convex set

$$R(X,\mu) = \left\{ \boldsymbol{y} : \boldsymbol{y} = \sum_{i=1}^{N} \lambda_i \boldsymbol{x}_i : \ \boldsymbol{x}_i \in X, \right.$$

$$\left. \sum_{i=1}^{N} \lambda_i = 1, \ 0 \leq \lambda_i \leq \mu, \ i = 1, 2, \ldots, N \right\} \tag{3.159}$$

It is apparent from the previous definition that $R(X, 1) \equiv \text{conv}\{X\}$ and that

$$R(X,\mu) \subseteq \text{conv}\{X\} \tag{3.160}$$

Figure 3.15a shows the respective convex hulls for the case of two intersecting data classes. In Figure 3.15b, full lines indicate the convex hulls, conv$\{X^+\}$ and conv$\{X^-\}$, and the dotted lines the reduced convex hulls $R(X^+,\mu)$, $R(X^-,\mu)$, for two different values of $\mu = 0.4$ and $\mu = 0.1$, respectively. It is readily apparent that the smaller the value of μ, the smaller the size of the reduced convex hull. For small enough values of μ, one can make $R(X^+,\mu)$ and $R(X^-,\mu)$ nonintersecting. Adopting a procedure similar to the one that led to (3.149)–(3.151), it is not difficult to see that finding two nearest points between $R(X^+,\mu)$ and $R(X^-,\mu)$ results in the ν-SVM dual optimization task given in (3.156)–(3.158). Observe that the only difference between the latter and the task for the separable case, defined in (3.149)–(3.151), lies in the range in which the Lagrange multipliers are allowed to be. In the separable class case, the constraints (3.150) and (3.151) imply that $0 \leq \lambda_i \leq 1$, which in its geometric interpretation means that the full convex hulls are searched for the nearest points. In contrast, in the nonseparable class case a lower upper bound (i.e.,

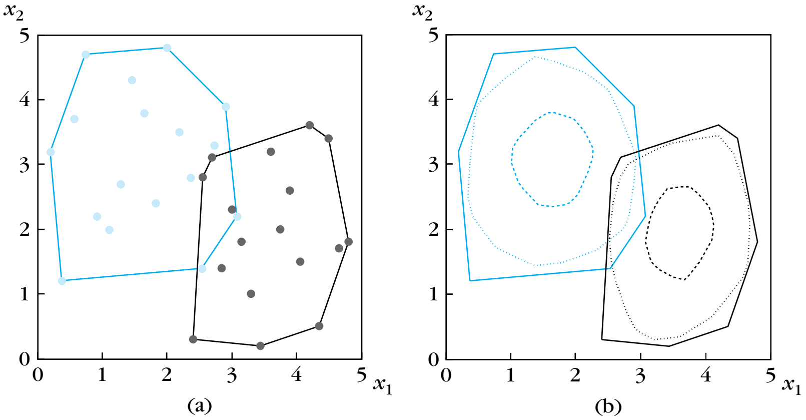

FIGURE 3.15

(a) Example of a data set with two intersecting classes and their respective convex hulls. (b) The convex hulls (indicated by full lines) and the resulting reduced convex hulls (indicated by dotted lines) corresponding to $\mu = 0.4$ and $\mu = 0.1$, respectively, for each class. The smaller the value of μ the smaller the RCH size.

$\mu \leq 1$) is imposed for the Lagrange multipliers. From the geometry point of view, this means that the search for the nearest points is limited within the respective reduced convex hulls.

Having established the geometric interpretation of the ν-SVM dual representation form, let us follow pure geometric arguments to draw the separating hyperplane. It is natural to choose it as the one bisecting the line segment joining two nearest points between the reduced convex hulls. Let $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ be two nearest points, with $\boldsymbol{x}^+ \in R(X^+, \mu)$ and $\boldsymbol{x}^- \in R(X^-, \mu)$. Also, let λ_i, $i = 1, 2, \ldots, N$, be the optimal set of multipliers resulting from the optimization task. Then, as can be deduced from Figure 3.16,

$$\boldsymbol{w} = \boldsymbol{x}^+ - \boldsymbol{x}^- = \sum_{i:y_i=1} \lambda_i \boldsymbol{x}_i - \sum_{i:y_i=-1} \lambda_i \boldsymbol{x}_i \tag{3.161}$$

$$= \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i \tag{3.162}$$

This is the same (within a scaling factor) as the $\boldsymbol{w}$ obtained from the KKT conditions associated with the ν-SVM task [Eq. (3.124)]. Thus, both approaches result in a separating hyperplane pointing in the same direction (recall from Section 3.2 that $\boldsymbol{w}$ defines the direction of the hyperplane). However, it is early to say that the two solutions are exactly the same. The hyperplane bisecting the line segment

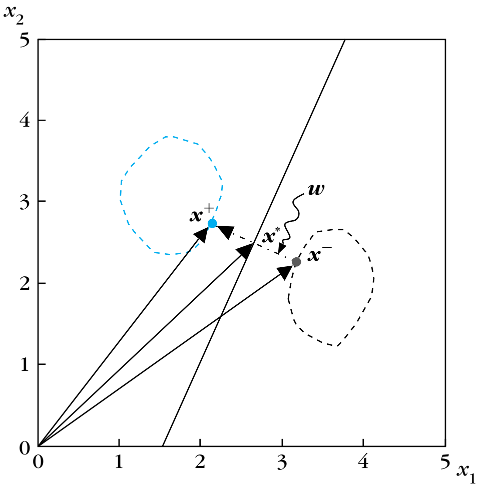

FIGURE 3.16

The optimal linear classifier resulting as the bisector of the segment joining the two closest points between the reduced convex hulls of the classes, for the case of the data set shown in Figure 3.15 and for $\mu = 0.1$.

joining the nearest points crosses the middle of this segment; that is, the point $\boldsymbol{x}^* = \frac{1}{2}(\boldsymbol{x}^+ + \boldsymbol{x}^-)$. Thus,

$$\boldsymbol{w}^T \boldsymbol{x}^* + w_0 = 0 \tag{3.163}$$

from which we get

$$w_0 = -\frac{1}{2}\boldsymbol{w}^T\left(\sum_{i:y_i=1} \lambda_i \boldsymbol{x}_i + \sum_{i:y_i=-1} \lambda_i \boldsymbol{x}_i\right) \tag{3.164}$$

This value for w_0 is, in general, different from the value resulting from the KKT conditions in (3.128). In conclusion, the geometric approach in the case of the nonseparable problem is equivalent to the ν-SVM formulation only to the extent that both approaches result in hyperplanes pointing in the same direction. However, note that the value in Eq. (3.128) can be obtained from that given in Eq. (3.164) in a trivial way [Crisp 99].

Remarks

- The choice of μ and consequently of $\nu = \frac{2}{\mu N}$ must guarantee that the feasible region is nonempty (i.e., a solution exists, Appendix C) and also that the solution is a nontrivial one (i.e., $\boldsymbol{w} \neq \boldsymbol{0}$). Let N^+ be the number of points in X^+ and N^- the number of points in X^-, where $N^+ + N^- = N$. Let $N_{\text{min}} = \min\{N^+, N^-\}$. Then it is readily seen from the crucial constraint $0 \leq \lambda_i \leq \mu$ and the fact that $\sum_i \lambda_i = 1$, in the definition of the reduced convex hull, that $\mu \geq \mu_{\text{min}} = \frac{1}{N_{\text{min}}}$. This readily suggests that ν cannot take any value but must be upper-bounded as

$$\nu \leq \nu_{\text{max}} = 2\frac{N_{\text{min}}}{N} \leq 1$$

Also, if the respective reduced convex hulls intersect, then the distance between the closest points is zero, leading to the trivial solution (Problem 3.19). Thus, nonintersection is guaranteed for some value μ_{max} such that $\mu \leq \mu_{\text{max}} \leq 1$, which leads to

$$\nu \geq \nu_{\text{min}} = \frac{2}{\mu_{\text{max}} N}$$

From the previous discussion it is easily deduced that for the feasible region to be nonempty it is required that

$$R(X^+, \mu_{\text{min}}) \cap R(X^-, \mu_{\text{min}}) = \emptyset$$

If $N^+ = N^- = \frac{N}{2}$, it is easily checked out that in this case each of the reduced convex hulls is shrunk to a point, which is the centroid of the respective class (e.g., $\frac{2}{N}\sum_{i:y_i=1}\boldsymbol{x}_i$). In other words, a solution is feasible if the centroids of the two classes do not coincide. Most natural!

- Computing the nearest points between reduced convex hulls turns out not to be a straightforward extension of the algorithms that have been developed for computing nearest points between convex hulls. This is because, for the latter case, such algorithms rely on the extreme points of the involved convex hulls. However, in this case, extreme points coincide with points in the original data sets, that is, X^+, X^-. This is not the case for the reduced convex hulls, where extreme points are *combinations* of points of the original data sets. The lower the value of μ, the higher the number of data samples that contribute to an extreme point in the respective reduced convex hull. A neat solution to this problem is given in [Mavr 05, Mavr 06, Mavr 07, Tao 04, Theo 07]. The developed nearest point algorithms are reported to offer computational savings, which in some cases can be significant, compared to the more classical algorithms in [Plat 99, Keer 01].

3.8 PROBLEMS

3.1 Explain why the perceptron cost function is a *continuous* piecewise linear function.

3.2 Show that if $\rho_k = \rho$ in the perceptron algorithm, the algorithm converges after $k_0 = \frac{\|\boldsymbol{w}(0)-\alpha\boldsymbol{w}^*\|}{\beta^2\rho(2-\rho)}$ steps, where $\alpha = \frac{\beta^2}{|\gamma|}$ and $\rho < 2$.

3.3 Show that the reward and punishment form of the perceptron algorithm converges in a finite number of iteration steps.

3.4 Consider a case in which class ω_1 consists of the two feature vectors $[0,0]^T$ and $[0,1]^T$ and class ω_2 of $[1,0]^T$ and $[1,1]^T$. Use the perceptron algorithm in its reward and punishment form, with $\rho = 1$ and $\boldsymbol{w}(0) = [0,0]^T$, to design the line separating the two classes.

3.5 Consider the two-class task of Problem 2.12 of the previous chapter with

$$\boldsymbol{\mu}_1^T = [1,1], \quad \boldsymbol{\mu}_2^T = [0,0], \quad \sigma_1^2 = \sigma_2^2 = 0.2$$

Produce 50 vectors from each class. To guarantee linear separability of the classes, disregard vectors with $x_1 + x_2 < 1$ for the [1, 1] class and vectors with $x_1 + x_2 > 1$ for the [0, 0] class. In the sequel, use these vectors to design a linear classifier using the perceptron algorithm of (3.21)–(3.23). After convergence, draw the corresponding decision line.

3.6 Consider once more the classification task of Problem 2.12. Produce 100 samples for each of the classes. Use these data to design a linear classifier via the LMS algorithm. Once all samples have been presented to the algorithm, draw the corresponding hyperplane to which the algorithm has converged. Use $\rho_k = \rho = 0.01$.

3.7 Show, using Kesler's construction, that the tth iteration step of the reward and punishment form of the perceptron algorithm (3.21)–(3.23), for an $\boldsymbol{x}_{(t)} \in \omega_i$, becomes

$$\begin{aligned} \boldsymbol{w}_i(t+1) &= \boldsymbol{w}_i(t) + \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{w}_i^T(t)\boldsymbol{x}_{(t)} \le \boldsymbol{w}_j^T(t)\boldsymbol{x}_{(t)}, j \neq i \\ \boldsymbol{w}_j(t+1) &= \boldsymbol{w}_j(t) - \rho \boldsymbol{x}_{(t)} \quad \text{if } \boldsymbol{w}_i^T(t)\boldsymbol{x}_{(t)} \le \boldsymbol{w}_j^T(t)\boldsymbol{x}_{(t)}, j \neq i \\ \boldsymbol{w}_k(t+1) &= \boldsymbol{w}_k(t), \quad \forall k \neq j \text{ and } k \neq i \end{aligned}$$

3.8 Show that the sum of error squares optimal weight vector tends asymptotically to the MSE solution.

3.9 Repeat Problem 3.6 and design the classifier using the sum of error squares criterion.

3.10 Show that the design of an M class linear, sum of error squares optimal, classifier reduces to M equivalent ones, with scalar desired responses.

3.11 Show that, if x, y are jointly Gaussian, the regression of y on x is given by

$$E[y|x] = \frac{\alpha\sigma_y x}{\sigma_x} + \mu_y - \frac{\alpha\sigma_y\mu_x}{\sigma_x}, \quad \text{where } \Sigma = \begin{bmatrix} \sigma_x^2 & \alpha\sigma_x\sigma_y \\ \alpha\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} \tag{3.165}$$

3.12 Let an M class classifier be given in the form of parameterized functions $g(\boldsymbol{x}; \boldsymbol{w}_k)$. The goal is to estimate the parameters $\boldsymbol{w}_k$ so that the outputs of the classifier give desired response values, depending on the class of $\boldsymbol{x}$. Assume that as $\boldsymbol{x}$ varies randomly in each class, the classifier outputs vary around the corresponding desired response values, according to a Gaussian distribution of known variance, assumed to be the same for all outputs. Show that in this case the sum of error squares criterion and the ML estimation result in identical estimates.

Hint: Take N training data samples of known class labels. For each of them form $y_i = g(\boldsymbol{x}_i; \boldsymbol{w}_k) - d_k^i$, where d_k^i is the desired response for the kth class of the ith sample. The y_i's are normally distributed with zero mean and variance σ^2. Form the likelihood function using the y_i's.

3.13 In a two-class problem, the Bayes optimal decision surface is given by $g(\boldsymbol{x}) = P(\omega_1|\boldsymbol{x}) - P(\omega_2|\boldsymbol{x}) = 0$. Show that if we train a decision surface $f(\boldsymbol{x}; \boldsymbol{w})$ in the MSE so as to give $+1(-1)$ for the two classes, respectively, this is equivalent to approximating $g(\cdot)$ in terms of $f(\cdot; \boldsymbol{w})$, in the MSE optimal sense.

3.14 Consider a two-class classification task with jointly Gaussian distributed feature vectors and with the same variance Σ in both classes. Design the linear MSE classifier and show that in this case the Bayesian classifier (Problem 2.11) and the resulting MSE one differ only in the threshold value. For simplicity, consider equiprobable classes.

Hint: To compute the MSE hyperplane $\boldsymbol{w}^T\boldsymbol{x}+w_0=0$, increase the dimension of $\boldsymbol{x}$ by one and show that the solution is provided by

$$\begin{bmatrix} R & E[\boldsymbol{x}] \\ E[\boldsymbol{x}]^T & 1 \end{bmatrix}\begin{bmatrix} \boldsymbol{w} \\ w_0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) \\ 0 \end{bmatrix}$$

Then relate R with Σ and show that the MSE classifier takes the form

$$(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^T\Sigma^{-1}\left(\boldsymbol{x}-\frac{1}{2}(\boldsymbol{\mu}_1+\boldsymbol{\mu}_2)\right)\geq 0$$

3.15 In an M class classification task, the classes can be linearly separated. Design M hyperplanes, so that hyperplane $g_i(\boldsymbol{x})=0$ leaves class ω_i on its positive side and the rest of the classes on its negative side. Demonstrate via an example, for example, $M=3$, that the partition of the space using this rule creates indeterminate regions (where no training data exist) for which more than one $g_i(\boldsymbol{x})$ is positive or all of them are negative.

3.16 Obtain the optimal line for the task of Example 3.5, via the KKT conditions. Restrict the search for the optimum among the lines crossing the origin.

3.17 Show that if the equality constraints (3.133)–(3.136) are substituted in the Lagrangian (3.123), the dual problem is described by the set of relations in (3.138)–(3.141).

3.18 Show that for the case of two linearly separable classes the hyperplane obtained as the SVM solution is the same as that bisecting the segment joining two closest points between the convex hulls of the classes.

3.19 Show that if ν in the ν-SVM is chosen smaller than $\nu_{\min}$, it leads to the trivial zero solution.

3.20 Show that if the soft margin SVM cost function is chosen to be

$$\frac{1}{2}||\boldsymbol{w}||^2+\frac{C}{2}\sum_{i=1}^{N}\xi_i^2$$

the task can be transformed into an instance of the class-separable case problem [Frie 98].

MATLAB PROGRAMS AND EXERCISES

Computer Programs

3.1 *Perceptron algorithm.* Write a MATLAB function for the perceptron algorithm. This will take as inputs: (a) a matrix X containing N l-dimensional column vectors, (b) an N-dimensional row vector y, whose ith component contains the class (−1 or +1) where the corresponding vector belongs, and (c) an initial value vector *w_ini* for the parameter vector. It returns the estimated parameter vector.

Solution

```
function w=perce(X,y,w_ini)
  [l,N]=size(X);
  max_iter=10000;   % Maximum allowable number of iterations
  rho=0.05;         % Learning rate
  w=w_ini;          % Initialization of the parameter vector
  iter=0;           % Iteration counter
  mis_clas=N;       % Number of misclassified vectors
  while (mis_clas>0) && (iter<max_iter)
    iter=iter+1;
    mis_clas=0;
    gradi=zeros(l,1);% Computation of the "gradient"
    % term
    for i=1:N
      if((X(:,i)'*w)*y(i)<0)
        mis_clas=mis_clas+1;
        gradi=gradi+rho*(-y(i)*X(:,i));
      end
    end
    w=w-rho*gradi; % Updating the parameter vector
  end
```

3.2 *Sum of error squares classifier:* Write a MATLAB function that implements the sum of error squares classifier for two classes. This will take as inputs: (a) a matrix X containing N l-dimensional column vectors, and (b) an N-dimensional row vector y whose ith component contains the class (−1 or +1) where the corresponding vector belongs. It returns the estimated parameter vector.

Solution

```
function w=SSErr(X,y)
  w=inv(X*X')*(X*y');
```

3.3 *LMS algorithm.* Write a MATLAB function for the LMS algorithm. This will take as inputs: (a) a matrix X containing N l-dimensional column vectors, (b) an N-dimensional row vector y whose ith component contains the class (−1 or +1) where the corresponding vector is assigned, and (c) an initial value vector *w_ini* for the parameter vector. It returns the estimated parameter vector.

Solution

```
function w=LMSalg(X,y,w_ini)
  [l,N]=size(X);
  rho=0.1;     % Learning rate initialization
  w=w_ini;    % Initialization of the parameter vector
  for i=1:N
    w=w+(rho/i)*(y(i)-X(:,i)'*w)*X(:,i);
  end
```

Computer Experiments

Note: In the sequel, it is advisable to use the command

```
randn('seed',0)
```

before generating the data sets, in order to initialize the Gaussian random number generator to 0 (or any other fixed number). This is important for the reproducibility of the results.

3.1 **a.** Generate two data sets $X1$ and $X_1^{'}$ of $N = 200$ two-dimensional vectors each. The first half of the vectors stem from the normal distribution with $m1 = [-5,\ 0]^T$ and $S1 = I$, while the second half of the vectors stem from the normal distribution with $m1 = [5,\ 0]^T$ and $S1 = I$, where I is the identity 2×2 matrix. Append each vector of both $X1$ and $X_1^{'}$ by inserting an additional coordinate, which is set equal to 1.

b. Apply the perceptron algorithm, the sum of error squares classifier, and the LMS algorithm on the previous data set, using various initial values for the parameter vector (where necessary).

c. Measure the performance of each one of the above methods on both $X1$ and $X1^{'}$.

d. Plot the data sets $X1$ and $X1^{'}$ as well as the line corresponding to the parameter vector w.

3.2 Repeat experiment 1 using now the sets $X2$ and $X2^{'}$ whose first half of their vectors stem from the normal distribution with $m1 = [-2,\ 0]^T$ and $S1 = I$, while the second half of their vectors stem from the normal distribution with $m1 = [2,\ 0]^T$ and $S1 = I$.

3.3 Repeat experiment 3.1 using now the sets $X3$ and $X3^{'}$ whose first half of the vectors stem from the normal distribution with $m1 = [-1,\ 0]^T$ and $S1 = I$, while the second half of the vectors stem from the normal distribution with $m1 = [1,\ 0]^T$ and $S1 = I$.

3.4 Discuss the results obtained by the previous experiments.

REFERENCES

[Allw 00] Allwein E.L., Schapire R.E., Singer Y. "Reducing multiclass to binary: aunifying approach for margin classifiers," *Journal of Machine Learning Research*, Vol. 1, pp. 113–141, 2000.

[Ande 82] Anderson J.A. "Logistic discrimination," in *Handbook of Statistics* (Krishnaiah R.P., Kanal L.N., eds.), North Holland, 1982.

[Baza 79] Bazaraa M.S., Shetty C.M. *Nonlinear Programming*, John Wiley & Sons, 1979.

[Bish 95] Bishop C. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[Bose 92] Bose B.E., Guyon I.M., Vapnik, V.N. "A training algorithm for optimal margin classifiers," *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pp. 144–152, Morgan Kaufman, 1992.

[Burg 97] Burges C.J.C., Schölkoff B. "Improving the accuracy and speed of support vectors learning machines," in *Advances in Neural Information Processing Systems 9* (Mozer M. Jordan M., Petsche T., eds.), pp. 375–381, MIT Press, 1997.

[Cao 06] Cao L.J., Keerthi S.S., Ong C.-J., Zhang J.Q., Periyathamby V., Fu X.J., Lee H.P. "Parallel sequential minimal optimization for the training of support vector machines," *IEEE Transactions on Neural Networks*, Vol. 17(4), pp. 1039–1049, 2006.

[Chan 00] Chang C.C., Hsu C.W., Lin C.J. "The analysis of decomposition methods for SVM," *IEEE Transactions on Neural Networks*, Vol. 11(4), pp. 1003–1008, 2000.

[Chan 01] Chang C.C., Lin C.J. "Training ν-support vector classifiers: Theory and algorithms," *Neural Computation* Vol. 13(9), pp. 2119–2147, 2001.

[Chen 06] Chen P.-H., Fan R.-E., Lin C.-J. "A study on SMO-type decomposition for support vector machines," *IEEE Transactions on Neural Networks*, Vol. 17(4), pp. 893–908, 2006.

[Chen 03] Chen P-H., Lin C.J., Schölkopf B. "A tutorial on ν-support vector machines," *Applied Stochastic Models in Business and Industry*, Vol. 21, pp. 111–136, 2005.

[Cid 99] Cid-Sueiro J., Arribas J.I., Urban-Munoz S., Figuieras-Vidal A.R. "Cost functions to estimate a-posteriori probabilities in multi-class problems," *IEEE Transactions on Neural Networks*, Vol. 10(3), pp. 645–656, 1999.

[Crisp 99] Crisp D.J., Burges C.J.C. "A geometric interpretation of ν-SVM classifiers," *Proceedings of Neural Information Processing*, Vol. 12, MIT Press, 1999.

[Diet 95] Dietterich T.G., Bakiri G. "Solving multi-class learning problems via error-correcting output codes," *Journal of Artificial Intelligence Research*, Vol. 2, pp. 263–286, 1995.

[Dong 05] Dong J.X., Krzyzak A., Suen C.Y. "Fast SVM training algorithm with decomposition on very large data sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27(4), pp. 603–618, 2005.

[Fei 06] Fei B., Liu J. "Binary tree of SVM: A new fast multi-class training and classification algorithm," *IEEE Transactions on Neural Networks*, Vol. 17(3), pp. 696–704, 2006.

[Fine 01] Fine S., Scheinberg K. "Efficient SVM training using low rank kernel representations," *Journal of Machine Learning Research*, Vol. 2, pp. 243–264, 2001.

[Flet 87] Fletcher R. *Practical Methods of Optimization*, 2nd ed., John Wiley & Sons, 1987.

[Fran 03] Franc V., Hlaváč V. "An iterative algorithm learning the maximal margin classifier," *Pattern Recognition*, Vol. 36, pp. 1985–1996, 2003.

[Frea 92] Frean M. "A thermal perceptron learning rule," *Neural Computation*, Vol. 4, pp. 946–957, 1992.

[Frie 98] FriessT.T. "The kernel adatron with bias and soft margin,"Technical Report,The University of Sheffield, Dept. of Automatic Control, England, 1998.

[Fuku 90] Fukunaga K. *Introduction to Statistical Pattern Recognition*, 2nd ed.,Academic Press, 1990.

[Gal 90] Gallant S.I. "Perceptron based learning algorithms," *IEEE Transactions on Neural Networks*,Vol. 1(2), pp. 179–191, 1990.

[Gema 92] Geman S.,Bienenstock E.,Doursat R. "Neural networks and the bias/variance dilemma," *Neural Computation*,Vol. 4, pp. 1–58, 1992.

[Gilb 66] Gilbert E.G. "An iterative procedure for computing the minimum of a quadratic form on a convex set," *SIAM Journal on Control*,Vol. 4(1), pp. 61–79, 1966.

[Guer 04] Guerrero-Curieses A., Cid-Sueiro J., Alaiz-Rodriguez R., Figueiras-Vidal A.R. "Local estimation of posterior class probabilities to minimize classification errors," *IEEE Transactions on Neural Networks*,Vol. 15(2), pp. 309–317, 2004.

[Hast 01] Hastie T.,Tibsharini R., Friedman J. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, Springer, 2001.

[Hayk 96] Haykin S. *Adaptive Filter Theory*, 3rd ed., Prentice Hall, 1996.

[Ho 65] Ho Y.H., Kashyap R.L. "An algorithm for linear inequalities and its applications," *IEEE Transactions on Electronic Computers*,Vol. 14(5), pp. 683–688, 1965.

[Hsu 02] Hsu C.W., Lin C.J. "A comparison of methods for multi-class SVM," *IEEE Transactions on Neural Networks*,Vol. 13, pp. 415–425, 2002.

[Hryc 92] Hrycej T., *Modular Learning in Neural Networks*, John Wiley & Sons, 1992.

[Hush 06] Hush D.,Kelly P. Scovel C.,Steinwart I. "QP algorithms with guaranteed accuracy and run time for support vector machines," *Journal of Machine Learning Research*,Vol. 7, pp. 733–769, 2006.

[Joac 98] JoachimsT. "Making large scale support vector machines practical," in*Advances in Kernel Methods*, (Schölkoph B., Burges C.J.C., Smola A. eds.), MIT Press, 1998.

[Kalou 93] Kalouptsidis N.,Theodoridis S. *Adaptive System Identification and Signal Processing Algorithms*, Prentice Hall, 1993.

[Keer 00] Keerthi S.S., Shevade S.K., Bhattacharyya C., Murthy K.R.K. "A fast iterative nearest point algorithm for support vector machine classifier design," *IEEE Transactions on Neural Networks*,Vol. 11(1), pp. 124–136, 2000.

[Keer 01] Keerthi S.S.,Shevade S.K.,Bhattacharyya C.,Murth K.R.K. "Improvements to Platt's SMO algorithm for SVM classifier design," *Neural Computation*,Vol. 13, pp. 637–649, 2001.

[Liu 06] Liu Y., You Z., Cao L. "A novel and quick SVM-based multi-class classifier," *Pattern Recognition*,Vol. 39(11), pp. 2258–2264, 2006.

[Luen 69] Luenberger D.G. *Optimization by Vector Space Methods*, JohnWiley & Sons, New York, 1969.

[McLa 92] McLachlan G. J. *DiscriminantAnalysis and Statistical Pattern Recognition*, JohnWiley & Sons, 1992.

[Matt 99] Mattera D., Palmieri F., Haykin S. "An explicit algorithm for training support vector machines," *IEEE Signal Processing Letters*,Vol. 6(9), pp. 243–246, 1999.

[Mavr 07] Mavroforakis M., Sdralis M., Theodoridis S. "A geometric nearest point algorithm for the efficient solution of the SVM classification task," *IEEE Transactions on Neural Networks*, Vol. 18(5), pp. 1545–1550, 2007.

[Mavr 06] Mavroforakis M., Theodoridis S. "A geometric approach to Support Vector Machine (SVM) classification," *IEEE Transactions on Neural Networks*, Vol. 17(3), pp. 671–682, 2006.

[Mavr 05] Mavroforakis M., Theodoridis S. "Support Vector Machine classification through geometry," *Proceedings of the XII European Signal Processing Conference (EUSIPCO)*, Antalya, Turkey, 2005.

[Mitc 74] Mitchell B.F., Demyanov V.F., Malozemov V.N. "Finding the point of a polyhedron closest to the origin," *SIAM Journal on Control*, Vol. 12, pp. 19–26, 1974.

[Min 88] Minsky M. L., Papert S.A. *Perceptrons*, expanded edition, MIT Press, MA, 1988.

[Muse 97] Muselli M. "On convergence properties of pocket algorithm," *IEEE Transactions on Neural Networks*, Vol. 8(3), pp. 623–629, 1997.

[Navi 01] Navia-Vasquez A., Perez-Cuz F., Artes-Rodriguez A., Figueiras-Vidal A. "Weighted least squares training of support vector classifiers leading to compact and adaptive schemes," *IEEE Transactions on Neural Networks*, Vol. 12(5), pp. 1047–1059, 2001.

[Nguy 06] Nguyen D., Ho T. "A bottom-up method for simplifying support vector solutions," *IEEE Transactions on Neural Networks*, Vol. 17(39), pp. 792–796, 2006.

[Osun 97] Osuna E., Freund R., Girosi F., "An improved training algorithm for support vector machines," *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pp. 276–285, Amelia Island, FL, 1997.

[Papo 91] Papoulis A. *Probability, Random Variables and Stochastic Processes*, 3rd ed., McGraw-Hill, 1991.

[Pear 90] Pearlmutter B., Hampshire J. "Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function," *Proceedings Connectionists Models Summer School*, Morgan Kauffman, 1990.

[Plat 00] Platt J.C., Cristianini N., Shawe-Taylor J. "Large margin DAGs for the multiclass classification," in *Advances in Neural Information Processing*, (Smola S.A., Leen T.K., Müller K.R., eds.), Vol. 12, pp. 547–553, MIT Press, 2000.

[Plat 99] Platt J. "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Learning* (Scholkopf B., Burges C.J.C., Smola A. J., eds), pp. 185–208, MIT Press, 1999.

[Platt 98] Platt J. "Sequential minimal optimization: A fast algorithm for training support vector machines," *Technical Report, Microsoft Research*, MSR-TR-98-14, April 21, 1998.

[Poul 95] Poulard H., "Barycentric correction procedure: A fast method of learning threshold units," *Proc. WCNN '95*, Vol. 1, Washington, DC, pp. 710–713, July, 1995.

[Rich 91] Richard M.D., Lippmann R.P. "Neural network classifiers estimate Bayesian a posteriori probabilities," *Neural Computation*, Vol. 3, pp. 461–483, 1991.

[Rifk 04] Rifkiy R., Klautau A. "In defense of one-vs-all classification," *Journal of Machine Learning Research*, Vol. 5, pp. 101–141, 2004.

[Robb 51] Robbins H., Monro S. "A stochastic approximation method," *Annals of Mathematical Statistics*, Vol. 22, pp. 400–407, 1951.

[Rose 58] Rosenblatt F. "The perceptron: only A probabilistic model for information storage and organization in the brain," *Psychological Review*, Vol. 65, pp. 386–408, 1958.

[Scho 00] Schölkoph B., Smola A.J., Williamson R.C., Bartlett P.L. "New support vector algorithms," *Neural Computation*, Vol. 12, pp. 1207–1245, 2000.

[Tao 04] Tao Q., Wu G.-W., Wang J. "A generalized S-K algorithm for learning ν-SVM classifiers," *Pattern Recognition Letters*, Vol. 25(10), pp. 1165–1171, 2004.

[Theo 07] Theodoridis S., Mavroforakis M. "Reduced convex hulls: A geometric approach to support vector machines," *IEEE Signal Processing Magazine*, Vol. 24(3), pp. 119–122, 2007.

[Tou 74] Tou J., Gonzalez R.C. *Pattern Recognition Principles*, Addison-Wesley, 1974.

[Tsan 06] Tsang I.W.-H., Kwok J.T. -Y., Zurada J.M. "Generalized core vector machines," *IEEE Transactions on Neural Networks*, Vol. 17(5), pp. 1126–1140, 2006.

[Vapn 98] Vapnik V.N. *Statistical Learning Theory*, John Wiley & Sons, 1998.

[Widr 60] Widrow B., Hoff M.E., Jr. "Adaptive switching circuits," *IRE WESCON Convention Record*, pp. 96–104, 1960.

[Widr 90] Widrow B., Lehr M.A. "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, Vol. 78(9), pp. 1415–1442, 1990.

[Yee 96] Yee T., Wild C. "Vector generalized additive models," *Journal of the Royal Statistical Society, Series B*, Vol. 58, pp. 481–493, 1996.

[Zhou 08] Zhou J., Peng H., Suen C.Y. "Data-driven decomposition for multi-class classification," *Pattern Recognition*, Vol. 41, pp. 67–76, 2008.