2

Linear Learning Machines

In supervised learning, the learning machine is given a training set of examples (or inputs) with associated labels (or output values). Usually the examples are in the form of attribute vectors, so that the input space is a subset of $\mathbb{R}^n$. Once the attribute vectors are available, a number of sets of hypotheses could be chosen for the problem. Among these, linear functions are the best understood and simplest to apply. Traditional statistics and the classical neural networks literature have developed many methods for discriminating between two classes of instances using linear functions, as well as methods for interpolation using linear functions. These techniques, which include both efficient iterative procedures and theoretical analysis of their generalisation properties, provide the framework within which the construction of more complex systems will be developed in the coming chapters. In this chapter we review results from the literature that will be relevant to the study of Support Vector Machines. We will first discuss algorithms and issues of classification, and then we will move on to the problem of regression. Throughout this book, we will refer to learning machines using hypotheses that form linear combinations of the input variables as linear learning machines.

Importantly, we will show that in most cases such machines can be represented in a particularly useful form, which we will call the dual representation. *This fact will prove crucial in later chapters. The important notions of* margin *and* margin distribution *are also introduced in this chapter. The classification results are all introduced for the binary or two-class case, and at the end of the chapter it is shown how to generalise them to multiple classes.*

2.1 Linear Classification

Binary classification is frequently performed by using a real-valued function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ in the following way: the input $\mathbf{x} = (x_1, \dots, x_n)'$ is assigned to the positive class, if $f(\mathbf{x}) \geq 0$, and otherwise to the negative class. We consider the case where $f(\mathbf{x})$ is a linear function of $\mathbf{x} \in X$, so that it can be written as

$$\begin{aligned} f(\mathbf{x}) &= \langle \mathbf{w} \cdot \mathbf{x} \rangle + b \\ &= \sum_{i=1}^{n} w_i x_i + b \end{aligned}$$

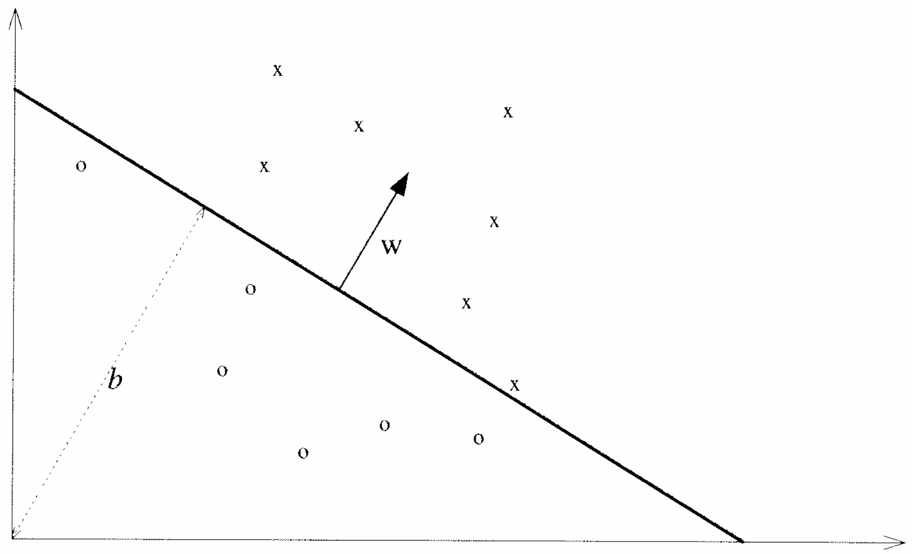

Figure 2.1: A separating hyperplane $(\mathbf{w}, b)$ for a two dimensional training set

where $(\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters that control the function and the decision rule is given by $\operatorname{sgn}(f(\mathbf{x}))$, where we will use the convention that $\operatorname{sgn}(0) = 1$. The learning methodology implies that these parameters must be learned from the data.

A geometric interpretation of this kind of hypothesis is that the input space X is split into two parts by the hyperplane defined by the equation $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ (see Figure 2.1). A hyperplane is an affine subspace of dimension $n - 1$ which divides the space into two half spaces which correspond to the inputs of the two distinct classes. For example in Figure 2.1 the hyperplane is the dark line, with the positive region above and the negative region below. The vector $\mathbf{w}$ defines a direction perpendicular to the hyperplane, while varying the value of b moves the hyperplane parallel to itself. It is therefore clear that a representation involving $n+1$ free parameters is necessary, if one wants to represent all possible hyperplanes in $\mathbb{R}^n$.

Both statisticians and neural network researchers have frequently used this simple kind of classifier, calling them respectively *linear discriminants* and *perceptrons*. The theory of linear discriminants was developed by Fisher in 1936, while neural network researchers studied perceptrons in the early 1960s, mainly due to the work of Rosenblatt. We will refer to the quantities $\mathbf{w}$ and b as the *weight vector* and *bias*, terms borrowed from the neural networks literature. Sometimes $-b$ is replaced by θ, a quantity known as the *threshold*.

As we are studying supervised learning from examples, we first introduce some notation we will be using throughout the book to refer to inputs, outputs, training sets, and so on.

Definition 2.1 We typically use X to denote the input space and Y to denote the output domain. Usually we will have $X \subseteq \mathbb{R}^n$, while for binary classification $Y = \{-1, 1\}$, for m-class classification $Y = \{1, 2, \ldots, m\}$, and for regression $Y \subseteq \mathbb{R}$. A *training set* is a collection of *training examples*, which are also called *training data*. It is usually denoted by

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)) \subseteq (X \times Y)^\ell,$$

where ℓ is the number of examples. We refer to the $\mathbf{x}_i$ as *examples* or *instances* and the y_i as their *labels*. The training set S is *trivial* if the labels of all the examples are equal. Note that if X is a vector space, the input vectors are column vectors as are the weight vectors. If we wish to form a row vector from $\mathbf{x}_i$ we can take the transpose $\mathbf{x}_i'$.

Several simple iterative algorithms optimising different cost functions were introduced in the 1960s for separating points from two populations by means of a hyperplane. In the following subsections we will review some of the best-known, and will highlight some of their most interesting properties. The case of the perceptron is particularly interesting not only for historical reasons, but also because, even in such a simple system, we can already find most of the central concepts that we will need for the theory of Support Vector Machines. Note that some algorithms, such as least squares, have been used both for regression and for classification. In order to avoid duplication, we will describe them in the regression section.

2.1.1 Rosenblatt's Perceptron

The first iterative algorithm for learning linear classifications is the procedure proposed by Frank Rosenblatt in 1956 for the perceptron. The algorithm created a great deal of interest when it was first introduced. It is an 'on-line' and 'mistake-driven' procedure, which starts with an initial weight vector $\mathbf{w}_0$ (usually $\mathbf{w}_0 = \mathbf{0}$ the all zero vector) and adapts it each time a training point is misclassified by the current weights. The algorithm is shown in Table 2.1. The algorithm updates the weight vector and bias directly, something that we will refer to as the primal form in contrast to an alternative dual representation which we will introduce below.

This procedure is guaranteed to converge provided there exists a hyperplane that correctly classifies the training data. In this case we say that the data are *linearly separable*. If no such hyperplane exists the data are said to be nonseparable. Below we will show that the number of iterations depends on a quantity called the margin. This quantity will play a central role throughout the book and so we will introduce a more formal definition here.

Definition 2.2 We define the *(functional) margin of an example* $(\mathbf{x}_i, y_i)$ *with respect to a hyperplane* $(\mathbf{w}, b)$ to be the quantity

$$\gamma_i = y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b).$$

```
Given a linearly separable training set $S$ and learning rate $\eta \in \mathbb{R}^+$
$\mathbf{w}_0 \leftarrow \mathbf{0}$; $b_0 \leftarrow 0$; $k \leftarrow 0$
$R \leftarrow \max_{1 \le i \le \ell} \|\mathbf{x}_i\|$
repeat
        for $i = 1$ to $\ell$
                if $y_i(\langle \mathbf{w}_k \cdot \mathbf{x}_i \rangle + b_k) \le 0$ then
                        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta y_i \mathbf{x}_i$
                        $b_{k+1} \leftarrow b_k + \eta y_i R^2$
                        $k \leftarrow k + 1$
                end if
        end for
until no mistakes made within the for loop
return $(\mathbf{w}_k, b_k)$ where $k$ is the number of mistakes
```

Table 2.1: **The Perceptron Algorithm (primal form)**

Note that $\gamma_i > 0$ implies correct classification of $(\mathbf{x}_i, y_i)$. The *(functional) margin distribution of a hyperplane* $(\mathbf{w}, b)$ *with respect to a training set* S is the distribution of the margins of the examples in S. We sometimes refer to the minimum of the margin distribution as the *(functional) margin of a hyperplane* $(\mathbf{w}, b)$ *with respect to a training set* S. In both definitions if we replace functional margin by *geometric margin* we obtain the equivalent quantity for the normalised linear function $\left(\frac{1}{\|\mathbf{w}\|}\mathbf{w}, \frac{1}{\|\mathbf{w}\|}b\right)$, which therefore measures the Euclidean distances of the points from the decision boundary in the input space. Finally, the *margin of a training set* S is the maximum geometric margin over all hyperplanes. A hyperplane realising this maximum is known as a *maximal margin hyperplane.* The size of its margin will be positive for a linearly separable training set.

Figure 2.2 shows the geometric margin at two points with respect to a hyperplane in two dimensions. The geometric margin will equal the functional margin if the weight vector is a unit vector. Figure 2.3 shows the margin of a training set.

Theorem 2.3 *(Novikoff) Let* S *be a non-trivial training set, and let*

$$R = \max_{1 \le i \le \ell} \|\mathbf{x}_i\|.$$

Suppose that there exists a vector $\mathbf{w}_{\text{opt}}$ *such that* $\|\mathbf{w}_{\text{opt}}\| = 1$ *and*

$$y_i(\langle \mathbf{w}_{\text{opt}} \cdot \mathbf{x}_i \rangle + b_{\text{opt}}) \ge \gamma$$

for $1 \le i \le \ell$. *Then the number of mistakes made by the on-line perceptron algorithm on* S *is at most*

$$\left(\frac{2R}{\gamma}\right)^2.$$

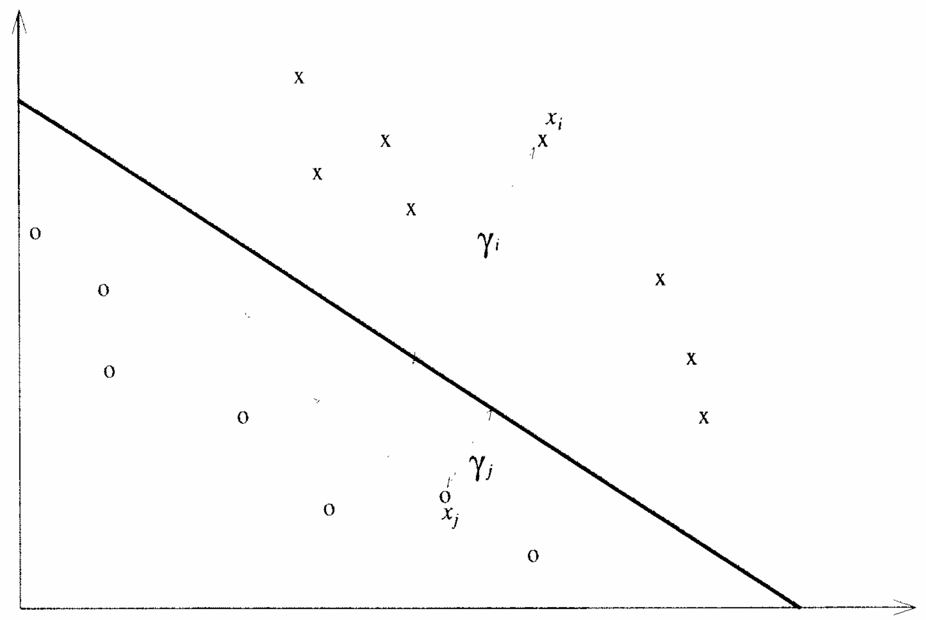

Figure 2.2: The geometric margin of two points

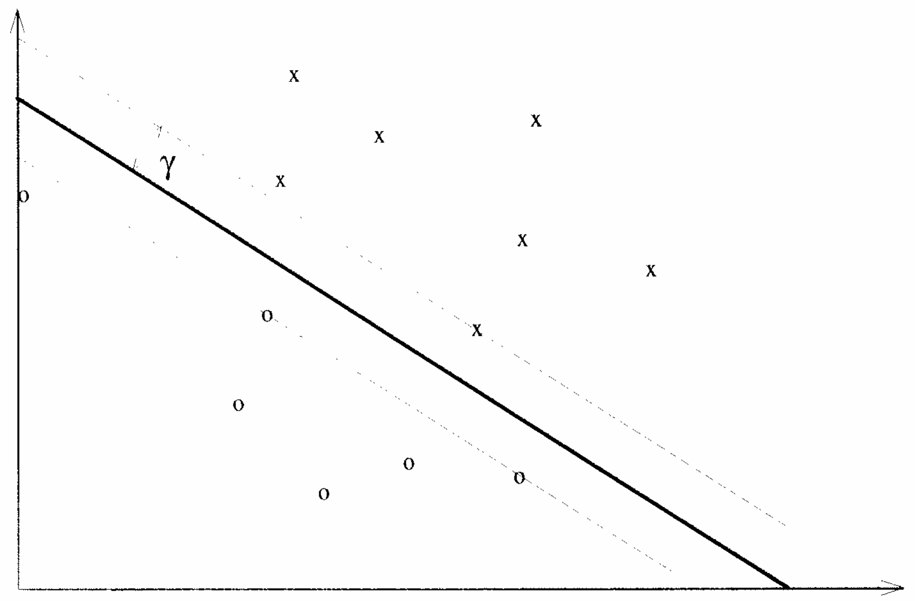

Figure 2.3: The margin of a training set

Proof For the analysis we augment the input vectors by an extra coordinate with value R. We denote the new vector by $\hat{\mathbf{x}}_i = (\mathbf{x}_i', R)'$, where $\mathbf{x}'$ denotes the transpose of $\mathbf{x}$. Similarly we add an extra coordinate to the weight vector $\mathbf{w}$ by incorporating the bias b, to form the augmented weight vector $\hat{\mathbf{w}} = (\mathbf{w}', b/R)'$. The algorithm starts with an augmented weight vector $\hat{\mathbf{w}}_0 = \mathbf{0}$ and updates it at each mistake. Let $\hat{\mathbf{w}}_{t-1}$ be the augmented weight vector prior to the tth mistake. The tth update is performed when

$$y_i \langle \hat{\mathbf{w}}_{t-1} \cdot \hat{\mathbf{x}}_i \rangle = y_i (\langle \mathbf{w}_{t-1} \cdot \mathbf{x}_i \rangle + b_{t-1}) \leq 0$$

where $(\mathbf{x}_i, y_i) \in S$ is the point incorrectly classified by $\hat{\mathbf{w}}_{t-1} = \left(\mathbf{w}_{t-1}', b_{t-1}/R\right)'$. The update is the following:

$$\hat{\mathbf{w}}_t = \left(\mathbf{w}_t', b_t/R\right)' = \left(\mathbf{w}_{t-1}', b_{t-1}/R\right)' + \eta y_i \left(\mathbf{x}_i', R\right)' = \hat{\mathbf{w}}_{t-1} + \eta y_i \hat{\mathbf{x}}_i,$$

where we have used the fact that

$$\begin{aligned} b_t/R &= b_{t-1}/R + \eta y_i R \\ \text{since } b_t &= b_{t-1} + \eta y_i R^2. \end{aligned}$$

The derivation

$$\langle \hat{\mathbf{w}}_t \cdot \hat{\mathbf{w}}_{\text{opt}} \rangle = \langle \hat{\mathbf{w}}_{t-1} \cdot \hat{\mathbf{w}}_{\text{opt}} \rangle + \eta y_i \langle \hat{\mathbf{x}}_i \cdot \hat{\mathbf{w}}_{\text{opt}} \rangle \geq \langle \hat{\mathbf{w}}_{t-1} \cdot \hat{\mathbf{w}}_{\text{opt}} \rangle + \eta \gamma$$

implies (by induction) that

$$\langle \hat{\mathbf{w}}_t \cdot \hat{\mathbf{w}}_{\text{opt}} \rangle \geq t \eta \gamma.$$

Similarly, we have

$$\begin{aligned} \|\hat{\mathbf{w}}_t\|^2 &= \|\hat{\mathbf{w}}_{t-1}\|^2 + 2\eta y_i \langle \hat{\mathbf{w}}_{t-1} \cdot \hat{\mathbf{x}}_i \rangle + \eta^2 \|\hat{\mathbf{x}}_i\|^2 \\ &\leq \|\hat{\mathbf{w}}_{t-1}\|^2 + \eta^2 \|\hat{\mathbf{x}}_i\|^2 \\ &\leq \|\hat{\mathbf{w}}_{t-1}\|^2 + \eta^2 \left(\|\mathbf{x}_i\|^2 + R^2\right) \\ &\leq \|\hat{\mathbf{w}}_{t-1}\|^2 + 2\eta^2 R^2, \end{aligned}$$

which implies that

$$\|\hat{\mathbf{w}}_t\|^2 \leq 2t\eta^2 R^2.$$

The two inequalities combined give the 'squeezing' relations

$$\|\hat{\mathbf{w}}_{\text{opt}}\| \sqrt{2t}\eta R \geq \|\hat{\mathbf{w}}_{\text{opt}}\| \|\hat{\mathbf{w}}_t\| \geq \langle \hat{\mathbf{w}}_t, \hat{\mathbf{w}}_{\text{opt}} \rangle \geq t\eta\gamma,$$

which together imply the bound

$$t \leq 2\left(\frac{R}{\gamma}\right)^2 \|\hat{\mathbf{w}}_{\text{opt}}\|^2 \leq \left(\frac{2R}{\gamma}\right)^2,$$

since $b_{\text{opt}} \leq R$ for a non-trivial separation of the data, and hence

$$\|\hat{\mathbf{w}}_{\text{opt}}\|^2 \leq \|\mathbf{w}_{\text{opt}}\|^2 + 1 = 2.$$

□

Remark 2.4 The theorem is usually given for zero bias and the bound is a factor 4 better in this case. However, the bias is updated in the perceptron algorithm and if the standard update is made (without the R^2 factor) the number of iterations depends on the margin of the augmented (including bias) weight training set. This margin is always less than or equal to γ and can be very much smaller. It will equal γ when $b_{\text{opt}} = 0$, and so the bound using the augmented margin will be a factor 4 better, though in this case the factor of 2 introduced in the last line of our proof can be avoided, making our bound only a factor 2 worse. In contrast, for cases where $|b_{\text{opt}}| = O(R)$, with $R > 1$, the bound obtained with the augmented training set will be a factor $O(R^2)$ worse than our bound.

Remark 2.5 The critical quantity in the bound is the square of the ratio of the radius of the ball containing the data and the margin of the separating hyperplane. This ratio is invariant under a positive rescaling of the data, and it is clear that rescaling will not affect the number of iterations taken by the algorithm, though it is at first counter-intuitive that this number does not depend on the learning rate. The reason is that there is a degree of freedom in the description of a thresholded linear function, as a positive rescaling of both weights and bias does not change its classification. Later we will use this fact to define the canonical maximal margin hyperplane with respect to a separable training set by fixing the margin equal to 1 and minimising the norm of the weight vector. The resulting value of the norm is inversely proportional to the margin.

The theorem proves that the algorithm converges in a finite number of iterations provided its margin is positive. Just iterating several times on the same sequence S, after a number of mistakes bounded by $\left(\frac{2R}{\gamma}\right)^2$ the perceptron algorithm will find a separating hyperplane and hence halt, provided one exists.

In cases where the data are not linearly separable, the algorithm will not converge: if executed iteratively on the sequence S it will continue to oscillate, changing hypothesis $\mathbf{w}_i$ every time it finds a misclassified point. However, a theorem similar to Novikoff's exists, bounding the number of errors made during one iteration. It uses a different measure of the margin distribution, a measure that will play an important role in later chapters. Intuitively, it generalises the notion of margin to account for a more global property of the training sample, using the margins achieved by training points other than those closest to the hyperplane. This measure of the margin distribution can also be used to measure the amount of 'non-separability' of the sample.

Definition 2.6 Fix a value $\gamma > 0$, we can define the *margin slack variable of an example* $(\mathbf{x}_i, y_i)$ *with respect to the hyperplane* $(\mathbf{w}, b)$ *and target margin* γ as

$$\xi\left((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma\right) = \xi_i = \max\left(0, \gamma - y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right)\right).$$

Informally this quantity measures how much a point fails to have a margin of γ from the hyperplane. If $\xi_i > \gamma$, then $\mathbf{x}_i$ is misclassified by $(\mathbf{w}, b)$. The norm $\|\xi\|_2$

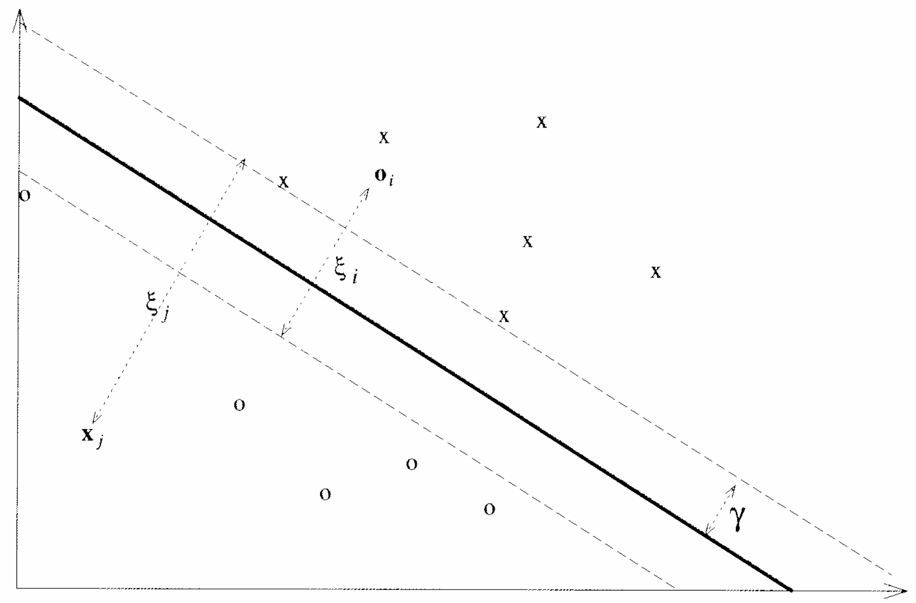

Figure 2.4: The slack variables for a classification problem

measures the amount by which the training set fails to have margin γ, and takes into account any misclassifications of the training data.

Figure 2.4 shows the size of the margin slack variables for two misclassified points for a hyperplane with unit norm. All of the other points in the figure have their slack variable equal to zero since they have a (positive) margin of more than γ.

Theorem 2.7 *(Freund and Schapire) Let S be a non-trivial training set with no duplicate examples, with $\|\mathbf{x}_i\| \le R$. Let $(\mathbf{w}, b)$ be any hyperplane with $\|\mathbf{w}\| = 1$, let $\gamma > 0$ and define*

$$D = \sqrt{\sum_{i=1}^{\ell} \xi_i^2} = \sqrt{\sum_{i=1}^{\ell} \xi\left((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma\right)^2}.$$

Then the number of mistakes in the first execution of the for *loop of the perceptron algorithm of Table 2.1 on S is bounded by*

$$\left(\frac{2(R+D)}{\gamma}\right)^2.$$

Proof The proof defines an extended input space parametrised by Δ in which there is a hyperplane with margin γ that has the same functionality as $(\mathbf{w}', b)'$ on unseen data. We can then apply Theorem 2.3 in the extended space. Finally,

optimising the choice of Δ will give the result. The extended input space has an extra coordinate for each training example. The new entries for example $\mathbf{x}_i$ are all zero except for the value Δ in the ith additional coordinate. Let $\tilde{\mathbf{x}}_i$ denote this extended vector and $\tilde{S}$ the corresponding training set. We now extend $\mathbf{w}$ with the value $y_i\xi_i/\Delta$ in the ith additional entry to give the vector $\tilde{\mathbf{w}}$. Observe that

$$y_i\left(\langle \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_i \rangle + b\right) = y_i\left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b\right) + \xi_i \ge \gamma,$$

showing that $(\tilde{\mathbf{w}}', b)'$ has margin γ on $\tilde{S}$. Note, however, that $\|\tilde{\mathbf{w}}\| = \sqrt{1 + D^2/\Delta^2}$, so that the geometric margin $\tilde{\gamma}$ is reduced by this factor. Since the extended training examples have non-zero entries in different coordinates, running the perceptron algorithm for the first *for* loop on $\tilde{S}$ has the same effect as running it on S, and we can bound the number of mistakes by Theorem 2.3 by

$$\left(\frac{2\tilde{R}}{\tilde{\gamma}}\right)^2 = \frac{4\left(R^2 + \Delta^2\right)\left(1 + D^2/\Delta^2\right)}{\gamma^2}.$$

The bound is optimised by choosing $\Delta = \sqrt{RD}$ giving the required result. $\square$

Remark 2.8 The reason we can only apply the theorem for the first iteration of the *for* loop is that after a training example $\tilde{\mathbf{x}}_i$ has been used to update the weight vector in the extended space, the ith additional coordinate of the weight vector will have a non-zero entry, which will affect the evaluation of $\tilde{\mathbf{x}}_i$ when it is processed in subsequent iterations. One could envisage an adaptation of the perceptron algorithm that would include these additional coordinates, though the value of Δ would have to either be a parameter or be estimated as part of the computation.

Remark 2.9 Since D can be defined for any hyperplane, the bound of the theorem does not rely on the data being linearly separable. The problem of finding the linear separation of non-separable data with the smallest number of misclassifications is NP-complete. A number of heuristics have been proposed for this problem, for example the pocket algorithm outputs the $\mathbf{w}$ that survived for the longest number of iterations. The extension suggested in the previous remark could be used to derive a version of the perceptron algorithm for non-separable data.

It is important to note that the perceptron algorithm works by adding misclassified positive training examples or subtracting misclassified negative ones to an initial arbitrary weight vector. Without loss of generality, we have assumed that the initial weight vector is the zero vector, and so the final hypothesis will be a linear combination of the training points:

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i,$$

```
Given training set $S$
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$; $b \leftarrow 0$
$R \leftarrow \max_{1 \le i \le \ell} \|\mathbf{x}_i\|$
repeat
    for $i = 1$ to $\ell$          w x +b
        if $y_i \left( \sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle + b \right) \le 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$
            $b \leftarrow b + y_i R^2$
        end if
    end for
until no mistakes made within the *for* loop
return $(\boldsymbol{\alpha}, b)$ to define function $h(\mathbf{x})$ of equation (2.1)
```

Table 2.2: **The Perceptron Algorithm (dual form)**

where, since the sign of the coefficient of $\mathbf{x}_i$ is given by the classification y_i, the α_i are positive values proportional to the number of times misclassification of $\mathbf{x}_i$ has caused the weight to be updated. Points that have caused fewer mistakes will have smaller α_i, whereas difficult points will have large values. This quantity is sometimes referred to as the *embedding strength* of the pattern $\mathbf{x}_i$, and will play an important role in later chapters. Once a sample S has been fixed, one can think of the vector $\boldsymbol{\alpha}$ as alternative representation of the hypothesis in different or dual coordinates. This expansion is however not unique: different $\boldsymbol{\alpha}$ can correspond to the same hypothesis $\mathbf{w}$. Intuitively, one can also regard α_i as an indication of the information content of the example $\mathbf{x}_i$. In the case of non-separable data, the coefficients of misclassified points grow indefinitely.

The decision function can be rewritten in dual coordinates as follows:

$$\begin{aligned} h(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b\right) \\ &= \operatorname{sgn}\left(\left\langle \sum_{j=1}^{\ell} \alpha_j y_j \mathbf{x}_j \cdot \mathbf{x} \right\rangle + b \right) \qquad (2.1) \\ &= \operatorname{sgn}\left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x} \rangle + b \right), \end{aligned}$$

and the perceptron algorithm can also be expressed entirely in this dual form as shown in Table 2.2. Note that the learning rate only changes the scaling of the hyperplanes, it does not affect the algorithm with a zero starting vector and so we have no longer included it.

This alternative formulation of the perceptron algorithm and its decision function has many interesting properties. For example, the fact that the points that are harder to learn have larger α_i can be used to rank the data according to their information content. Indeed, in the analysis of the simple perceptron

algorithm we have already found many of the important concepts that will be used in the theory of Support Vector Machines: the margin, the margin distribution, and the dual representation.

Remark 2.10 Since the number of updates equals the number of mistakes and each update causes 1 to be added to exactly one of its components, the 1-norm of the vector $\boldsymbol{\alpha}$ satisfies

$$\|\boldsymbol{\alpha}\|_1 \le \left(\frac{2R}{\gamma}\right)^2,$$

the bound on the number of mistakes given in Theorem 2.3. We can therefore view the 1-norm of $\boldsymbol{\alpha}$ as a measure of the complexity of the target concept in the dual representation.

Remark 2.11 The training data only enter the algorithm through the entries of the matrix $\mathbf{G} = \left(\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle\right)_{i,j=1}^{\ell}$, known as the Gram matrix, whose properties are briefly discussed in Appendix B and which will be related to other similar matrices in later chapters. This observation will have important consequences in Chapter 3.

2.1.2 Other Linear Classifiers

The problem of learning a hyperplane that separates two (separable) sets of points is an ill-posed one, in the sense that in general several different solutions exist. For example the perceptron algorithm may give a different solution depending on the order in which the examples are processed. The danger with ill-posed problems is that not all solutions may be equally useful. One way to render it well-posed is to try to optimise a different cost function, which ensures that if a solution exists it is always unique. For example we can choose not simply to learn any rule that correctly separates the two classes, but to choose from among these rules the one that is a maximum distance from the data. This is the hyperplane that realises the maximal margin. It is also said to have *maximal stability*. An iterative algorithm similar to the perceptron algorithm exists that is guaranteed to converge to the maximal margin solution. We will briefly analyse this algorithm in Chapter 7.

The perceptron algorithm is guaranteed to converge only if the data are linearly separable. A procedure that does not suffer from this limitation is Fisher's discriminant, which is aimed at finding the hyperplane $(\mathbf{w}, b)$ on which the projection of the data is maximally separated. The cost function to be optimised is the following:

$$F = \frac{(m_1 - m_{-1})^2}{\sigma_1^2 + \sigma_{-1}^2}$$

where m_i and σ_i are respectively the mean and standard deviation of the function output values

$$\{\langle \mathbf{w} \cdot \mathbf{x}_j \rangle + b : y_j = i\}$$

for the two classes, $i = 1, -1$.

The hyperplane that optimises this criterion can be found by solving a system of linear equations with a symmetric matrix formed from the training data and right hand side the difference between the two class means.

2.1.3 Multi-class Discrimination

The problem of two-class discrimination we have studied so far can also be solved by defining a weight vector $\mathbf{w}_i$ and a bias b_i for each class. Each time a new instance has to be classified, both functions are evaluated, and the point $\mathbf{x}$ is assigned to class 1 if $\langle \mathbf{w}_1 \cdot \mathbf{x} \rangle + b_1 \geq \langle \mathbf{w}_{-1} \cdot \mathbf{x} \rangle + b_{-1}$, to class -1 otherwise. This approach is equivalent to discrimination using the single hyperplane $(\mathbf{w}, b)$, with the substitutions $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_{-1}$, $b = b_1 - b_{-1}$.

For a multi-class classification problem the output domain is $Y = \{1, 2, \dots, m\}$. The generalisation of linear learning machines to the m-class case is straightforward: to each of the m classes are associated a weight vector and a bias, $(\mathbf{w}_i, b_i)$, $i \in \{1, \dots, m\}$, and the decision function is given by

$$c(\mathbf{x}) = \arg \max_{1 \leq i \leq m} \left(\langle \mathbf{w}_i \cdot \mathbf{x} \rangle + b_i \right).$$

Geometrically this is equivalent to associating a hyperplane to each class, and to assigning a new point $\mathbf{x}$ to the class whose hyperplane is furthest from it. The input space is split into m simply connected and convex regions.

Algorithms for learning the m hyperplanes simultaneously from data exist, and are extensions of the basic procedures outlined above.