Chapter

4 LEARNING WITH SUPPORT VECTOR MACHINES (SVM)

4.1 INTRODUCTION

The classical regression and Bayesian classification statistical techniques (overview given in Section 3.5) are based on the very strict assumption that probability distribution models or probability density functions are known. Unfortunately, in real-world practical situations, there is not enough information about the underlaying distributions, and *distribution-free regression* or *classification* is needed that does not require knowledge of probability distributions. The only available information is the training dataset.

Under the assumption that the data follows a normal distribution, statistical techniques result in linear regression functions. For the classification problems with normally distributed classes and equal covariance matrices for corresponding classes, we get linear discriminant functions using statistical techniques.

The linear functions are extremely powerful for the regression/classification problems whenever the stated assumptions are true. Unfortunately, the classical statistical paradigm turns out to be inappropriate for many real-life problems because the underlying real-life data generation laws may typically be very far from normal distribution.

Till the 1980s, most of the data analysis and learning methods were confined to linear statistical techniques. Most of the optimal algorithms and theoretical results were available for inference of linear dependencies from data; for nonlinear ones, only greedy algorithms, local minima, and heuristic search were known.

A paradigm shift occurred in the 1980s when researchers, armed with powerful computers of the day, boldly embraced nonlinear methods of learning. The simultaneous introduction of decision trees (Chapter 8) and neural network algorithms (Chapter 5) revolutionalized the practice of pattern recognition and numeric prediction. These methods opened the possibility of efficient learning of nonlinear dependencies.

A second paradigm shift occurred in the 1990s with the introduction of the 'kernel methods'. The differences with the previous approaches are worth mentioning. Most of the earlier learning

algorithms had, to a large extent, been based on heuristics or on loose analogies with natural learning systems, e.g., model of nervous systems (neural networks). They were mostly the result of creativity and extensive tuning by the designer, and the underlying reasons for their performance were not fully understood. A large part of the work was devoted to designing heuristics to avoid local minima in hypothesis search process.

With the emergence of computational learning theory (Section 2.3), new efficient representations of nonlinear functions have been discovered and used for the design of learning algorithms. This has led to the creation of powerful algorithms, whose training often amounts to optimization. In other words, they are free from local minima. The use of optimization theory marks a radical departure from the previous greedy search algorithms. In a way, researchers now have the power of nonlinear function learning together with the conceptual and computational convenience, that was, hitherto, a characteristic of linear systems. Support Vector Machine (SVM), probably, represents the best known example of this class of algorithms.

In the present state-of-the-art, no machine learning method is inherently superior to any other; it is the type of problem, prior distribution, and other information that determine which method should provide the best performance. If one algorithm seems to outperform another in a particular situation, it is a consequence of its fit to the particular problem, not the general superiority of the algorithm. Machine learning involves searching through a space of possible hypotheses to determine one that fits the observed data and any prior knowledge held by the learner. We have covered in this book, hypotheses space which is being exploited by the practitioners in data mining problems.

In the current chapter, we present the basic concepts of SVM in an easily digestible way. Applied learning algorithms for support vector classification and regression have been thoroughly explained. The support vector machine is currently considered to be the best off-the-shelf learning algorithm and has been applied successfully in various domains.

Support vector machines were originally designed for binary classification. Initial research attempts were diverted towards making several two-class SVMs to do multiclass classification. Recently, several single-shot multiclass classification algorithms appeared in literature. Our focus in this chapter will be on binary classification; multiclass problems will be reduced to binary classification problems.

Support vector regression is the natural extension of methods used for classification. We will see in this chapter that SVM regression analysis retains all the properties of SVM classifiers. It has already become a powerful technique for predictive data analysis with many applications in varied areas of study.

In Section 3.5, we observed that the practical limitation of statistical approach is the assumed initial knowledge available on the process under investigation. For the Bayes procedure, one should know the underlying probability distributions. If only the forms of underlying distributions were known, we can use the training samples to estimate the values of their parameters.

In this chapter, we shall instead assume that we know the proper forms for the *discriminant functions*, and use the samples to estimate the values of parameters of the classifier. We shall examine various procedures for determining discriminant functions; none of them requiring knowledge of the forms of underlying distributions.

We shall be concerned with linear discriminant functions which have a variety of pleasant analytical properties. As we observed in Section 3.5, they can be optimal if the underlying

distributions are cooperative, such as Gaussians having equal covariance. Even when they are not optimal, we might be willing to sacrifice some performance in order to gain the advantage of their simplicity. Linear discriminant functions are relatively easy to compute and in the absence of information suggesting otherwise, linear classifiers are attractive candidates for initial, trial classifiers.

The problem of finding a linear discriminant function will be formulated as a problem of minimizing a criterion function. The obvious criterion function for classification purposes is the *misclassificastion error* (refer to Section 2.8). Instead of deriving discriminants based on misclassification error, we investigate related criterion function that is analytically more tractable.

4.2 LINEAR DISCRIMINANT FUNCTIONS FOR BINARY CLASSIFICATION

Let us assume two classes of patterns described by two-dimensional feature vectors (coordinates x_1 and x_2) as shown in Fig. 4.1. Each pattern is represented by vector $\mathbf{x} = [x_1\ x_2]^T \in \Re^2$. In Fig. 4.1, we have used circle to denote Class 1 patterns and square to denote Class 2 patterns. In general, patterns of each class will be characterized by random distributions of the corresponding feature vectors.

Figure 4.1 also shows a straight line separating the two classes. We can easily write the equation of the straight line in terms of the coordinates (features) x_1 and x_2 using coefficients or *weights* w_1 and w_2 and a *bias (offset)* or *threshold* term w_0, as given in Eqn (4.1). The weights determine the slope of the straight line, and the bias determines the deviation from the origin of the straight line intersections with the axes:

$$g(\mathbf{x}) = w_1\, x_1 + w_2\, x_2 + w_0 = 0 \tag{4.1}$$

We say that $g(\mathbf{x})$ is a *linear discriminant function* that divides (categorizes) $\Re^2$ into two *decision regions*.

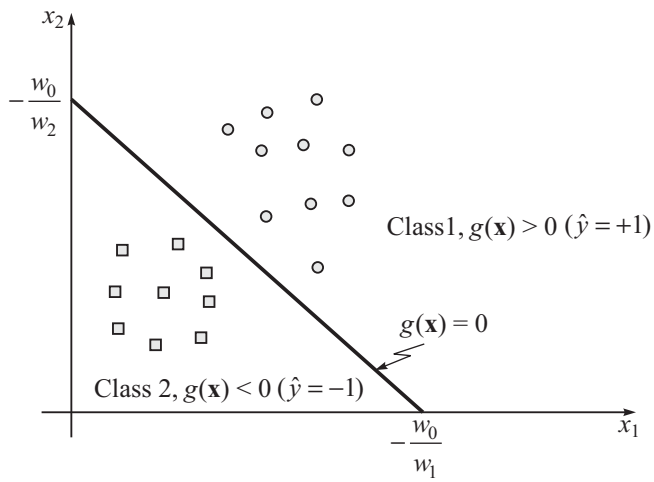


Figure 4.1 Linear discriminant function in two-dimensional space

The generalization of the linear discriminant function for an n-dimensional feature space in $\Re^n$ is straight forward:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0 \tag{4.2}$$

$\mathbf{x} = [x_1\ x_2\ \ldots\ x_n]^T$ is the feature vector

$\mathbf{w} = [w_1\ w_2\ \ldots\ w_n]^T$ is a *weight vector*

w_0 = *bias* parameter

The discriminant function is now a linear n-dimensional surface, called a *hyperplane*; symbolized as $\mathcal{H}$ in the discussion that follows.

For the discriminant function of the form of Eqn (4.2), a two-category classifier implements the following decision rule:

$$\text{Decide Class 1 if } g(\mathbf{x}) > 0 \text{ and Class 2 if } g(\mathbf{x}) < 0. \tag{4.3}$$

Thus, $\mathbf{x}$ is assigned to Class 1 if the inner product $\mathbf{w}^T\mathbf{x}$ exceeds the threshold (bias) $-w_0$, and to Class 2 otherwise. If $g(\mathbf{x}) = 0$, $\mathbf{x}$ can ordinarily be assigned to any class, but in this chapter, we shall leave the assignment undefined.

Figure 4.2 shows the architecture of a typical implementation of the linear classifier. It consists of two computational units: an aggregation unit and an output unit. The aggregation unit collects the n weighted input signals $w_1x_1, w_2\,x_2, \ldots, w_n\,x_n$ and sums them together. Note that the summation also has a bias term—a constant input $x_0 = 1$ with a weight of w_0. This sum is then passed on to the output unit, which is a step filter that returns –1 if its input is negative and +1 if its input is positive. In other words, the step filter implements the sign function:

$$\hat{y} = sgn\left(\sum_{j=1}^{n} w_j\,x_j + w_0\right) \tag{4.4a}$$

$$= sgn\,(\mathbf{w}^T\mathbf{x} + w_0) = sgn(g(\mathbf{x})) \tag{4.4b}$$

The *sgn* function extracts the appropriate pattern label from the decision surface $g(\mathbf{x})$. This means that linear classifier implemented in Fig. 4.2 represents a *decision function* of the form,

$$\hat{y} = sgn\,(g(\mathbf{x})) = sgn\,(\mathbf{w}^T\mathbf{x} + w_0) \tag{4.5}$$

The values ±1 in output unit are not unique; only change of sign is important. ±1 is taken for computational convenience.

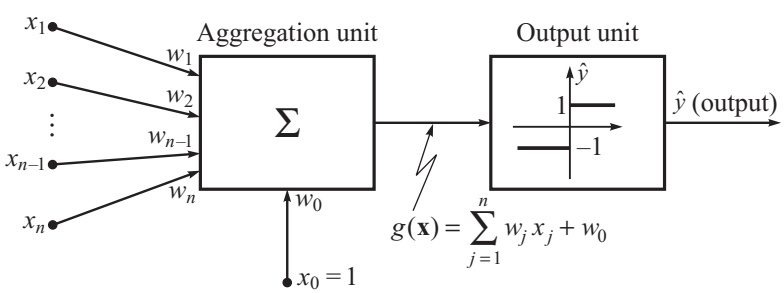


Figure 4.2 A simple linear classifier

If $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ are two points on the decision hyperplane, then the following is valid:

$$\mathbf{w}^T\mathbf{x}^{(1)} + w_0 = \mathbf{w}^T\mathbf{x}^{(2)} + w_0 = 0$$

This implies that

$$\mathbf{w}^T(\mathbf{x}^{(1)} - \mathbf{x}^{(2)}) = 0$$

The difference $(\mathbf{x}^{(1)} - \mathbf{x}^{(2)})$ obviously lies on the decision hyperplane for any $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$. The scalar product is equal to zero, meaning that the weights vector $\mathbf{w}$ is normal (perpendicular) to the decision hyperplane. Without changing the normal vector $\mathbf{w}$, varying w_0 moves the hyperplane parallel to itself. Note also that $\mathbf{w}^T\mathbf{x} + w_0 = 0$ has an inherent degree of freedom. We can rescale the hyperplane to $K\mathbf{w}^T\mathbf{x} + Kw_0 = 0$ for $K \in \Re^+$ (positive real numbers) without changing the hyperplane. Geometry for $n = 2$ with $w_1 > 0$, $w_2 > 0$ and $w_0 < 0$ is shown in Fig. 4.3.

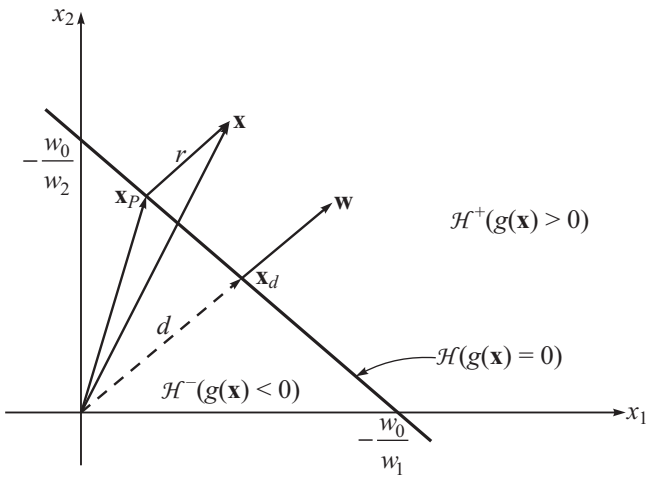


Figure 4.3 Linear decision boundary between two classes

The location of any point $\mathbf{x}$ may be considered relative to the hyperplane $\mathcal{H}$. Defining $\mathbf{x}_P$ as the normal projection of $\mathbf{x}$ onto $\mathcal{H}$ (shown in Fig. 4.3), we may decompose $\mathbf{x}$ as,

$$\mathbf{x} = \mathbf{x}_P + r\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{4.6}$$

where $\|\mathbf{w}\|$ is the Euclidean norm of $\mathbf{w}$, and $\mathbf{w}/\|\mathbf{w}\|$ is a unit vector (unit length with direction that of $\mathbf{w}$). Since by definition

$$g(\mathbf{x}_P) = \mathbf{w}^T\mathbf{x}_P + w_0 = 0$$

it follows that

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = \mathbf{w}^T\left(\mathbf{x}_P + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + w_0$$

$$= \mathbf{w}^T \mathbf{x}_P + w_0 + \frac{\mathbf{w}^T r\, \mathbf{w}}{\|\mathbf{w}\|}$$

$$= r\,\frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} = r\,\frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} = r\|\mathbf{w}\|$$

or
$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|} \tag{4.7}$$

In other words, $|g(\mathbf{x})|$ is a measure of the Euclidean distance of the point $\mathbf{x}$ from the decision hyperplane $\mathcal{H}$. If $g(\mathbf{x}) > 0$, we say that the point $\mathbf{x}$ is on the *positive side* of the hyperplane, and if $g(\mathbf{x}) < 0$, we say that point $\mathbf{x}$ is on the *negative side* of the hyperplane. When $g(\mathbf{x}) = 0$, the point $\mathbf{x}$ is on the hyperplane $\mathcal{H}$.

In general, the hyperplane $\mathcal{H}$ divides the feature space into two half-spaces: decision region $\mathcal{H}^+$ (positive side of hyperplane $\mathcal{H}$) for Class 1 ($g(\mathbf{x}) > 0$) and region $\mathcal{H}^-$ (negative side of hyperplane $\mathcal{H}$) for Class 2 ($g(\mathbf{x}) < 0$). The assignment of vector $\mathbf{x}$ to $\mathcal{H}^+$ or $\mathcal{H}^-$ can be implemented as,

$$\mathbf{w}^T \mathbf{x} + w_0 \begin{cases} > 0 \text{ if } \mathbf{x} \in \mathcal{H}^+ \\ = 0 \text{ if } \mathbf{x} \in \mathcal{H} \\ < 0 \text{ if } \mathbf{x} \in \mathcal{H}^- \end{cases} \tag{4.8}$$

The perpendicular distance d from the coordinates origin to the hyperplane $\mathcal{H}$ is given by $w_0/\|\mathbf{w}\|$, as is seen below.

$$g(\mathbf{x}_d) = \mathbf{w}^T \mathbf{x}_d + w_0 = 0;\ \mathbf{x}_d = d\,\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Therefore,

$$d\,\frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} + w_0 = 0;\ d\,\frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} = -w_0;\ d = \frac{-w_0}{\|\mathbf{w}\|} \tag{4.9}$$

The origin is on the negative side of $\mathcal{H}$ if $w_0 < 0$, and if $w_0 > 0$, the origin is on the positive side of $\mathcal{H}$. If $w_0 = 0$, the hyperplane passes through the origin.

Geometry for $n = 3$ is shown in Fig. 4.4.

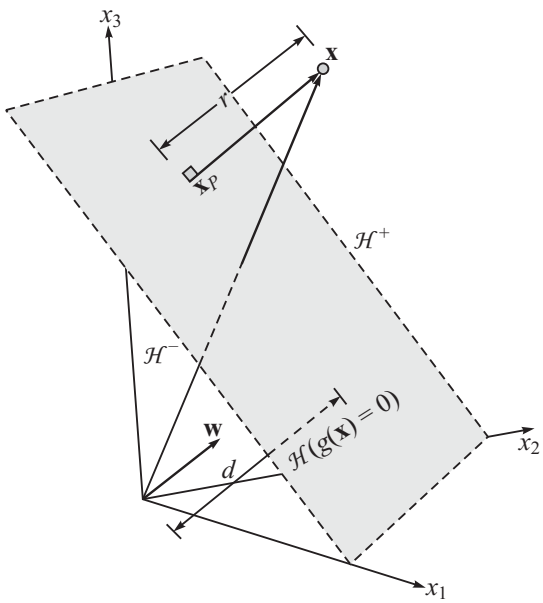


Figure 4.4 Hyperplane $\mathcal{H}$ separates the feature space into two half-space $\mathcal{H}^+(g(\mathrm{x}) > 0)$ and $\mathcal{H}^-(g(\mathrm{x}) < 0)$

4.3 PERCEPTRON ALGORITHM

Let us assume that we have a set of N samples $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$; some labeled Class 1 and some labeled Class 2:

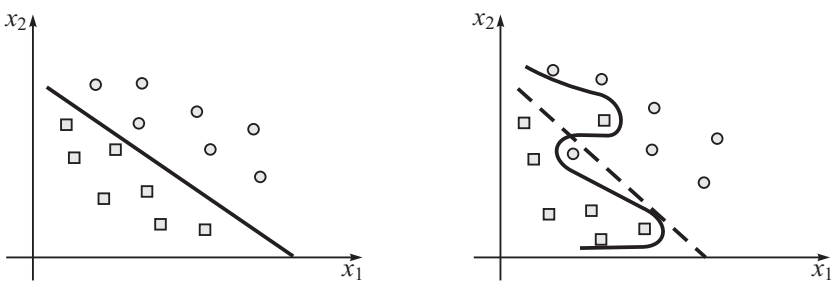
$$\mathcal{D}: \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\} \tag{4.10}$$

with $\mathbf{x}^{(i)} \in \Re^n$, and $y^{(i)} \in \{+1, -1\}$.

We want to use these samples to determine the weights $\mathbf{w}$ and w_0 in a linear discriminant function

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{4.11}$$

Let us say there is a reason behind the belief that there is a solution for which the likelihood of error is quite low. This leads to the desire to seek a weight vector that correctly classifies all the samples. If there does exist such a weight vector, the samples are said to be *linearly separable* (Fig. 4.5a).



(a) Linearly seperable (b) Linearly inseperable

Figure 4.5

Another strong alternative is available in *Neural Networks* (NN). History has proved that limitations of Rosenblatt's perceptron can be overcome by Neural Networks (discussed in the next chapter). The perceptron criterion function considers misclassified samples, and the gradient procedures for minimization are not applicable. The neural networks primarily solve the regression problems considering all the samples and *minimum squared-error criterion*, and employ gradient procedures for minimization. The algorithms for separable—as well as inseparable—data classification are first developed in the context of regression problems and then adapted for classification problems.

In the present chapter, our interest is in SVM-based solutions to real-life (nonlinear) classification and regression problems. To explain how a support vector machine works for these problems, it is perhaps easiest to start with the case of linearly separable patterns in the context of binary pattern classification. In this context, the main idea of a support vector machine is to construct a hyperplane as the decision surface in such a way that the margin of separation between Class 1 and Class 2 examples is maximized. We will then take up the more difficult case of linearly nonseparable patterns. With the material on how to find the optimal hypersurface for linearly nonseparable patterns at hand, we will formally describe the construction of a support vector machine for real-life (nonlinear) pattern recognition task. As we shall see shortly, basically the idea of a support vector machine hinges on the following two mathematical operations:

(i) Nonlinear mapping of input patterns into a high-dimensional feature space.
(ii) Construction of optimal hyperplane for linearly separating the feature vectors discovered in Step (i).

The final stage of our presentation will be to extend these results for application to multiclass classification problems, and nonlinear regression problems.

4.4 LINEAR MAXIMAL MARGIN CLASSIFIER FOR LINEARLY SEPARABLE DATA

Let the set of training (data) examples $\mathcal{D}$ be

$$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\} \tag{4.16}$$

where $\mathbf{x} = [x_1\ x_2\ \ldots\ x_n]^T$ is an n-dimensional *input vector* (pattern with n-features) for the ith example in a real-valued space $\mathbf{X} \subseteq \Re^n$; y is its *class label* (output value), and $y \in \{+1, -1\}$. +1 denotes Class 1 and –1 denotes Class 2.

To build a classifier, SVM finds a linear function of the form

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{4.17}$$

so that the input vector $\mathbf{x}^{(i)}$ is assigned to Class 1 if $g(\mathbf{x}^{(i)}) > 0$, and to Class 2 if $g(\mathbf{x}^{(i)}) < 0$, i.e.,

$$y^{(i)} = \begin{cases} +1 \text{ if } \mathbf{w}^T\mathbf{x}^{(i)} + w_0 > 0 \\ -1 \text{ if } \mathbf{w}^T\mathbf{x}^{(i)} + w_0 < 0 \end{cases} \tag{4.18}$$

Hence, $g(\mathbf{x})$ is a real-valued function; g: $\mathbf{X} \subseteq \Re^n \rightarrow \Re$.

$\mathbf{w} = [w_1\ w_2\ \ldots\ w_n]^T \in \Re^n$ is called the *weight vector* and $w_0 \in \Re$ is called the *bias*.

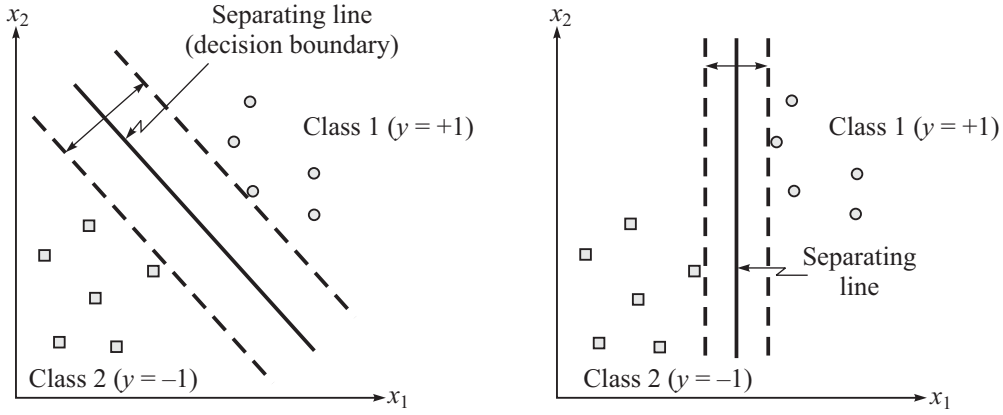
In essence, SVM finds a hyperplane

$$\mathbf{w}^T\mathbf{x} + w_0 = 0 \tag{4.19}$$

that separates Class 1 and Class 2 training examples. This hyperplane is called the *decision boundary* or *decision surface*. Geometrically, the hyperplane (4.19) divides the input space into two half spaces: one half for Class 1 examples and the other half for Class 2 examples. Note that hyperplane (4.19) is a line in a two-dimensional space and a plane in a three-dimensional space.

For linearly separable data, there are many hyperplanes (lines in two-dimensional feature space; Fig. 4.9) that can perform separation. How can one find the best one? The SVM framework provides good answer to this question. Among all the hyperplanes that minimize the training error, find the one with the largest *margin*—the gap between the data points of the two classes. This is an intuitively acceptable approach: select the decision boundary that is far away from both the classes (Fig. 4.10). Large-margin separation is expected to yield good classification on previously unseen data, i.e., good generalization.

From Section 4.2, we know that in $\mathbf{w}^T\mathbf{x} + w_0 = 0$, $\mathbf{w}$ defines a direction perpendicular to the hyperplane. $\mathbf{w}$ is called the *normal vector* (or simply *normal*) of the hyperplane. Without changing the normal vector $\mathbf{w}$, varying w_0 moves the hyperplane parallel to itself. Note also that $\mathbf{w}^T\mathbf{x} + w_0 = 0$ has an inherent degree of freedom. We can rescale the hyperplane to $K\mathbf{w}^T\mathbf{x} + Kw_0 = 0$ for $K \in \Re^+$ (positive real numbers), without changing the hyperplane.



(a) Large margin separation (b) Small margin separation

Figure 4.10

Since SVM maximizes the margin between Class 1 and Class 2 data points, let us find the margin. The linear function $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ gives an algebraic measure of the distance r from $\mathbf{x}$ to the hyperplane $\mathbf{w}^T\mathbf{x} + w_0 = 0$. We have seen earlier in Section 4.2 that this distance is given by (Eqn (4.7))

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|} \tag{4.20}$$

Now consider a Class 1 data point $(\mathbf{x}^{(i)}, +1)$ that is closest to the hyperplane $\mathbf{w}^T\mathbf{x} + w_0 = 0$ (Fig. 4.11).

The distance d_1 of this data point from the hyperplane is

$$d_1 = \frac{g(\mathbf{x}^{(i)})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T\mathbf{x}^{(i)} + w_0}{\|\mathbf{w}\|} \tag{4.21a}$$

Similarly,

$$d_2 = \frac{g(\mathbf{x}^{(k)})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T\mathbf{x}^{(k)} + w_0}{\|\mathbf{w}\|} \tag{4.21b}$$

where $(\mathbf{x}^{(k)}, -1)$ is a Class 2 data point closest to the hyperplane $\mathbf{w}^T\mathbf{x} + w_0 = 0$.

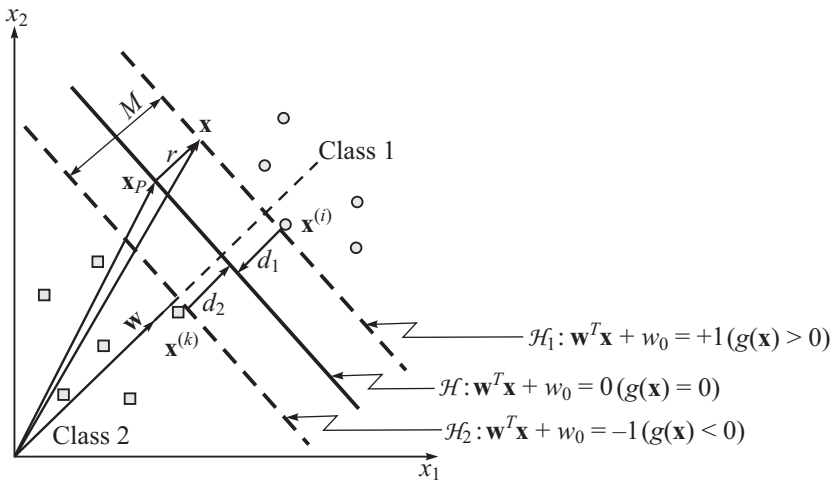


Figure 4.11 Geometric interpretation of algebraic distances of points to a hyperplane for two-dimenstional case

We define two parallel hyperplanes $\mathcal{H}_1$ and $\mathcal{H}_2$ that pass through $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(k)}$, respectively. $\mathcal{H}_1$ and $\mathcal{H}_2$ are also parallel to the hyperplane $\mathbf{w}^T\mathbf{x} + w_0 = 0$. We can rescale $\mathbf{w}$ and w_0 to obtain (this rescaling, as we shall see later, simplifies the quest for significant patterns, called *support vectors*)

$$\begin{aligned} \mathcal{H}_1 &: \mathbf{w}^T\mathbf{x} + w_0 = +1 \\ \mathcal{H}_2 &: \mathbf{w}^T\mathbf{x} + w_0 = -1 \end{aligned} \tag{4.22}$$

such that

$$\begin{aligned} \mathbf{w}^T\mathbf{x}^{(i)} + w_0 \geq 1 \;\text{ if }\; y^{(i)} = +1 \\ \mathbf{w}^T\mathbf{x}^{(i)} + w_0 \leq -1 \;\text{ if }\; y^{(i)} = -1 \end{aligned} \tag{4.23a}$$

or equivalently

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1 \tag{4.23b}$$

which indicates that no training data fall between hyperplanes $\mathcal{H}_1$ and $\mathcal{H}_2$. The distance between the two hyperplanes is the margin M. In the light of rescaling given by (4.22),

$$d_1 = \frac{1}{\|\mathbf{w}\|}; d_2 = \frac{-1}{\|\mathbf{w}\|} \tag{4.24}$$

where the '–' sign indicates that $\mathbf{x}^{(k)}$ lies on the side of the hyperplane $\mathbf{w}^T\mathbf{x} + w_0 = 0$ opposite to that where $\mathbf{x}^{(i)}$ lies. From Fig. 4.11, it follows that

$$M = \frac{2}{\|\mathbf{w}\|} \tag{4.25}$$

Equation (4.25) states that maximizing the margin of separation between classes is equivalent to minimizing the Euclidean norm of the weight vector $\mathbf{w}$.

Since SVM looks for the separating hyperplane that minimizes the Euclidean norm of the weight vector, this gives us an optimization problem. A full description of the solution method requires a significant amount of optimization theory, which is beyond the scope of this book. We will only use relevant results from optimization theory, without giving formal definitions, theorems or proofs (refer to [54, 55] for details).

Our interest here is in the following nonlinear optimization problem with inequality constraints:

$$\begin{aligned} &\textit{minimize } f(\mathbf{x}) \\ &\text{subject to } g_i(\mathbf{x}) \geq 0;\ i = 1, \ldots, m \end{aligned} \tag{4.26}$$

where $\mathbf{x} = [x_1\ x_2 \cdots x_n]^T$, and the functions f and g_i are continuously differentiable.

The optimality conditions are expressed in terms of the *Lagrangian function*

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) \tag{4.27}$$

where $\boldsymbol{\lambda} = [\lambda_1 \cdots \lambda_m]^T$ is a vector of Lagrange multipliers.

An optimal solution to the problem (4.26) must satisfy the following necessary conditions, called *Karush-Kuhn-Tucker* (KKT) *conditions:*

$$\begin{aligned} &\text{(i)} \quad \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_j} = 0;\ j = 1, \ldots, n \\ &\text{(ii)} \quad g_i(\mathbf{x}) \geq 0;\ i = 1, \ldots, m \\ &\text{(iii)} \quad \lambda_i \geq 0;\ i = 1, \ldots, m \\ &\text{(iv)} \quad \lambda_i\, g_i(\mathbf{x}) = 0;\ i = 1, \ldots, m \end{aligned} \tag{4.28}$$

In view of condition (iii), the vector of Lagrange multipliers belongs to the set $\{\boldsymbol{\lambda} \in \Re^m, \boldsymbol{\lambda} \geq \mathbf{0}\}$. Also note that condition (ii) is the original set of constraints.

Our interest, as we will see shortly, is in convex functions f and linear functions g_i. For this class of optimization problems, when there exist vectors $\mathbf{x}^0$ and $\boldsymbol{\lambda}^0$ such that the point $(\mathbf{x}^0, \boldsymbol{\lambda}^0)$ satisfies the

KKT conditions (4.28), then $\mathbf{x}^0$ gives the global minimum of the function $f(\mathbf{x})$, with the constraint given in (4.26).

Let

$$L^*(\mathbf{x}) = \max_{\boldsymbol{\lambda} \in \Re^m} L(\mathbf{x}, \boldsymbol{\lambda}), \text{ and } L_*(\boldsymbol{\lambda}) = \min_{\mathbf{x} \in \Re^n} L(\mathbf{x}, \boldsymbol{\lambda})$$

It is clear from these equations that for any $\mathbf{x} \in \Re^n$ and $\boldsymbol{\lambda} \in \Re^m$,

$$L_*(\boldsymbol{\lambda}) \leq L(\mathbf{x}, \boldsymbol{\lambda}) \leq L^*(\mathbf{x})$$

and thus, in particular

$$L_*(\boldsymbol{\lambda}) \leq L^*(\mathbf{x})$$

This holds for any $\mathbf{x} \in \Re^n$ and $\boldsymbol{\lambda} \in \Re^m$; so it holds for the $\boldsymbol{\lambda}$ that maximizes the left-hand side, and the $\mathbf{x}$ that minimizes the right-hand side. Thus,

$$\max_{\boldsymbol{\lambda} \in \Re^m} \min_{\mathbf{x} \in \Re^n} L(\mathbf{x}, \boldsymbol{\lambda}) \leq \min_{\mathbf{x} \in \Re^n} \max_{\boldsymbol{\lambda} \in \Re^m} L(\mathbf{x}, \boldsymbol{\lambda})$$

The two problems, *min-max* and *max-min*, are said to be *dual* to each other. We refer to the min-max problem as the *primal problem*. The objective to be minimized, $L^*(\mathbf{x})$, is referred to as the *primal function*. The max-min problem is referred to as the *dual problem*, and $L_*(\boldsymbol{\lambda})$ as the *dual function*. The optimal primal and dual function values are equal when f is a convex function and g_i are linear functions. The concept of duality is widely used in the optimization literature. The aim is to provide an alternative formulation of the problem which is more convenient to solve computationally and/or has some theoretical significance. In the context of SVM, the dual problem is not only easy to solve computationally, but also crucial for using *kernel functions* to deal with nonlinear decision boundaries. This will be clear later.

The nonlinear optimization problem defined in (4.26) can be represented as min-max problem, as follows:

For the Lagrangian (4.27), we have

$$L^*(\mathbf{x}) = \max_{\boldsymbol{\lambda} \in \Re^m} \left[f(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) \right]$$

Since $g_i(\mathbf{x}) \geq 0$ for all i, $\lambda_i = 0$ ($i = 1, \ldots, m$) would maximize the Lagrangian; thus,

$$L^*(\mathbf{x}) = f(\mathbf{x})$$

Therefore, our original constrained problem (4.26) becomes the min-max primal problem:

$$\underset{\mathbf{x} \in \Re^n}{\text{minimize}}\ L^*(\mathbf{x})$$

$$\text{subject to } g_i(\mathbf{x}) \geq 0;\ i = 1, \ldots, m$$

The concept of duality gives the following formulation for max-min dual problem:

$$\underset{\boldsymbol{\lambda} \in \Re^m, \boldsymbol{\lambda} \geq 0}{\text{maximize}}\ L_*(\boldsymbol{\lambda})$$

More explicitly, this nonlinear optimization problem with *dual variables* $\boldsymbol{\lambda}$, can be written in the form:

$$\underset{\boldsymbol{\lambda} \geq \mathbf{0}}{maximize} \; \underset{\mathbf{x} \in \Re^n}{min} \left[f(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) \right] \tag{4.29}$$

Let us now state the learning problem in SVM.

Given a set of linearly separable training examples,

$$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\},$$

the learning problem is to solve the following constrained minimization problem:

$$\begin{aligned} &minimize && f(\mathbf{w}) = \tfrac{1}{2} \mathbf{w}^T \mathbf{w} \\ &\text{subject to} && y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1; \; i = 1, \ldots, N \end{aligned} \tag{4.30}$$

This formulation is called the primal formulation of *hard-margin* SVM. Solving this problem will produce the solutions for $\mathbf{w}$ and w_0 which in turn, give us the maximal margin hyperplane $\mathbf{w}^T\mathbf{x} + w_0 = 0$ with the margin $2/\|\mathbf{w}\|$.

The objective function is quadratic and convex in parameters $\mathbf{w}$, and the constraints are linear in parameters $\mathbf{w}$ and w_0. The dual formulation of this constrained optimization problem is obtained as follows.

First we construct the Lagrangian:

$$L(\mathbf{w}, w_0, \boldsymbol{\lambda}) = \tfrac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^{N} \lambda_i [y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1] \tag{4.31}$$

The KKT conditions are as follows:

$$\begin{aligned} &\text{(i)} \quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{0}; \text{ which gives } \mathbf{w} = \sum_{i=1}^{N} \lambda_i \, y^{(i)} \mathbf{x}^{(i)} \\ &\qquad \frac{\partial L}{\partial w_0} = 0; \text{ which gives } \sum_{i=1}^{N} \lambda_i \, y^{(i)} = 0 \\ &\text{(ii)} \quad y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 \geq 0; \; i = 1, \ldots, N \\ &\text{(iii)} \quad \lambda_i \geq 0; \; i = 1, \ldots, N \\ &\text{(iv)} \quad \lambda_i [y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1] = 0; \; i = 1, \ldots, N \end{aligned} \tag{4.32}$$

From condition (i) of KKT conditions (4.32), we observe that the solution vector has an expansion in terms of training examples. Note that although the solution $\mathbf{w}$ is unique (due to the strict convexity of the function $f(\mathbf{w})$), the dual variables λ_i need not be. There is a dual variable λ_i for each training data point. Condition (iv) of KKT conditions (4.32) shows that for data points not on the margin hyperplanes (i.e., $\mathcal{H}_1$ and $\mathcal{H}_2$), $\lambda_i = 0$:

$$y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 > 0 \Rightarrow \lambda_i = 0$$

For data points on the margin hyperplanes, $\lambda_i \geq 0$:

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) - 1 = 0 \Rightarrow \lambda_i \geq 0$$

However, the data points on the margin hyperplanes with $\lambda_i = 0$ do not contribute to the solution $\mathbf{w}$, as is seen from condition (i) of KKT conditions (4.32). The data points on the margin hyperplanes with associated dual variables $\lambda_i > 0$ are called *support vectors*, which give the name to the algorithm, *support vector machines*.

To postulate the dual problem, we first expand Eqn (4.31), term by term, as follows:

$$L(\mathbf{w}, w_0, \boldsymbol{\lambda}) = \tfrac{1}{2}\,\mathbf{w}^T\,\mathbf{w} - \sum_{i=1}^{N}\lambda_i y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)} - w_0\sum_{i=1}^{N}\lambda_i y^{(i)} + \sum_{i=1}^{N}\lambda_i \tag{4.33}$$

Transformation from the primal to the corresponding dual is carried out by setting the partial derivatives of the Lagrangian (4.33) with respect to the *primal variables* (i.e., $\mathbf{w}$ and w_0) to zero, and substituting the resulting relations back into the Lagrangian. The objective is to merely substitute condition (i) of KKT conditions (4.32) into the Lagrangian (4.33) to remove the primal variables; which gives us the dual objective function.

The third term on the right-hand side of Eqn (4.33) is zero by virtue of condition (i) of KKT conditions (4.32). Furthermore, from this condition we have,

$$\mathbf{w}^T\,\mathbf{w} = \sum_{i=1}^{N}\lambda_i y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)} = \sum_{i=1}^{N}\sum_{k=1}^{N}\lambda_i\lambda_k y^{(i)}y^{(k)}\mathbf{x}^{(i)T}\,\mathbf{x}^{(k)}$$

Accordingly, minimization of function L in Eqn (4.33) with respect to primal variables $\mathbf{w}$ and w_0, gives us the following dual objective function:

$$L_*(\boldsymbol{\lambda}) = \sum_{i=1}^{N}\lambda_i - \tfrac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}\lambda_i\lambda_k y^{(i)}y^{(k)}\mathbf{x}^{(i)T}\,\mathbf{x}^{(k)} \tag{4.34}$$

We may now state the dual optimization problem.

Given a set of linearly separable training examples $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, find the dual variables $\{\lambda_i\}_{i=1}^{N}$, that maximize the objective function (4.34) subject to the constraints

- $\displaystyle\sum_{i=1}^{N}\lambda_i y^{(i)} = 0$ (4.35)
- $\lambda_i \geq 0$; $i = 1, \ldots, N$

This formulation is dual formulation of the *hard-margin SVM*.

Having solved the dual problem numerically (using MATLAB's **quadprog** function, for example), the resulting optimum λ_i values are then used to compute $\mathbf{w}$ and w_0. $\mathbf{w}$ is computed using condition (i) of KKT conditions (4.32):

$$\mathbf{w} = \sum_{i=1}^{N}\lambda_i y^{(i)}\mathbf{x}^{(i)} \tag{4.36}$$

and w_0 is computed using condition (iv) of KKT conditions (4.32):

$$\lambda_i[y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) - 1] = 0;\ i = 1, \ldots, N \tag{4.37}$$

Note that though there are N values of λ_i in Eqn (4.36), most vanish with $\lambda_i = 0$ and only a small percentage have $\lambda_i > 0$. The set of $\mathbf{x}^{(i)}$ whose $\lambda_i > 0$ are the *support vectors*, and as we see in Eqn (4.36), $\mathbf{w}$ is the weighted sum of these training instances that are selected as the support vectors:

$$\mathbf{w} = \sum_{i \in svindex} \lambda_i y^{(i)} \mathbf{x}^{(i)} \tag{4.38}$$

where *svindex* denotes the set of indices of support vectors.

From Eqn (4.38), we see that the support vectors $\mathbf{x}^{(i)}$; $i \in$ *svindex*, satisfy

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) = 1$$

and lie on the margin. We can use this fact to calculate w_0 from any support vector as,

$$w_0 = \frac{1}{y^{(i)}} - \mathbf{w}^T\mathbf{x}^{(i)}$$

For $y^{(i)} \in [+1, -1]$, we can equivalently express this equation as,

$$w_0 = y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} \tag{4.39}$$

Instead of depending on one support vector to compute w_0, in practice, all support vectors are used to compute w_0, and then their average is taken for the final value of w_0. This is because the values of λ_i are computed numerically and can have numerical errors.

$$w_0 = \frac{1}{|svindex|} \sum_{i \in svindex} (y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}) \tag{4.40}$$

where $|svindex|$ corresponds to total number of indices in the set *svindex*, i.e., total number of support vectors.

The majority of λ_i are 0, for which $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) > 1$. These are the $\mathbf{x}^{(i)}$ points that exist more than adequately away from the discriminant, and have zero effect on the hyperplane. The instances that are not support vectors have no information; the same solution will be obtained on removing any subset from them. From this viewpoint, the SVM algorithm can be said to be similar to the k-NN algorithm (Section 3.4) which stores only the instances neighboring the class discriminant.

During testing, we do not enforce a margin. We calculate

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{4.41a}$$

and choose the class according to the sign of $g(\mathbf{x})$: $sgn\ (g(\mathbf{x}))$ which we call the *indicator function* i_F,

$$i_F = \hat{y} = sgn\ (\mathbf{w}^T\mathbf{x} + w_0) \tag{4.41b}$$

Choose Class 1 ($\hat{y} = +1$) if $\mathbf{w}^T\mathbf{x} + w_0 > 0$, and Class 2 ($\hat{y} = -1$) otherwise.

Example 4.1

In this example, we visualize SVM (hard-margin) formulation in two variables. Consider the toy dataset given in Table 4.1.

SVM finds a hyperplane

$$\mathcal{H}: w_1 x_1 + w_2 x_2 + w_0 = 0$$

and two bounding planes

$$\mathcal{H}_1: w_1 x_1 + w_2 x_2 + w_0 = +1$$
$$\mathcal{H}_2: w_1 x_1 + w_2 x_2 + w_0 = -1$$

such that

$$w_1 x_1 + w_2 x_2 + w_0 \geq +1 \quad \text{if } y^{(i)} = +1$$
$$w_1 x_1 + w_2 x_2 + w_0 \leq -1 \quad \text{if } y^{(i)} = -1$$

or equivalently

$$y^{(i)}(w_1 x_1 + w_2 x_2 + w_0) \geq 1$$

We write these constraints explicitly as (refer to Table 4.1),

$(-1)\ [w_1 + w_2 + w_0] \geq 1$

$(-1)\ [2w_1 + w_2 + w_0] \geq 1$

$(-1)\ [w_1 + 2w_2 + w_0] \geq 1$

$(-1)\ [2w_1 + 2w_2 + w_0] \geq 1$

$(+1)\ [4w_1 + 4w_2 + w_0] \geq 1$

$(+1)\ [4w_1 + 5w_2 + w_0] \geq 1$

$(+1)\ [5w_1 + 4w_2 + w_0] \geq 1$

$(+1)\ [5w_1 + 5w_2 + w_0] \geq 1$

Table 4.1 Data for classification

Sample i	$x_1^{(i)}$	$x_2^{(i)}$	$y^{(i)}$
1	1	1	–1
2	2	1	–1
3	1	2	–1
4	2	2	–1
5	4	4	+1
6	4	5	+1
7	5	4	+1
8	5	5	+1

The constraint equations in matrix form:

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \end{bmatrix}}_{\mathbf{Y}} \left[\underbrace{\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \\ 2 & 2 \\ 1.5 & 1.5 \\ 4 & 4 \\ 4 & 5 \\ 5 & 4 \\ 5 & 5 \\ 4.5 & 4.5 \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{\mathbf{w}} + w_0 \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}}_{\mathbf{e}} \right] \geq \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}}_{\mathbf{e}}$$

or

$$\mathbf{Y}(\mathbf{X}\mathbf{w} + w_0\mathbf{e}) \geq \mathbf{e} \tag{4.42a}$$

For a dataset with N samples, and n features in vector $\mathbf{x}$,

$$\underset{(N\times N)}{\mathbf{Y}} = \begin{bmatrix} y^{(1)} & 0 & \dots & 0 \\ 0 & y^{(2)} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & y^{(N)} \end{bmatrix}; \underset{(N\times n)}{\mathbf{X}} = \begin{bmatrix} \mathbf{x}^{(1)T} \\ \mathbf{x}^{(2)T} \\ \vdots \\ \mathbf{x}^{(N)T} \end{bmatrix}; \underset{n\times 1}{\mathbf{w}} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}; \underset{(N\times 1)}{\mathbf{e}} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{4.42b}$$

$$\mathbf{Y} = diag(\mathbf{y}); \mathbf{y} = [y^{(1)}\, y^{(2)} \dots y^{(N)}]^T \tag{4.42c}$$

Our aim is to find the weight matrix $\mathbf{w}$ and the bias term w_0 that maximize the margin of separation between the hyperplanes $\mathcal{H}_1$ and $\mathcal{H}_2$, and at the same time satisfy the constraint equations (4.42). It gives us an optimization problem

$$\underset{\mathbf{w},\, w_0}{maximize} \left(\frac{1}{2}\mathbf{w}^T\mathbf{w}\right)$$
$$\text{subject to } \mathbf{Y}(\mathbf{X}\mathbf{w} + w_0\mathbf{e}) \geq \mathbf{e} \tag{4.43}$$

Once we obtain $\mathbf{w}$ and w_0, we have our decision boundary:

$$\mathbf{w}^T\mathbf{x} + w_0 = 0$$

and for a new unseen data point $\mathbf{x}$, we assign $sgn\ (\mathbf{w}^T\mathbf{x} + w_0)$ as the class value.

For solving the above problem in *primal*, we need to rewrite the problem in the standard QP (Quadratic Programming) format.

Another way to solve the above primal problem is the Lagrangian dual. The problem is more easily solved in terms of its Lagrangian dual variables:

$$\underset{\boldsymbol{\lambda} \geq \mathbf{0}}{maximize}\left[\min_{\mathbf{w},\, w_0} L(\mathbf{w}, w_0, \boldsymbol{\lambda})\right]$$

where (Eqn (4.31)) (4.44)

$$L(\mathbf{w}, w_0, \boldsymbol{\lambda}) = \tfrac{1}{2}\mathbf{w}^T\mathbf{w} - \boldsymbol{\lambda}^T[\mathbf{Y}(\mathbf{X}\mathbf{w} + w_0\mathbf{e}) - \mathbf{e}]$$

$\underset{(N\times 1)}{\boldsymbol{\lambda}} = [\lambda_1\ \lambda_2\ \ldots\ \lambda_N]^T$ is a vector of Lagrange multipliers.

It leads to the *dual optimization problem* (Eqns (4.34–4.35))

$$\underset{\boldsymbol{\lambda}}{maximize}\left[\boldsymbol{\lambda}^T\mathbf{e} - \tfrac{1}{2}\boldsymbol{\lambda}^T\mathbf{Y}\mathbf{X}\mathbf{X}^T\mathbf{Y}\boldsymbol{\lambda}\right]$$

subject to $\mathbf{e}^T\mathbf{Y}\boldsymbol{\lambda} = 0;\ \boldsymbol{\lambda} \geq \mathbf{0}$ (4.45)

Some standard quadratic optimization programs typically *minimize* the given objective function:

$$\underset{\boldsymbol{\lambda}}{minimize}\left[\tfrac{1}{2}\boldsymbol{\lambda}^T\mathbf{Q}\boldsymbol{\lambda} - \mathbf{e}^T\boldsymbol{\lambda}\right]$$

subject to $\mathbf{y}^T\boldsymbol{\lambda} = 0;\ \boldsymbol{\lambda} \geq \mathbf{0}$ (4.46)

where $\mathbf{Q} = \mathbf{Y}\,\mathbf{X}\,\mathbf{X}^T\,\mathbf{Y}$

The input required for the above program is only $\mathbf{X}$ and $\mathbf{y}$. It returns the Lagrange multiplier vector $\boldsymbol{\lambda}$.

Having solved the dual problem numerically (using a standard optimization program), optimum λ_i values are then used to compute $\mathbf{w}$ and w_0 (Eqns (4.38, 4.40)):

$$\mathbf{w} = \sum_{i \in svindex} \lambda_i\, y^{(i)}\mathbf{x}^{(i)} \tag{4.47a}$$

$$w_0 = \frac{1}{|svindex|}\left[\sum_{i \in svindex}[y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}]\right] \tag{4.47b}$$

Using the MATLAB **quadprog** routine for the dataset of Table 4.1, we obtain [56]

$$\boldsymbol{\lambda}^T = [0\quad 0\quad 0\quad 0.25\quad 0\quad 0.25\quad 0\quad 0\quad 0\quad 0]$$

This means that data points with index 4 and 6 are *support vectors*; i.e., support vectors are:

$$\begin{bmatrix}2\\2\end{bmatrix};\ \begin{bmatrix}4\\4\end{bmatrix}$$

and *svindex* is {4, 6}; |*svindex*| = 2.

$$\mathbf{w} = \sum_{i \in svindex} \lambda_i \, y^{(i)} \mathbf{x}^{(i)} = 0.25\,[-1] \begin{bmatrix} 2 \\ 2 \end{bmatrix} + 0.25\,[1] \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$w_0 = \tfrac{1}{2}\,[y^{(4)} - \mathbf{w}^T\mathbf{x}^{(4)} + y^{(6)} - \mathbf{w}^T\mathbf{x}^{(6)}]$$

$$= \tfrac{1}{2}\left[-1 - [w_1 \; w_2] \begin{bmatrix} 2 \\ 2 \end{bmatrix} + 1 - [w_1 \; w_2] \begin{bmatrix} 4 \\ 4 \end{bmatrix}\right]$$

$$= -3$$

Therefore, the decision hyperplane $g(\mathbf{x})$ is

$$g(\mathbf{x}) = 0.5x_1 + 0.5x_2 - 3$$

and the *indicator function*

$$i_F = \hat{y} = sgn(g(\mathbf{x})) = sgn(0.5x_1 + 0.5x_2 - 3)$$

4.5 LINEAR SOFT MARGIN CLASSIFIER FOR OVERLAPPING CLASSES

The linear hard-margin classifier gives a simple SVM when the samples are linearly separable. In practice, however, the training data is almost always linearly nonseparable because of random errors owing to different reasons. For instance, certain instances may be wrongly labeled. The labels could be different even for two input vectors that are identical.

If SVM has to be of some use, it should permit noise in the training data. But, with noisy data, the linear SVM algorithm described in earlier section, will not obtain a solution as the constraints cannot be satisfied. For instance, in Fig. 4.12, there exists a Class 2 point (square) in the Class 1 region, and a Class 1 point (circle) in the Class 2 area. However, in spite of the couple of mistakes, the decision boundary seems to be good. But the *hard-margin* classifier presented previously cannot be used, because all the constraints

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1;\; i = 1, \ldots, N$$

cannot be satisfied.

So the constraints have to be modified to permit mistakes. To allow errors in data, we can relax the margin constraints by introducing *slack* variables, $\zeta_i (\geq 0)$, as follows:

$$\mathbf{w}^T \mathbf{x}^{(i)} + w_0 \geq 1 - \zeta_i \qquad \text{for} \qquad y^{(i)} = +1$$

$$\mathbf{w}^T \mathbf{x}^{(i)} + w_0 \leq -1 + \zeta_i \qquad \text{for} \qquad y^{(i)} = -1$$

Thus, we have the new constraints

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1 - \zeta_i;\; i = 1, \ldots, N$$

$$\zeta_i \geq 0 \tag{4.48}$$

The geometric interpretation is shown in Fig. 4.12.

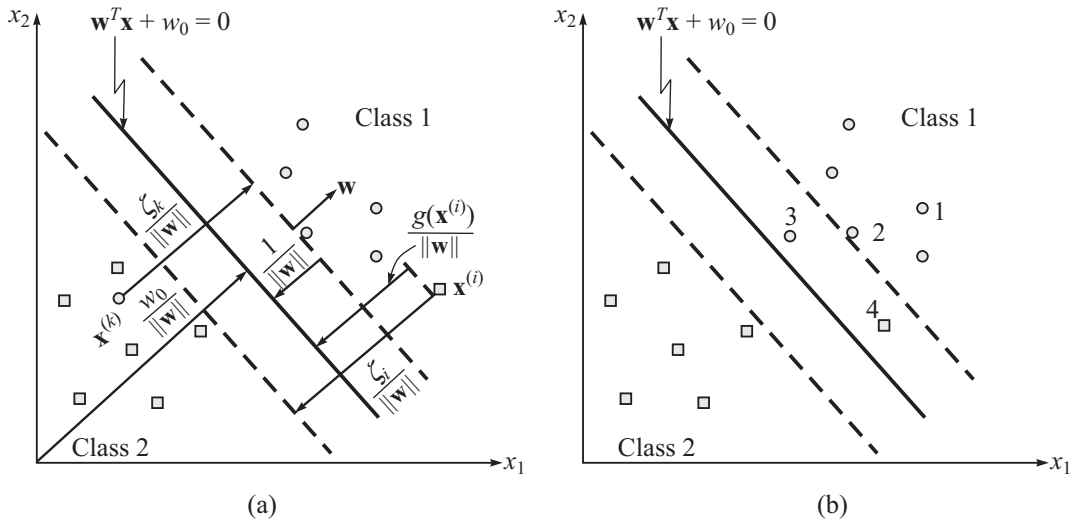


Figure 4.12 Soft decision boundary

In classifying an instance, there are four possible cases (see Fig. 4.12(b)). Instance 1 is on the correct side and far away from the origin; $y^{(i)}g(\mathbf{x}^{(i)}) > 1$, $\zeta_i = 0$. Instance 2 is on the correct side and on the margin; $\zeta_i = 0$. For instance 3, $\zeta_i = 1 - g(\mathbf{x}^{(i)})$, $0 < \zeta_i < 1$, the instance is on the correct side but in the margin and not sufficiently away. For instance 4, $\zeta_i = 1 + g(\mathbf{x}^{(i)}) > 1$, the instance is on the wrong side—this is a misclassification.

We also need to penalize the errors in the objective function. A natural way is to assign an extra cost for errors to change the objective function to

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\sum_{i=1}^{N}\zeta_i\right); C \geq 0$$

where C is a user specified penalty parameter. This parameter is a trade-off parameter between margin and mistakes.

The parameter C trades off complexity, as measured by norm of weight vector, and data misfit, as measured by the number of nonseperable points. Note that we are penalizing not only the misclassified points but also the ones in the margin for better generalization. Increasing C corresponds to assigning a high penalty to errors, simultaneously resulting in larger weights. The width of the soft-margin can be controlled by penalty parameter C.

The new optimization problem becomes

$$\begin{aligned} &\textit{minimize } \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\zeta_i \\ &\text{subject to } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1 - \zeta_i;\ i = 1, \ldots, N \\ &\qquad\qquad \zeta_i \geq 0;\ i = 1, \ldots, N \end{aligned} \tag{4.48a}$$

This formulation is called the *soft-margin SVM*.

Proceeding in the manner similar to that described earlier for separable case, we may formulate the dual problem for nonseparable patterns as follows.

The Lagrangian

$$L(\mathbf{w}, w_0, \boldsymbol{\zeta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \tfrac{1}{2}\,\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\zeta_i - \sum_{i=1}^{N}\lambda_i[y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) - 1 + \zeta_i] - \sum_{i=1}^{N}\mu_i\zeta_i \tag{4.49}$$

where $\lambda_i, \mu_i \geq 0$ are the dual variables.

The KKT conditions for optimality are as follows:

$$\text{(i)}\quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N}\lambda_i y^{(i)}\mathbf{x}^{(i)} = 0$$

$$\frac{\partial L}{\partial w_0} = -\sum_{i=1}^{N}\lambda_i y^{(i)} = 0$$

$$\frac{\partial L}{\partial \zeta_i} = C - \lambda_i - \mu_i = 0;\ i = 1, \ldots, N$$

$$\text{(ii)}\quad y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) - 1 + \zeta_i \geq 0;\ i = 1, \ldots, N \tag{4.50}$$

$$\zeta_i \geq 0;\ i = 1, \ldots, N$$

$$\text{(iii)}\quad \lambda_i \geq 0;\ i = 1, \ldots, N$$

$$\mu_i \geq 0;\ i = 1, \ldots, N$$

$$\text{(iv)}\quad \lambda_i(y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) - 1 + \zeta_i) = 0;\ i = 1, \ldots, N$$

$$\mu_i\zeta_i = 0;\ i = 1, \ldots, N$$

We substitute the relations in condition (i) of KKT conditions (4.50) into the Lagrangian (4.49) to obtain dual objective function. From the relation $C - \lambda_i - \mu_i = 0$, we can deduce that $\lambda_i \leq C$ because $\mu_i \geq 0$. Thus, the dual formulation of the *soft-margin SVM* is

$$\textit{maximize}\ L_*(\boldsymbol{\lambda}) = \sum_{i=1}^{N}\lambda_i - \tfrac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}\lambda_i\lambda_k y^{(i)}y^{(k)}\mathbf{x}^{(i)T}\mathbf{x}^{(k)}$$

$$\text{subject to}\ \sum_{i=1}^{N}\lambda_i y^{(i)} = 0 \tag{4.51}$$

$$0 \leq \lambda_i \leq C;\ i = 1, \ldots, N$$

Interestingly, ζ_i and μ_i are not in the dual objective function; the objective function is identical to that for the separable case. The only difference is the constraint $\lambda_i \leq C$ (inferred from $C - \lambda_i - \mu_i = 0$ and $\mu_i \geq 0$). The dual problem (4.51) can also be solved numerically, and the resulting λ_i values are then used to compute $\mathbf{w}$ and w_0. The weight vector $\mathbf{w}$ is computed using Eqn (4.36).

The bias parameter w_0 is computed using condition (iv) of KKT conditions (4.50):

$$\lambda_i(y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) - 1 + \zeta_i) = 0 \tag{4.52a}$$

$$\mu_i\zeta_i = 0 \tag{4.52b}$$

Since we do not have values for ζ_i, we have to get around it. λ_i can have values in the interval $0 \leq \lambda_i \leq C$. We will separate it into the following three cases:

Case 1: $\lambda_i = 0$

We know that $C - \lambda_i - \mu_i = 0$. With $\lambda_i = 0$, we get $\mu_i = C$. Since $\mu_i\zeta_i = 0$ (Eqn (4.52b)), this implies that $\zeta_i = 0$; which means that the corresponding *i*th pattern is correctly classified without any error (as it would have been with hard-margin SVM). Such patterns may lie on margin hyperplanes or outside the margin. However, they do not contribute to the optimum value of **w**, as is seen from Eqn (4.36).

Case 2: $0 < \lambda_i < C$

We know that $C - \lambda_i - \mu_i = 0$. Therefore, $\mu_i = C - \lambda_i$, which means $\mu_i > 0$. Since $\mu_i\zeta_i = 0$ (Eqn (4.52b), this implies that $\zeta_i = 0$. Again the corresponding *i*th pattern is correctly classified. Also, from Eqn (4.52a), we see that for $\zeta_i = 0$ and $0 < \lambda_i < C$, $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) = 1$; so the corresponding patterns are on the margin.

Case 3: $\lambda_i = C$

With $\lambda_i = C$, $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) + \zeta_i = 1$, and $\zeta_i > 0$. But $\zeta_i \geq 0$ is a constraint of the problem. So $\zeta_i > 0$; which means that the corresponding pattern is mis-classified or lies inside the margin.

Note that support vectors have their $\lambda_i > 0$, and they define **w** as given by Eqn (4.36). We can compute **w** from the following equation (refer Eqn (4.38)):

$$\mathbf{w} = \sum_{i \in svindex} \lambda_i y^{(i)}\mathbf{x}^{(i)} \tag{4.53a}$$

where *svindex* denotes the set of indices of support vectors (patterns with $\lambda_i > 0$).

Of all the support vectors, those whose $\lambda_i < C$ (Case 2), are the ones that are on the margin, and we can use them to calculate w_0; they satisfy

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) = 1$$

We can compute w_0 from the following equation (refer Eqn (4.40)):

$$w_0 = \frac{1}{|svmindex|} \sum_{i \in svmindex} (y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}) \tag{4.53b}$$

where *svmindex* are the set of support vectors that fall on the margin.

Finally, expressions for both the decision function $g(\mathbf{x})$ and an indicator function $i_F = sgn\ (g(\mathbf{x}))$ for a soft-margin classifier are the same as for linearly separable classes (refer Eqns (4.41)):

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{4.54a}$$

$$i_F = \hat{y} = sgn\,(g(\mathbf{x})) = sgn\,(\mathbf{w}^T\mathbf{x} + w_0) \tag{4.54b}$$

The following points need attention of the reader:

- A significant property of SVM is that the solution is sparse in λ_i. Majority of the training data points exist outside the margin area and their λ_i's in the solution are 0. The data points on the margin with $\lambda_i = 0$, do not contribute to the solution either. Only those data points that are on the margin hyperplanes with $0 < \lambda_i < C$, and those mis-classified or inside the margin ($\lambda_i = C$) make contribution to the solution. In the absence of this sparsity property, SVM would prove to be impractical for huge datasets.
- Parameter C in the optimization formulation (4.51) is the regularization parameter, fine-tuned with the help of cross-validation. It defines the trade-off between margin maximization and error minimization. In case it is too big, there is high penalty for nonseparable points, and we may store several support vectors and overfit. In case it is too small, we may come across very simple solutions that underfit.
  The tuning process can be rather time-consuming for huge datasets. Many heuristic rules for selection of C have been recommended in the literature. Refer to [57] for a heuristic formula for the selection for the parameter, which has proven to be close to optimal in many practical situations. More about the difficulty associated with choice of C will appear in a later section.
- The final decision boundary is

$$\mathbf{w}^T\mathbf{x} + w_0 = 0$$

Substituting for $\mathbf{w}$ and w_0 from Eqns (4.53), we obtain

$$\left(\sum_{i \in svindex} \lambda_i y^{(i)} \mathbf{x}^{(i)}\right)^T \mathbf{x} + \frac{1}{|svmindex|}\left[\sum_{k \in svmindex}\left[y^{(k)} - \left(\sum_{i \in svindex} \lambda_i y^{(i)} \mathbf{x}^{(i)}\right)^T \mathbf{x}^{(k)}\right]\right] = 0$$

or

$$\sum_{i \in svindex} (\lambda_i y^{(i)} \mathbf{x}^{(i)^T} \mathbf{x})$$

$$+ \frac{1}{|svmindex|}\left[\sum_{k \in svmindex}\left[y^{(k)} - \left(\sum_{i \in svindex} \lambda_i y^{(i)} \mathbf{x}^{(i)^T} \mathbf{x}^{(k)}\right)\right]\right] = 0 \tag{4.55}$$

We notice that $\mathbf{w}$ and w_0 do not need to be explicitly computed. As we will see in a later section, this is crucial for using kernel functions to handle nonlinear decision boundaries.

Example 4.2

In Example 4.1, SVM (hard margin) formulation was developed in matrix form. In the following, we give matrix form of SVM (soft margin) formulation [56].

If all the data points are not linearly separable, we allow training error or in other words, allow points to lie between bounding hyperplanes and beyond. When a point, say $\mathbf{x}^{(i)} = [x_1^{(i)}\ x_2^{(i)}\ \cdots\ x_n^{(i)}]^T$

with $y^{(i)} = +1$ lies either between the bounding hyperplanes or beyond (into the region where $\mathbf{w}^T\mathbf{x}^{(i)} + w_0 \leq -1$), we add a positive quantity ζ_i to the left of inequality (refer to (4.23a)) to satisfy the constraint $\mathbf{w}^T\mathbf{x}^{(i)} + w_0 + \zeta_i \geq +1$. Similarly, when a point with $y^{(i)} = -1$ lies either between the bounding hyperplanes or beyond (into the region where $\mathbf{w}^T\mathbf{x}^{(i)} + w_0 \geq +1$), we subtract a positive quantity ζ_i from the left of the inequality (refer to (4.23a)) to satisfy the constraint $\mathbf{w}^T\mathbf{x}^{(i)} + w_0 - \zeta_i \leq -1$. For all other points, let us assume that we are adding ζ_i terms with zero values. Thus, we have the constraints (Eqn (4.47))

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1 - \zeta_i\,;\ \zeta_i \geq 0$$

The constraint equation in matrix form can be written as,

$$\mathbf{Y}(\mathbf{X}\mathbf{w} + w_0\mathbf{e}) + \boldsymbol{\zeta} \geq \mathbf{e} \tag{4.56}$$

where

$$\underset{(N\times 1)}{\boldsymbol{\zeta}} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \vdots \\ \zeta_N \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

and matrices/vectors $\mathbf{Y}$, $\mathbf{X}$, $\mathbf{w}$ and $\mathbf{e}$ have already been defined in Eqns (4.42).

The optimization problem becomes (Eqn (4.48a))

$$\underset{\mathbf{w},\boldsymbol{\zeta}}{Minimize}\left[\tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\,\mathbf{e}^T\boldsymbol{\zeta}\right]$$
$$\text{subject to } \mathbf{Y}(\mathbf{X}\mathbf{w} + w_0\mathbf{e}) + \boldsymbol{\zeta} \geq \mathbf{e}, \text{ and } \boldsymbol{\zeta} \geq \mathbf{0} \tag{4.57}$$

Here, C is a scalar value (≥ 0) that controls the trade-off between margin and errors. This C is to be supplied at the time of training. Proper choice of C is crucial for good generalization performance of the classifier. Usually, the value of C is obtained by trial-and-error with cross-validation.

Minimization of the quantity $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\,\mathbf{e}^T\boldsymbol{\zeta}$ with respect to $\mathbf{w}$ and $\boldsymbol{\zeta}$ causes maximum separation between the bounding planes with minimum number of points crossing their respective bounding planes.

Proceeding in the manner similar to that described in Example 4.1 for hard-margin SVM, we may formulate the dual problem for soft-margin SVM as follows (Eqn (4.51)):

$$\underset{\boldsymbol{\lambda}}{Minimize}\left\{-\mathbf{e}^T\boldsymbol{\lambda} + \tfrac{1}{2}\boldsymbol{\lambda}^T\mathbf{Q}\,\boldsymbol{\lambda}\right\}$$
$$\text{subject to } \mathbf{y}^T\boldsymbol{\lambda} = 0,\ \mathbf{0} \leq \boldsymbol{\lambda} \leq C\mathbf{e} \tag{4.58}$$

where $\boldsymbol{\lambda} = [\lambda_1\ \lambda_2\ \ldots\ \lambda_N]^T$ are dual variables, and all other matrices/vectors have been defined earlier in Example 4.1.

The main difference between soft-margin and hard-margin SVM classifiers is that in case of soft margin, the Lagrange multipliers λ_i are bounded; $0 \leq \lambda_i \leq C$. We will separate the bounds on λ_i into three cases:

1. $\lambda_i = 0$. This leads to $\zeta_i = 0$, implying that the corresponding ith pattern is correctly classified.
2. $0 < \lambda_i < C$. This also leads to $\zeta_i = 0$, implying that corresponding ith pattern is correctly classified. The points with $\zeta_i = 0$ and $0 < \lambda_i < C$, fall on the bounding planes of the class to which the points belong. These are the support vectors that lie on the margin.
3. $\lambda_i = C$. In this case, $\zeta_i > 0$; this implies that the corresponding ith pattern is mis-classified or lies inside the margin. Since $\lambda_i \neq 0$, these are also support vectors

Once we obtain Lagrange multipliers, $\boldsymbol{\lambda}$, using quadratic programming, we can compute $\mathbf{w}$ and w_0 using Eqns (4.53).

As we will see in Section 4.7, SVM formulation for nonlinear classifiers is similar to the one given in this example. There, we will consider a toy dataset to illustrate numerical solution of SVM (soft-margin) problem.

4.6 KERNEL-INDUCED FEATURE SPACES

So far we have considered parametric models for classification (and regression) in which the form of mapping from input vector $\mathbf{x}$ to class label y (or continuous real-valued output y) is linear. One common strategy in machine learning is to learn nonlinear functions with a linear machine. For this, we need to change the representation of the data:

$$\mathbf{x} = \{x_1, \ldots, x_n\} \Rightarrow \boldsymbol{\phi}(\mathbf{x}) = \{\phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})\} \tag{4.59}$$

where $\boldsymbol{\phi}(.)$ is a nonlinear map from input feature space to some other feature space. The selection of $\boldsymbol{\phi}$ is constrained to yield a new feature space in which the linear machine can be used. Hence, the set of hypotheses we consider will be functions of the type

$$g(\boldsymbol{\phi}(\mathbf{x}), \mathbf{w}) = \sum_{l=1}^{m} w_l \, \phi_l(\mathbf{x}) + w_0 = \mathbf{w}^T \boldsymbol{\phi} + w_0 \tag{4.60}$$

where $\mathbf{w}$ is now the m-dimensional weight parameter.

This means that we will build nonlinear machines in two steps:

(i) first a fixed nonlinear mapping transforms the data into a new feature space, and
(ii) then a linear machine is used to classify the data in the new feature space.

By selecting m functions $\phi_l(\mathbf{x})$ judiciously, one can approximate any nonlinear discriminant function in $\mathbf{x}$ by such a linear expansion. The resulting discriminant function is not linear in $\mathbf{x}$, but it is linear in $\boldsymbol{\phi}(\mathbf{x})$. The m functions merely map points in the n-dimensional $\mathbf{x}$-space to points in the m-dimensional $\boldsymbol{\phi}$-space. The homogeneous discriminant function $\mathbf{w}^T\boldsymbol{\phi} + w_0$ separates points in the transformed space by a hyperplane. Thus, the mapping from $\mathbf{x}$ to $\boldsymbol{\phi}$ reduces the problem of finding nonlinear discriminant function in $\mathbf{x}$ to one of finding linear discriminant function in $\boldsymbol{\phi}(\mathbf{x})$. With a clever choice of nonlinear $\boldsymbol{\phi}$-functions, we can obtain arbitrary nonlinear decision regions in $\mathbf{x}$-space, in particular those leading to minimum errors.

The central difficulty is naturally choosing the appropriate m-dimensional mappings $\boldsymbol{\phi}(\mathbf{x})$ of the original input feature vectors $\mathbf{x}$. This approach of the design of nonlinear machines is linked to our expectation that the patterns which are not linearly separable in $\mathbf{x}$-space become linearly separable in $\boldsymbol{\phi}$-space. This expectation is based on the observations that by selecting $\phi_l(\mathbf{x})$; $l = 1, \ldots, m$, functions judiciously and letting m sufficiently large, one can approximate any nonlinear discriminant function in $\mathbf{x}$ by linear expansion (4.60).

In the sequel, we will first try to justify our expectations that by going to a higher dimensional space, the classification task may be transformed into a linear one, and then study popular alternatives for the choice of functions $\phi_l(\cdot)$.

Let us first attempt to intuitively understand why going to a higher dimensional space increases the chances of a linear separation. For linear function expansions such as,

$$g(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{n} w_j x_j + w_0$$

or

$$g(\boldsymbol{\phi}(\mathbf{x}), \mathbf{w}) = \sum_{l=1}^{m} w_j \, \phi_l(\mathbf{x}) + w_0$$

the VC dimension increases as the number of weight parameters increases. Using a transformation $\boldsymbol{\phi}(\mathbf{x})$ to a higher dimensional $\boldsymbol{\phi}$-space ($m > n$), typically amounts to increasing the *capacity* (refer to Section 2.3) of the learning machine, and rendering problems separable that are not linearly separable to start with.

Cover's theorem [58] formalizes the intuition that the *number of linear separations increases with the dimensionality*. The number of possible linear separations of *well distributed N* points in an n-dimensional space ($N > n + 1$), as per this theorem, equals

$$2 \sum_{j=0}^{n} \binom{N-1}{j} = 2 \sum_{j=0}^{n} \frac{(N-1)!}{(N-1-j)!\, j!}$$

The more we increase n, the more terms there are in the sum, and thus the larger is the resulting number.

The linear separability advantage of high-dimensional feature spaces comes with a cost: the *computational complexity*. The approach of selecting $\phi_l(\mathbf{x})$; $l = 1, \ldots, m$, with $m \to \infty$ may not work; such a classifier would have too many free parameters to be determined from a limited number of training data. Also, if the number of parameters is too large relative to the number of training examples, the resulting model will overfit the data, affecting the generalization performance.

So there are problems. First how do we choose the nonlinear mapping to a higher dimensional space? Second, the computations involved will be costly. It so happens that in solving the quadratic optimization problem of the linear SVM (i.e., when searching for a linear SVM in the new higher dimensional space), the training tuples appear only in the form of dot products (refer to Eqn (4.51)):

$$\langle \boldsymbol{\phi}(\mathbf{x}^{(i)}), \boldsymbol{\phi}(\mathbf{x}^{(k)}) \rangle = [\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T \, [\boldsymbol{\phi}(\mathbf{x}^{(k)})]$$

Note that the dot product requires one multiplication and one addition for each of the m-dimensions. Also, we have to compute the dot product with every one of the support vectors. In training, we have to compute a similar dot product several times. The dot product computation required is very heavy and costly. We need a *trick* to avoid the dot product computations.

Luckily, we can use a math trick. Instead of computing the dot product on the transformed data tuples, it turns out that it is mathematically equivalent to instead apply a *kernel function*, $K(\mathbf{x}^{(i)}, \mathbf{x}^{k})$, to the original input data. That is,

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = \langle \boldsymbol{\phi}(\mathbf{x}^{(i)}), \boldsymbol{\phi}(\mathbf{x}^{(k)}) \rangle$$

In other words, everywhere that $\langle \boldsymbol{\phi}(\mathbf{x}^{(i)}), \boldsymbol{\phi}(\mathbf{x}^{(k)}) \rangle$ appears in the training algorithm, we can replace it with $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$. In this way, all calculations are made in the original input space, which is of potentially much lower dimensionality.

Another feature of the *kernel trick* addresses the problem of choosing nonlinear mapping $\boldsymbol{\phi}(\mathbf{x})$. It turns out that we don't even have to know what the mapping is. That is, admissible kernel substitutions $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$ can be determined without the need of first selecting a mapping function $\boldsymbol{\phi}(\mathbf{x})$.

Let us study the kernel trick in a little more detail for appreciating this important property of SVMs to solve nonlinear classification problems.

Example 4.3

Given a database with feature vectors $\mathbf{x} = [x_1\ x_2]^T$. Consider the nonlinear mapping

$$(x_1, x_2) \xrightarrow{\ \phi\ } (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \phi_3(\mathbf{x}), \phi_4(\mathbf{x}), \phi_5(\mathbf{x}), \phi_6(\mathbf{x}))$$

$$\boldsymbol{\phi}(\mathbf{x}) = [1\ \sqrt{2}x_1\ \sqrt{2}x_2\ \sqrt{2}x_1x_2\ x_1^2\ x_2^2]^T$$

For this mapping, the dot product is

$$[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T \boldsymbol{\phi}(\mathbf{x}^{(k)}) = [1\ \sqrt{2}x_1^{(i)}\ \sqrt{2}x_2^{(i)}\ \sqrt{2}x_1^{(i)}x_2^{(i)}\ (x_1^{(i)})^2\ (x_2^{(i)})^2] \begin{bmatrix} 1 \\ \sqrt{2}x_1^{(k)} \\ \sqrt{2}x_2^{(k)} \\ \sqrt{2}x_1^{(k)}x_2^{(k)} \\ (x_1^{(k)})^2 \\ (x_2^{(k)})^2 \end{bmatrix}$$

$$= 1 + 2x_1^{(i)}x_1^{(k)} + 2x_2^{(i)}x_2^{(k)} + 2x_1^{(i)}x_2^{(i)}x_1^{(k)}x_2^{(k)} + (x_1^{(i)})^2(x_1^{(k)})^2 + (x_2^{(i)})^2(x_2^{(k)})^2$$

$$= [(\mathbf{x}^{(i)})^T \mathbf{x}^{(k)} + 1]^2$$

The inner product of vectors in new (higher dimensional) space has been expressed as a function of the inner product of the corresponding vectors in the original (lower dimensional) space. The *kernel function* is

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = [\mathbf{x}^{(i)T}\, \mathbf{x}^{(k)} + 1]^2$$

Constructing Kernels

In order to exploit *kernel trick*, we need to be able to construct valid kernel functions. A straightforward way of computing kernel $K(\cdot)$ from a map $\boldsymbol{\phi}(\cdot)$ is to choose a feature mapping $\boldsymbol{\phi}(\mathbf{x})$ and then use it to find the corresponding kernel:

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = [\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T \boldsymbol{\phi}(\mathbf{x}^{(k)}) \tag{4.61}$$

The kernel trick makes it possible to map the data *implicitly* into a higher dimensional feature space, and to train a linear machine in such a space, potentially side-stepping the computational problems inherent in this high-dimensional space. One of the curious facts about using a kernel is that we do not need to know the underlying feature map $\boldsymbol{\phi}(\cdot)$ in order to be able to learn in the new feature space. Kernel trick shows a way of computing dot products in these high-dimensional spaces without *explicitly* mapping into the spaces.

The straightforward way of computing kernel $K(\cdot)$ from the map $\boldsymbol{\phi}(\cdot)$ given in Eqn (4.61), can be inverted, i.e., we can choose a kernel rather than the mapping and apply it to the learning algorithm directly.

We can, of course, first propose a kernel function and then expand it to identify $\boldsymbol{\phi}(\mathbf{x})$. Identification of $\boldsymbol{\phi}$ is not needed if we can show whether the proposed function is a kernel or not without the need of corresponding mapping function.

Mercer's Theorem: In the following, we introduce Mercer's theorem, which provides a test whether a function $K(\mathbf{x}^{(i)},\mathbf{x}^{(k)})$ constitutes a valid kernel without having to construct the function $\boldsymbol{\phi}(\mathbf{x})$ explicitly.

Let $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$ be a *symmetric* function on the finite input space. Then $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$ is a kernel function if and only if the matrix

$$\mathbf{K} = \begin{bmatrix} K(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & K(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \cdots & K(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}) \\ \vdots & \vdots & K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) & \vdots \\ K(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}) & K(\mathbf{x}^{(N)}, \mathbf{x}^{(2)}) & \cdots & K(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}) \end{bmatrix} \tag{4.62}$$

is *positive semidefinite*.

For any function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$ satisfying Mercer's theorem, there exists a space in which $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$ defines an inner product. What, however, Mercer's theorem does not disclose to us is how to find this space. That is, we do not have a general tool to construct the mapping function $\boldsymbol{\phi}(\cdot)$ once we know the kernel function (in simple cases, we can expand $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$ and rearrange it to give $[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T \boldsymbol{\phi}(\mathbf{x}^{(k)})$). Furthermore, we lack the means to know the dimensionality of the space, which can even be infinite. For further details, see [53].

What are some of the kernel functions that could be used? Properties of the kinds of kernel functions that could be used to replace the dot product scenario just described, have been studied. The most popular general-purpose kernel functions are:

Polynomial kernel of degree d

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = (\mathbf{x}^{(i)T}\, \mathbf{x}^{(k)} + c)^d;\ c > 0,\ d \geq 2 \tag{4.63}$$

Gaussian radial basis function kernel

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(k)}\|^2}{2\sigma^2}\right);\ \sigma > 0 \tag{4.64}$$

The feature vector that corresponds to the Gaussian kernel has infinite dimensionality.

Sigmoidal kernel

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = \tanh\,(\beta\, \mathbf{x}^{(i)T}\, \mathbf{x}^{(k)} + \gamma) \tag{4.65}$$

for appropriate values of β and γ so that Mercer's conditions are satisfied. One possibility is $\beta = 2$, $\gamma = 1$.

Each of these results in a different nonlinear classifier in (the original) input space.

There are no golden rules for determining which admissible kernel will result in the most accurate SVM. In practice, the kernel chosen does not generally make a large difference in resulting accuracy. SVM training always finds a global solution, unlike neural networks (discussed in the next chapter) where many local minima usually exist.

4.7 NONLINEAR CLASSIFIER

The SVM formulations debated till now, need Class 1 and Class 2 examples to be capable of linear representation, that is, with the decision boundary being a hyperplane. But, for several real-life datasets, the decision boundaries are nonlinear. To deal with nonlinear case, the formulation and solution methods employed for the linear case are still applicable. Only input data is transformed from its original space into another space (generally, a much higher dimensional space) so that a linear decision boundary can separate Class 1 examples from Class 2 in the transformed space, called the *feature space*. The original data space is known as the *input space*.

Let the set of training (data) examples be

$$\mathcal{D} = \{\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\} \tag{4.66}$$

where $\mathbf{x} = [x_1\, x_2\, \ldots\, x_n]^T$.

Figure 4.13 illustrates the process. In the input space, the training examples cannot be linearly separated; in the feature space, they can be separated linearly.

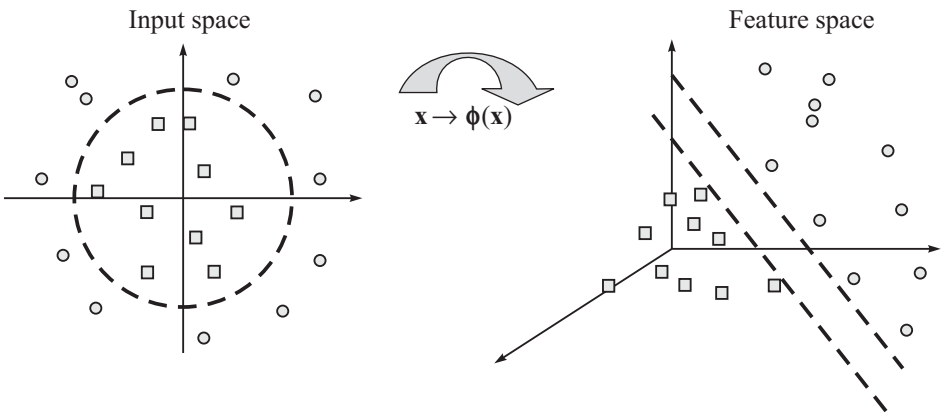


Figure 4.13 Transformation from the input space to feature space

With the transformation, the optimization problem in (4.48a) becomes

$$\begin{aligned} &\textit{minimize } \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\zeta_i \\ &\text{subject to } y^{(i)}(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}^{(i)}) + w_0) \geq 1 - \zeta_i\,;\; i = 1, \ldots, N \\ &\zeta_i \geq 0;\; i = 1, \ldots, N \end{aligned} \tag{4.67}$$

The corresponding dual is (refer to (4.51))

$$\begin{aligned} &\textit{minimize } L_*(\boldsymbol{\lambda}) = \sum_{i=1}^{N}\lambda_i - \tfrac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}\lambda_i\,\lambda_k y^{(i)}y^{(k)}[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T\boldsymbol{\phi}(\mathbf{x}^{(k)}) \\ &\text{subject to } \sum_{i=1}^{N}\lambda_i y^{(i)} = 0 \\ &0 \leq \lambda_i \leq C;\; i = 1, \ldots, N \end{aligned} \tag{4.68}$$

The potential issue with this strategy is that there are chances of it suffering from the curse of dimensionality. The number of dimensions in the feature space may be very large with certain useful transformations, even with a reasonable number of attributes in the input space. Luckily, explicit transformations are not required as we see that for the dual problem (4.68), the building of the decision boundary only requires the assessment of $[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T\boldsymbol{\phi}(\mathbf{x})$ in the feature space. With reference to (4.55), we have the following decision boundary in the feature space:

$$\sum_{i=1}^{N}\lambda_i y^{(i)}[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T\boldsymbol{\phi}(\mathbf{x}) + w_0 = 0 \tag{4.69}$$

Thus, if we have to compute $[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T\boldsymbol{\phi}(\mathbf{x})$ in the feature space using the input vectors $\mathbf{x}^{(i)}$ and $\mathbf{x}$ directly, then we would not need to know the feature vector $\boldsymbol{\phi}(\mathbf{x})$ or even the mapping $\boldsymbol{\phi}$ itself. In SVM, this is done through the use of *kernel functions*, denoted by K (details given in the previous section):

$$K(\mathbf{x}^{(i)}, \mathbf{x}) = [\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T\boldsymbol{\phi}(\mathbf{x}) \tag{4.70}$$

We replace $[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T\boldsymbol{\phi}(\mathbf{x})$ in (4.69) with kernel (4.70). We would never need to explicitly know what $\boldsymbol{\phi}$ is.

Example 4.4

The basic idea in designing nonlinear SVMs is to map input vectors $\mathbf{x} \in \Re^n$ into higher dimensional feature-space vectors $\mathbf{z} \in \Re^m$; $m > n$. $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$ where $\boldsymbol{\phi}$ represents a mapping $\Re^n \rightarrow \Re^m$. Note that input space is spanned by components x_j; $j = 1, \ldots, n$, of an input vector $\mathbf{x}$, and feature space is spanned by components $\mathbf{z}_l$; $l = 1, \ldots, m$, of vector $\mathbf{z}$. By performing such a mapping, we expect that in feature space, the learning algorithm will be able to linearly separate the mapped data by applying the linear SVM formulation. This approach leads to a solution of a quadratic optimization problem with inequality constraints in $\mathbf{z}$-space. The solution for an *indicator function*, *sgn* $(\mathbf{w}^T\mathbf{z} + w_0)$, which is a linear classifier in feature space, creates a nonlinear separating hypersurface in the original input space.

There are two basic problems in taking this approach when mapping an input $\mathbf{x}$-space into higher-order $\mathbf{z}$-space:

1. Choice of $\boldsymbol{\phi}(\mathbf{x})$, which should result in a rich class of decision hypersurfaces.
2. Calculation of the scalar product $\mathbf{z}^T\mathbf{z}$, which can be computationally very discouraging if the feature-space dimension m is very large.

The explosion in dimensionality from n to m can be avoided in calculations by noticing that in the quadratic optimization problem (Eqn (4.51)), training data only appear in the form of scalar products $\mathbf{x}^T\mathbf{x}$. These products are replaced by scalar products $\mathbf{z}^T\mathbf{z}$ in feature space, and the latter are expressed by using a *symmetric* kernel function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$ that results in *positive semidefinite* kernel matrix (Eqn (4.62))

$$\mathbf{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = \begin{bmatrix} K(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & K(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \cdots & K(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}) \\ \vdots & \vdots & K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) & \vdots \\ K(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}) & K(\mathbf{x}^{(N)}, \mathbf{x}^{(2)}) & \cdots & K(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}) \end{bmatrix} \tag{4.71}$$

The required scalar products $\mathbf{z}^T\mathbf{z}$ in feature space are calculated directly by computing kernels $\mathbf{K}(\mathbf{x}^{(i)},\mathbf{x}^{(k)})$ for given training data vectors in an input space. In this way, we bypass the computational complexity of an extremely high dimensionality of feature space. By applying kernels, we do not even have to know what the actual mapping $\boldsymbol{\phi}(\mathbf{x})$ is. Thus, using the chosen kernel $\mathbf{K}(\mathbf{x}^{(i)},\mathbf{x}^{(k)})$, an SVM can be constructed that operates in an infinite dimensional space.

Another problem with kernel-induced nonlinear classification approach is regarding the choice of a particular type of kernel function. There is no clear-cut answer. No theoretical proofs yet exist supporting applications for any particular type of kernel function. For the time being, one can only suggest that various models be tried on a given dataset and that the one with the best generalization capacity be chosen.

The learning algorithm for nonlinear soft-margin SVM classifier has already been given. Let us give the matrix formulation here, which is helpful in using standard quadratic optimization software.

In the matrix form, nonlinear SVM (soft margin) formulation is (Eqn (4.58))

$$\underset{\boldsymbol{\lambda}}{Minimize}\left\{-\mathbf{e}^T\boldsymbol{\lambda}+\tfrac{1}{2}\boldsymbol{\lambda}^T\mathbf{Q}\,\boldsymbol{\lambda}\right\} \tag{4.72}$$

$$\text{subject to } \mathbf{y}^T\boldsymbol{\lambda}=0,\ \mathbf{0}\le\boldsymbol{\lambda}\le C\mathbf{e}$$

Here, $\mathbf{Q} = \mathbf{YKY}$, and $\mathbf{K}$ is kernel matrix (Eqn (4.71). This formulation follows from Eqn (4.68). As an illustration, with consider toy dataset with $n = 1$.

$$\mathbf{X}=\begin{bmatrix}1\\2\\5\\6\end{bmatrix};\ \mathbf{y}=\begin{bmatrix}-1\\-1\\1\\-1\end{bmatrix}$$

For the choice of the kernel function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = (\mathbf{x}^{(i)T}\mathbf{x}^{(k)} + 1)^2$, the matrix $\mathbf{Q} = \mathbf{YKY}$ is given by:

$$\mathbf{Q}=\begin{bmatrix}4 & 9 & -36 & 49\\ 9 & 25 & -121 & 169\\ -36 & -121 & 676 & -961\\ 49 & 169 & -961 & 1369\end{bmatrix}$$

The SVM formulation with $C = 50$, yields [56]

$$\boldsymbol{\lambda} = [0 \quad 2.5 \quad 7.333 \quad 4.833]^T = [\lambda_1 \quad \lambda_2 \quad \lambda_3 \quad \lambda_4]^T.$$

From Eqn (4.55), we have,

$$\mathbf{w}=\sum_{i\in svindex}\lambda_i y^{(i)}\mathbf{z}^{(i)}\ ;\ svindex=\{2,3,4\}$$

This gives

$$\mathbf{w}^T\mathbf{z}=\sum_{i\in svindex}\lambda_i y^{(i)}\mathbf{z}^{(i)T}\mathbf{z}$$

$$= \sum_{i \in svindex} \lambda_i y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) \tag{4.73}$$

$$= \sum_{i \in svindex} \lambda_i y^{(i)} (\mathbf{x}^{(i)T} \mathbf{x} + 1)^2$$

For the given data,

$$\mathbf{w}^T \mathbf{z} = (2.5)\,(-1)\,(2x+1)^2 + (7.333)\,(1)\,(5x+1)^2 + (4.833)\,(-1)\,(6x+1)^2$$

$$= -0.667x^2 + 5.333x$$

The bias w_0 is determined from the requirement that at the support-vector points $x = 2$, 5 and 6, the outputs must be –1, +1 and –1, respectively. From Eqn (4.55), we have

$$w_0 = \frac{1}{|svmindex|}\left[\sum_{k \in svmindex}\left[y^{(k)} - \left(\sum_{i \in svindex} \lambda_i y^{(i)}\, \mathbf{z}^{(i)T} \mathbf{z}^{(k)}\right)\right]\right]$$

$$= \frac{1}{|svmindex|}\left[\sum_{k \in svmindex}\left[y^{(k)} - \left(\sum_{i \in svindex} \lambda_i y^{(i)}\, K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})\right)\right]\right] \tag{4.74}$$

$$= \frac{1}{|svmindex|}\left[\sum_{k \in svmindex}\left[y^{(k)} - \left(\sum_{i \in svindex} \lambda_i y^{(i)}\, (\mathbf{x}^{(i)T} \mathbf{x}^{(k)} + 1)^2\right)\right]\right]$$

$$= \tfrac{1}{3}[-1 - (-0.667\,(2)^2 + 5.333\,(2)) + 1 - (-0.667\,(5)^2$$

$$+ 5.333\,(5)) - 1 - (-0.667\,(6)^2 + 5.333\,(6))]$$

$$= -9$$

Therefore, the nonlinear decision function in the input space:

$$g(\mathbf{x}) = -0.667x^2 + 5.333x - 9$$

and indicator function

$$i_F = \hat{y} = sgn\,(g(\mathbf{x}))$$

$$= sgn\,(-0.667x^2 + 5.333x - 9)$$

The nonlinear decision function and the indicator function for one-dimensional data under consideration, are shown in Fig. 4.14.

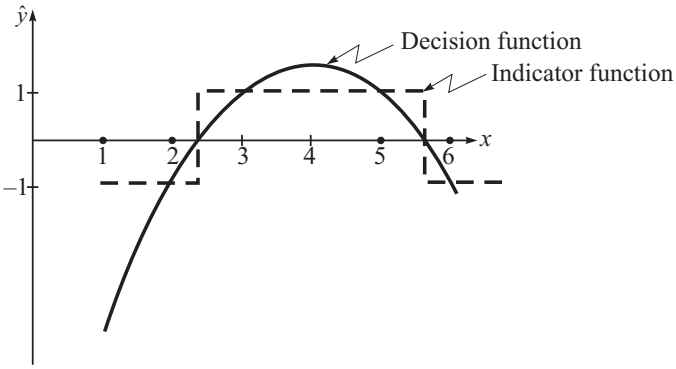


Figure 4.14 Nonlinear SV classification for Example 4.4

4.8 REGRESSION BY SUPPORT VECTOR MACHINES

Initially developed for solving classification problems, SV techniques can also be successfully applied in regression (numeric prediction) problems. Unlike classification (pattern recognition) problems where the desired outputs are discrete values: $y \in [+1, -1]$, here the system responses $y \in \Re$ are continuous values. The general regression learning problem is set as follows:

The learning machine is given N training data

$$\mathcal{D}: \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}; \mathbf{x} \in \Re^n, y \in \Re \tag{4.75}$$

where inputs $\mathbf{x}$ are n-dimensional vectors and scalar output y has continuous values. The objective is to learn the input-output relationship $\hat{y} = f(\mathbf{x})$: a nonlinear regression model.

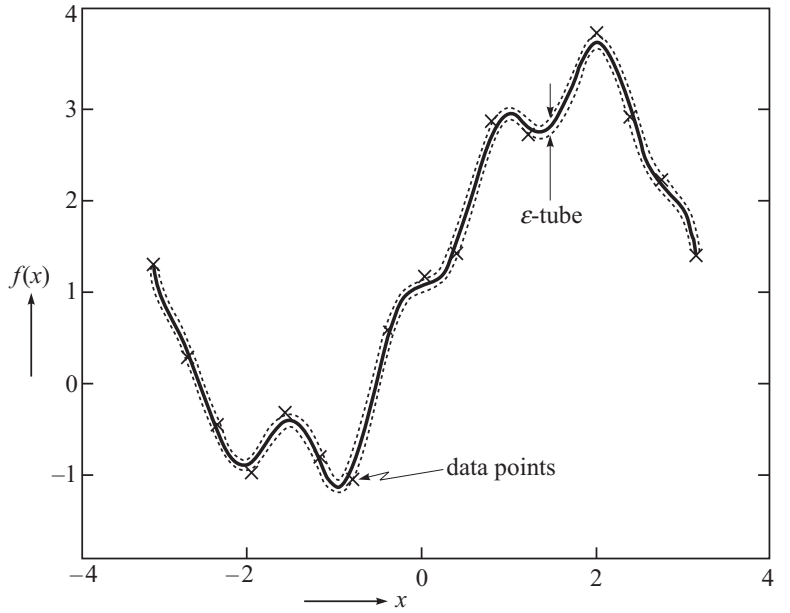
In regression, typically some measure for *error of approximation* is used instead of margin between an optimal separating hyperplane and support vectors, which was used in the design of SV classifiers. In our regression formulations described in earlier chapters, we have used *sum-of-error-squares* criterion (Section 2.7). Here, in SV regression, our goal is to find a function $\hat{y} = f(\mathbf{x})$ that has at most ε deviation (when ε is a prescribed parameter) from the actually obtained targets y for all the training data. In other words, we do not care about errors as long as they are less than ε, but any deviation larger than this will be treated as *regression error*.

To account for the regression error in our SV formulation, we use *ε-insensitive loss function*:

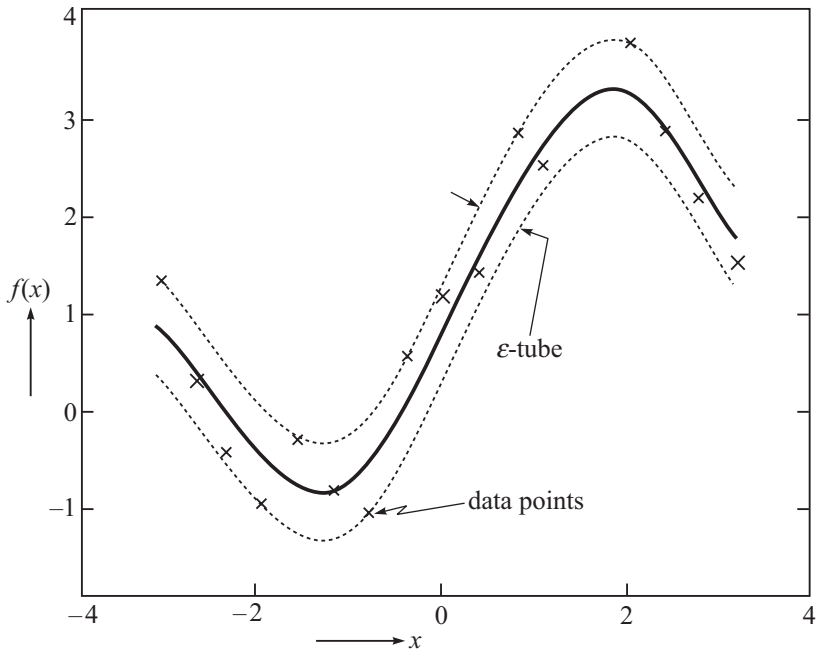
$$|y - f(\mathbf{x})|_\varepsilon \triangleq \begin{cases} 0 & \text{if } |y - f(\mathbf{x})| \le \varepsilon \\ |y - f(\mathbf{x})| - \varepsilon & \text{otherwise} \end{cases} \tag{4.76}$$

This loss (error) function defines an *ε-insensitivily zone* (*ε-tube*). We tolerate errors up to ε (data point $(\mathbf{x}^{(i)}, y^{(i)})$ within the ε-insensitivity zone or ε-tube), and the errors beyond (above/below the ε-tube) have a *linear* effect (unlike sum-of-error-squares criterion). The error function is, therefore, more tolerant to noise and is thus more *robust*. There is a region of no error, which results in sparseness.

The parameter ε defines the requirements on the accuracy of approximation. An increase in ε means a reduction in accuracy requirements; it results in smoothing effects on modeling highly noisy polluted data. On the other hand, a decrease in ε may result in complex model that overfits the data (refer to Fig. 4.15).



(a) $\varepsilon = 0.1$



(b) $\varepsilon = 0.5$

Figure 4.15 One-dimensional SV regression

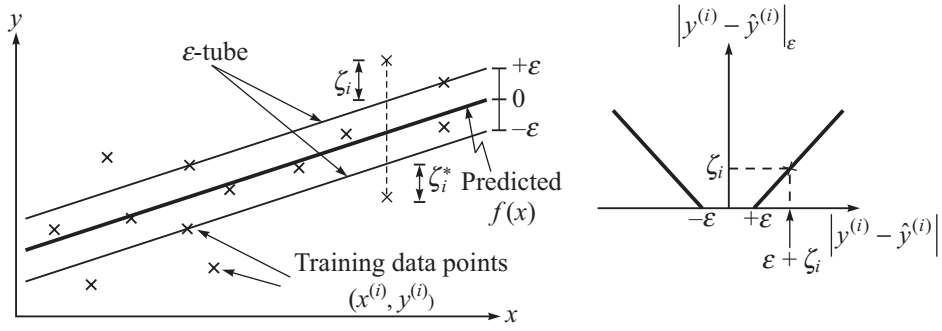
In formulating an SV algorithm for regression, the objective is to minimize the error (loss) and $||\mathbf{w}||^2$ simultaneously. The role of $||\mathbf{w}||^2$ in the objective function is to reduce model complexity; thereby preventing problem of overfitting, thus improving generalization.

4.8.1 Linear Regression

For pedagogical reasons, we begin by describing SV formulation for *linear regression*; functions $f(\cdot)$ taking the form

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0;\ \mathbf{w} \in \Re^n,\ w_0 \in \Re \tag{4.77}$$

Analogous to the 'soft margin' classifier described earlier, we introduce (non-negative) slack variables $\zeta_i, \zeta_i^*; i = 1, \ldots, N$; to measure the deviation of training examples outside the ε-insensitivity zone. Figure 4.16 shows how the ε-insensitivity zone looks like when the regression is linear.



(a) One-dimensional support vector linear regression (b) ε-insensitive loss function

Figure 4.16

If a point $(\mathbf{x}^{(i)}, y^{(i)})$ falls within the ε-tube, the associated ζ_i, ζ_i^* is zero. If it is above the tube, $\zeta_i > 0$, $\zeta_i^* = 0$ and $\zeta_i = 0$, $\zeta_i^* > 0$ if the point is below it.

$$|y^{(i)} - f(\mathbf{x}^{(i)})| - \varepsilon = \zeta_i \text{ for data 'above' the } \varepsilon\text{-insensitivity zone} \tag{4.78a}$$

$$|y^{(i)} - f(\mathbf{x}^{(i)})| - \varepsilon = \zeta_i^* \text{ for data 'below' the } \varepsilon\text{-insensitivity zone} \tag{4.78b}$$

The loss (error) is equal to zero for training data points inside the tube ($|y^{(i)} - \hat{y}^{(i)}| \le \varepsilon$); the loss is ζ_i for data 'above' the tube ($y_i - \hat{y}^{(i)} - \varepsilon = \zeta_i$), and ζ_i^* for data 'below' the tube ($\hat{y}^{(i)} - y^{(i)} - \varepsilon = \zeta_i^*$). Only the data points outside the tube contribute to the loss (error) with deviations penalized in a linear fashion.

Minimizing $||\mathbf{w}||^2$ simultaneously with minimizing the loss, results in small values for $\mathbf{w}$ and thereby a *flat* function $f(\mathbf{x})$ given by (4.77).

Analogous to the SV formulation for soft margin linear classifiers, we arrive at the following formulation of linear regression:

$$\begin{aligned} &\textit{minimize } \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}(\zeta_i + \zeta_i^*) \\ &\text{subject to } y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} - w_0 \le \varepsilon + \zeta_i;\ i = 1, \ldots, N \\ &\mathbf{w}^T\mathbf{x}^{(i)} + w_0 - y^{(i)} \le \varepsilon + \zeta_i^*;\ i = 1, \ldots, N \\ &\zeta_i,\ \zeta_i^* \ge 0;\ i = 1, \ldots, N \end{aligned} \tag{4.79}$$

Note that the constant $C > 0$, which influences a trade-off between an approximation error and the weights vector norm $\|\mathbf{w}\|$, is a design parameter chosen by the user. An increase in C penalizes larger errors (large ζ_i and ζ_i^*) and in this way leads to a decrease in approximation error. However, this can be achieved only by increasing the weights vector norm $\|\mathbf{w}\|$, which does not guarantee good generalization performance. Another design parameter chosen by the user is the required precision embodied in an ε value that defines the size of an ε-tube.

As with procedures applied to SV classifiers, the constrained optimization problem (4.79) is solved by forming the Lagrangian:

$$\begin{aligned} &L(\mathbf{w}, w_0, \boldsymbol{\zeta}, \boldsymbol{\zeta}^*, \boldsymbol{\lambda}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}, \boldsymbol{\mu}^*) \\ &= \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}(\zeta_i + \zeta_i^*) - \sum_{i=1}^{N}\lambda_i(\varepsilon + \zeta_i - y^{(i)} + \mathbf{w}^T\mathbf{x}^{(i)} + w_0) \\ &\quad - \sum_{i=1}^{N}\lambda_i^*(\varepsilon + \zeta_i^* + y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} - w_0) - \sum_{i=1}^{N}(\mu_i\zeta_i + \mu_i^*\zeta_i^*) \end{aligned} \tag{4.80}$$

where $\mathbf{w}$, w_0, ζ_i and ζ_i^* are the primal variables, and λ_i, λ_i^*, μ_i, $\mu_i^* \ge 0$ are the dual variables.

The KKT conditions are as follows:

$$\begin{aligned} &\text{(i)} \quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N}(\lambda_i - \lambda_i^*)\,\mathbf{x}^{(i)} = \mathbf{0} \\ &\qquad \frac{\partial L}{\partial w_0} = \sum_{i=1}^{N}(\lambda_i^* - \lambda_i) = 0 \\ &\qquad \frac{\partial L}{\partial \zeta_i} = C - \lambda_i - \mu_i = 0;\ i = 1, \ldots, N \\ &\qquad \frac{\partial L}{\partial \zeta_i^*} = C - \lambda_i^* - \mu_i^* = 0;\ i = 1, \ldots, N \\ &\text{(ii)} \quad \varepsilon + \zeta_i - y^{(i)} + \mathbf{w}^T\mathbf{x}^{(i)} + w_0 \ge 0;\ i = 1, \ldots, N \\ &\qquad \varepsilon + \zeta_i^* + y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} - w_0 \ge 0;\ i = 1, \ldots, N \\ &\qquad \zeta_i,\ \zeta_i^* \ge 0;\ i = 1, \ldots, N \\ &\text{(iii)} \ \lambda_i,\ \lambda_i^*,\ \mu_i,\ \mu_i^* \ge 0;\ i = 1, \ldots, N \end{aligned} \tag{4.81}$$

(iv) $\lambda_i(\varepsilon + \zeta_i - y^{(i)} + \mathbf{w}^T\mathbf{x}^{(i)} + w_0) = 0;\ i = 1, \ldots, N$

$\lambda_i^*(\varepsilon + \zeta_i^* + y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} - w_0) = 0;\ i = 1, \ldots, N$

$\mu_i\zeta_i = 0;\ i = 1, \ldots, N$

$\mu_i^*\zeta_i^* = 0;\ i = 1, \ldots, N$

Substituting the relations in condition (i) of KKT conditions (4.81) into Lagrangian (4.80), yields the dual objective function. The procedure is parallel to what has been followed earlier. The resulting dual optimization problem is

$$\textit{maximize}\ L_*(\boldsymbol{\lambda}, \boldsymbol{\lambda}^*) = -\varepsilon\sum_{i=1}^{N}(\lambda_i + \lambda_i^*) + \sum_{i=1}^{N}(\lambda_i - \lambda_i^*)y^{(i)} - \tfrac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}(\lambda_i - \lambda_i^*)(\lambda_k - \lambda_k^*)\,\mathbf{x}^{(i)T}\mathbf{x}^{(k)} \tag{4.82}$$

$$\text{subject to}\ \sum_{i=1}^{N}(\lambda_i - \lambda_i^*) = 0;\ \lambda_i, \lambda_i^* \in [0, C]$$

From condition (i) of KKT conditions (4.81), we have,

$$\mathbf{w} = \sum_{i=1}^{N}(\lambda_i - \lambda_i^*)\,\mathbf{x}^{(i)} \tag{4.83}$$

Thus, the weight vector $\mathbf{w}$ is completely described as a linear combination of the training patterns $\mathbf{x}^{(i)}$. One of the most important properties of SVM is that the solution is sparse in λ_i, λ_i^*. For $|\hat{y}^{(i)} - y^{(i)}| < \varepsilon$, the second factor in the following KKT conditions (conditions (iv) in (4.81)):

$$\begin{aligned}\lambda_i(\varepsilon + \zeta_i - y^{(i)} + \mathbf{w}^T\mathbf{x}^{(i)} + w_0) &= \lambda_i(\varepsilon + \zeta_i - y^{(i)} + \hat{y}^{(i)}) = 0\\ \lambda_i^*(\varepsilon + \zeta_i^* + y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} - w_0) &= \lambda_i^*(\varepsilon + \zeta_i^* + y^{(i)} - \hat{y}^{(i)}) = 0\end{aligned} \tag{4.84}$$

are nonzero; hence λ_i, λ_i^* have to be zero. This equivalently means that all the data points inside the ε-insensitive tube (a large number of training examples belong to this category) have corresponding λ_i, λ_i^* equal to zero. Further, from Eqn (4.84), it follows that only for $|\hat{y}^{(i)} - y^{(i)}| \geq \varepsilon$, the dual variables λ_i, λ_i^* may be nonzero. Since there can never be a set of dual variables λ_i, λ_i^* which are both simultaneously nonzero, as this would require slacks in both directions ('above' the tube and 'below' the tube), we have $\lambda_i \times \lambda_i^* = 0$.

From conditions (i) and (iv) of KKT conditions (4.81), it follows that

$$\begin{aligned}(C - \lambda_i)\zeta_i &= 0\\ (C - \lambda_i^*)\zeta_i^* &= 0\end{aligned} \tag{4.85}$$

Thus, the only samples $(\mathbf{x}^{(i)}, y^{(i)})$ with corresponding λ_i, $\lambda_i^* = C$ lie outside the ε-insensitive tube around f. For λ_i, $\lambda_i^* \in (0, C)$, we have ζ_i, $\zeta_i^* = 0$ and moreover the second factor in Eqn (4.84) has to vanish. Hence, w_0 can be computed as follows:

$$w_0 = y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} - \varepsilon \quad \text{for} \quad \lambda_i \in (0, C)$$
$$w_0 = y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} + \varepsilon \quad \text{for} \quad \lambda_i^* \in (0, C) \tag{4.86}$$

All the data points with $\lambda_i, \lambda_i^* \in (0, C)$ may be used to compute w_0, and then their average taken as the final value for w_0.

Once we solve the quadratic optimization problem for $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^*$, we see that all instances that fall in the ε-tube have $\lambda_i = \lambda_i^* = 0$; these are the instances that are fitted with enough precision. The support vectors satisfy either $\lambda_i > 0$ or $\lambda_i^* > 0$, and are of two types. They may be instances that are on the boundary of the tube (either λ_i or λ_i^* is between 0 and C), and we use these to calculate w_0. Instances that fall outside the ε-tube ($\lambda_i = C$) are of second type of support vectors. For these instances, we do not have a good fit.

Using condition (i) of KKT conditions (4.81), we can write the fitted line as a weighted sum of the support vectors;

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = \sum_{i \in svindex} (\lambda_i - \lambda_i^*)\mathbf{x}^{(i)T}\mathbf{x} + w_0 \tag{4.87a}$$

where *svindex* denotes the set of indices of support vectors. Note that for each $i \in svindex$, one element of the pair (λ_i, λ_i^*) is zero.

The parameter w_0 may be obtained from either of the equations in (4.86). If the former one is used for which $i \in$ set of instances that correspond to support vectors on the upper boundary ($\lambda_i \in (0, C)$) of the ε-tube (let us denote these points as belonging to the set *svm1index* of indices of support vectors that fall on the upper boundary), then we have (refer to Eqn (4.55)),

$$w_0 = \frac{1}{|svm1index|}\left[\sum_{k \in svm1index}\left[y^{(k)} - \varepsilon - \left(\sum_{i \in svindex}(\lambda_i - \lambda_i^*)\mathbf{x}^{(i)T}\mathbf{x}^{(k)}\right)\right]\right] \tag{4.87b}$$

4.8.2 Nonlinear Regression

For nonlinear regression, the quadratic optimization problem follows from Eqn (4.82):

$$\begin{aligned} \textit{maximize } L_*(\boldsymbol{\lambda}, \boldsymbol{\lambda}^*) = &-\varepsilon\sum_{i=1}^{N}(\lambda_i + \lambda_i^*) + \sum_{i=1}^{N}(\lambda_i - \lambda_i^*)y^{(i)} \\ &-\tfrac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}(\lambda_i - \lambda_i^*)(\lambda_k - \lambda_k^*)\mathbf{z}^{(i)T}\mathbf{z}^{(k)} \end{aligned}$$
$$\text{subject to } \sum_{i=1}^{N}(\lambda_i - \lambda_i^*) = 0;\ \lambda_i\ \lambda_i^* \in [0, C] \tag{4.88}$$

where $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$

The dot product $\mathbf{z}^{(i)T}\mathbf{z}^{(k)} = [\boldsymbol{\phi}(x^{(i)})]^T\boldsymbol{\phi}(\mathbf{x}^{(k)})$ in Eqn (4.88) can be replaced with a kernel $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$.

The nonlinear regression function (refer to Eqn (4.87a))

$$f(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + w_0 = \sum_{i \in svindex} (\lambda_i - \lambda_i^*)[\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T \boldsymbol{\phi}(\mathbf{x}) + w_0$$

$$= \sum_{i \in svindex} (\lambda_i - \lambda_i^*) K(\mathbf{x}^{(i)}, \mathbf{x}) + w_0 \tag{4.89}$$

The parameter w_0 for this nonlinear regression solution may be obtained from either of the equations in (4.86). If the former one is used for which $i \in$ set of instances that correspond to support vectors on the upper boundary ($\lambda_i \in (0, C)$) of the ε-tube (let us denote these points as belonging to the set *svm1index*), then we have (refer to Eqn (4.87b))

$$w_0 = \frac{1}{|svm1index|}\left[\sum_{k \in svm1index}\left[y^{(k)} - \varepsilon - \left(\sum_{i \in svindex} (\lambda_i - \lambda_i^*) K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})\right)\right]\right] \tag{4.90}$$

Regression problems when solved using SVM algorithm presented in this section, are basically *quadratic optimization problems*. Attempting standard quadratic programming routines (e.g., MATLAB) will be a rich learning experience for the readers. To help the readers use the standard routine, we give matrix formulation of SVM regression in the example that follows.

Example 4.5

In this example, we illustrate the SVM nonlinear regressor formulation; the formulation will be described in matrix form so as to help use of a standard quadratic optimization software.

As with nonlinear classification, input vector $\mathbf{x} \in \Re^n$ are mapped into vectors $\mathbf{z}$ of a higher-dimensional feature space: $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$, where $\boldsymbol{\phi}$ represents a mapping $\Re^n \rightarrow \Re^m$. The linear regression problem is then solved in this feature space. The solution for a regression hypersurface, which is linear in a feature space, will create a nonlinear regressing hypersurface in the original input space.

It can easily be shown that $\hat{y} = \mathbf{w}^T\mathbf{z} + w_0$ is a regression expression, and with the ε-insensitive loss function, the formulation leads to the solution equations of the form (refer to (4.82))

$$\textit{Minimize } \tfrac{1}{2}\bar{\boldsymbol{\lambda}}^T \mathbf{Q}\, \bar{\boldsymbol{\lambda}} + \mathbf{g}^T \bar{\boldsymbol{\lambda}} \tag{4.91}$$

subject to $[\mathbf{e}^T \; -\mathbf{e}^T]\; \bar{\boldsymbol{\lambda}} = 0$, and $0 \le \lambda_i, \lambda_i^* \le C$; $i = 1, \ldots, N$

where

$$\mathbf{Q} = \begin{bmatrix} \mathbf{K} & -\mathbf{K} \\ -\mathbf{K} & \mathbf{K} \end{bmatrix}; \bar{\boldsymbol{\lambda}} = \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\lambda}^* \end{bmatrix}, \text{ and } \mathbf{g} = \begin{bmatrix} \varepsilon - \mathbf{y} \\ \varepsilon + \mathbf{y} \end{bmatrix}$$

K as given in Eqn (4.71),

$$\boldsymbol{\lambda} = [\lambda_1 \; \lambda_2 \; \ldots \; \lambda_N]^T$$

$$\varepsilon - \mathbf{y} = [\varepsilon - y^{(1)} \;\; \varepsilon - y^{(2)} \; \ldots \; \varepsilon - y^{(N)}]^T$$

After computing Lagrange multipliers λ_i and λ_i^* using a quadratic optimization routine, we find optimal desired nonlinear regression function as (Eqns (4.89–4.90))

$$
\begin{aligned}
f(\mathbf{x}) &= \sum_{i \in svindex} (\lambda_i - \lambda_i^*)\, \mathbf{z}^{(i)T}\mathbf{z} + w_0 \\
&= \sum_{i \in svindex} (\lambda_i - \lambda_i^*)\, [\boldsymbol{\phi}(\mathbf{x}^{(i)})]^T \boldsymbol{\phi}(\mathbf{x}) + w_0 \\
&= \sum_{i \in svindex} (\lambda_i - \lambda_i^*)\, K(\mathbf{x}^{(i)}, \mathbf{x}) \\
&\quad + \frac{1}{|svm1index|} \left[\sum_{k \in svm1index} \left[y^{(k)} - \varepsilon - \left(\sum_{i \in svindex} (\lambda_i - \lambda_i^*) K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) \right) \right] \right]
\end{aligned}
\tag{4.92}
$$

There are a number of learning parameters that can be utilized for constructing SV machines for regression. The two most relevant are insensitivity parameter ε and penalty parameter C. Both the parameters are chosen by the user. An increase in ε has smoothing effects on modeling highly noisy polluted data. An increase in ε means a reduction in requirements for the accuracy of approximation. We have already commented on the selection of parameter C. More on it will appear later.

The SV training works almost perfectly for not too large datasets. However, when the number of data points is large, the quadratic programming problem becomes extremely difficult to solve with standard methods. Some approaches to resolve the quadratic programming problem for large datasets have been developed. We will talk about this aspect of SV training in a later section.

4.9 DECOMPOSING MULTICLASS CLASSIFICATION PROBLEM INTO BINARY CLASSIFICATION TASKS

Support vector machines were originally designed for binary classification. Initial research attempts were directed towards making several two-class SVMs to do multiclass classification. Recently, several single-shot multiclass classification algorithms appeared in the literature.

At present, there are two types of approaches for multiclass classification. In the first approach called 'indirect methods', we construct several binary SVMs and combine the outputs for predicting the class. In the second approach called 'direct methods', we consider all in a single optimization problem. Because of computational complexity of training in the direct methods, indirect methods so far are most widely used as they do not pose any numerical difficulties while training. We limit our discussion to indirect methods.

There are two popular methods in the category of indirect methods: One-Against-All (OAA), and One-Against-One (OAO). In the general case, both OAA and OAO are special cases of *error-correcting-output codes* that decompose a multiclass problem to a set of two-class problems [59].

4.9.1 One-Against-All (OAA)

Consider the training set

$$\mathcal{D}: \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$$

where the label $y^{(i)}$ for each observation can take on any value $\{y_q\}$; $q = 1, \ldots, M$. The precise nature of the label set is not important as long as there exists a unique label for each class in the classification problem.

The task is to design M linear discirminant functions:

$$g_q(\mathbf{x}) = \mathbf{w}_q^T \mathbf{x} + w_{q0}; \; q = 1, 2, \ldots, M \tag{4.93a}$$

The decision function is given by $g_k(\mathbf{x})$, where

$$\begin{aligned} k &= arg \max_{1 \le q \le M} (g_q(\mathbf{x})) \\ &= arg \max_{1 \le q \le M} (\mathbf{w}_q^T \mathbf{x} + w_{q0}) \end{aligned} \tag{4.93b}$$

New point $\mathbf{x}$ is assigned to class k.

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 farthest from it. The design of M linear discreminants (4.93a) follows the following procedure.

In the OAA technique [53, 60], given M classes, we construct M linear decision surfaces g_1, g_2, …, g_M. Each decision surface is trained to separate one class from the others. In other words, the decision surface g_1 is trained to separate Class 1 from all other classes; the decision surface g_2 is trained to separate Class 2 from all other classes, and so on. For the classification of an unknown point, a *voting scheme* based on which of the M decision surfaces return the largest value for this unknown point, is used. We then use the decision surface that returns the largest value for the unknown point to assign this point to the class:

$$\text{Class} = arg \max_{q \in \{1, \ldots, M\}} g_q(\mathbf{x}) \tag{4.94}$$

This approach is called the *winner-takes-all* approach.

Let us examine this construction in more detail. To train M decision surfaces, we construct M binary training sets:

$$\mathcal{D}^q = \mathcal{D}_+^q \cup \mathcal{D}_-^q; \; q = 1, \ldots, M$$

where

$\mathcal{D}_+^q$: the set of all observations in $\mathcal{D}$ that are members of the class q

and

$\mathcal{D}_-^q$: the set of all remaining observations in $\mathcal{D}$, i.e., the set of all observations in $\mathcal{D}$ that are *not* members of the class q.

For convenience, we label the training set $\mathcal{D}^q$ with the class labels $\{+1, -1\}$; the label +1 is used for observations in class q, and the label –1 is used for observations that are not in class q.

We train decision surface g_q on the corresponding dataset $\mathcal{D}^q$, which gives rise to

$$g_q(\mathbf{x}) = \mathbf{w}_q^T \mathbf{x} + w_{q0}$$

The decision surface g_q: $\Re^n \rightarrow \Re$ returns a signed real value, which can be interpreted as the distance of the point $\mathbf{x} \in \Re^n$ to the decision surface. If the value returned is +ve, the point $\mathbf{x}$ is above the decision surface and is taken as a member of the class +1 with respect to the decision surface, and if the value returned is –ve, the point is below the decision surface and is considered to be a member of the class –1 with respect to the decision surface. The value returned can also be interpreted as a *confidence* value: If our decision surface returns a large +ve value for a specific point, we are quite confident that the point belongs to class +1; on the contrary, if our decision surface returns a large –ve value for a point, we can confidently say that the point does not belong to Class +1. Since our training set $\mathcal{D}^q$ for the decision surface g_q was laid out in such a way that all observations in class q are considered +ve examples, it follows that a decision surface g_k that returns the largest positive value for some point $\mathbf{x}$ among all other decision surfaces $g_1, \ldots, g_M$, assigns the point to class k with $k \in \{1, 2, \ldots, M\}$

Although the OAA classification technique has shown to be robust in real-world applications, the fact that the individual training sets for each decision surface are highly *unbalanced* could be a potential source of problems. The *pairwise classification* technique (One-Against-One (OAO) classification technique) avoids this situation by constructing decision surface for each pair of classes [53, 60].

4.9.2 One-Against-One (OAO)

In pairwise classification, we train a classifier for each pair of classes. For M classes, this results in $M(M-1)/2$ binary classifiers.

We let $g_{q,k}$ denote the decision surface that separates the pair of classes $q \in \{1, \ldots, M\}$ and $k \in \{1, \ldots, M\}$ with $q \neq k$. We label class q samples by +1 and class k samples by –1, and train each decision surface

$$g_{q,k}(\mathbf{x}) = \mathbf{w}_{q,k}^T \mathbf{x} + w_{q,k0} \tag{4.95}$$

on the data set

$$\mathcal{D}^{q,k} = \mathcal{D}^q \cup \mathcal{D}^k$$

where

$\mathcal{D}^q$: the set of all observations in $\mathcal{D}$ with the label q

and

$\mathcal{D}^k$: the set of all observations in $\mathcal{D}$ with the label k

Once we have constructed all the pairwise decision surfaces $g_{q,k}$ using the corresponding training sets $\mathcal{D}^{q,k}$, we can classify an unseen point by applying each of the $M(M-1)/2$ decision surfaces to this point, keeping track of how many times the point was assigned to what class label. The class label with the highest count is then considered the label for the unseen point.

Voting Scheme

c_q = the frequency of 'wins' for class q computed by applying $g_{q,k}$ for all $k \neq q$.

This results in a vector

$$\mathbf{c} = [c_1, \ldots, c_M]$$

of frequencies of 'wins' of each class. The final decision is made by the most frequent class:

$$\text{Class } q = arg \max_{q=1,\ldots M} c_q \tag{4.96}$$

Frequencies of 'wins' = number of votes.

There is a likelihood of a tie in the voting scheme of OAO classification. We can breakdown the tie through the interpretation of the actual values returned by decision surfaces as confidence values. On adding up absolute values of the confidence values assigned to each of the tied labels, we take the winner to be the tied label possessing the maximum sum of confidence values.

It seems that OAO classification solves our problem of unbalanced datasets. However, it solves the problem at the expense of introducing a new complication: the fact that for M classes, we have to construct $M(M-1)/2$ decision surfaces. For small M, the difference between the number of decision surfaces we have to build for the OAA and OAO techniques, is not that drastic. (For M = 4, OAA requires 4 binary classifiers and OAO requires 6). However, for large M, the difference can be quite drastic (For M = 10, OAA requires 10 binary classifiers and OAO requires 45).

The individual classifiers in OAO technique, however, are usually smaller in size (they have fewer support vectors) then they would be in the OAA approach. This is for two reasons: first, the training sets are smaller, and second, the problems to be learned are usually easier, since the classes have less overlap. Since the size of QP in each classifier is smaller, it is possible to train fast.

Nevertheless, if M is large, then the resulting OAO system may be slower than the corresponding OAA. Platt et al. [61] improved the OAO approach and proposed a method called *Directed Acyclic Graph SVM* (DAGSVM) that forms a tree-like structure to facilitate the testing phase.

4.10 VARIANTS OF BASIC SVM TECHNIQUES

Basically, support vector machines form a learning algorithm family rather than a single algorithm. The basic concept of the SVM-based learning algorithm is pretty simple: find a good learning boundary while maximizing the margin (i.e., the distance between the closest learning samples that correspond to different classes). Each algorithm that optimizes an objective function in which the maximal margin heuristic is encoded, can be considered a *variant* of basic SVM.

In the variants, improvements are proposed by researchers to gain speed, accuracy, low computer-memory requirement and ability to handle multiple classes. Every variant holds good in a particular field under particular circumstances.

Since the introduction of SVM, numerous variants have been developed. In this section, we highlight some of these which have earned popularity in their usage or are being actively researched.

Changing the Metric of Margin from L_2-norm to L_1-norm

The standard L_2-norm SVM is a widely used tool in machine learning. The L_1-norm SVM is a variant of the standard L_2-norm SVM.

The L_1-norm formulation is given by (Eqns (4.48a)):

$$\textit{minimize } \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\zeta_i$$

$$\text{subject to } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1 - \zeta_i\,;\; i = 1, \ldots, N$$

$$\zeta_i \geq 0; \qquad i = 1, \ldots, N$$

In L_2-norm SVM, the sum of squares of error (slack) variables are minimized along with the reciprocal of the square of the margin between the boundary planes. The formulation of the problem is given by:

$$\begin{aligned} &\textit{minimize } \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{N}(\zeta_i)^2 \\ &\text{subject to } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1 - \zeta_i\,;\; i = 1, \ldots, N \\ &\qquad\qquad \zeta_i \geq 0;\; i = 1, \ldots, N \end{aligned} \tag{4.97}$$

It has been argued that the L_1-norm penalty has advantages over the L_2-norm penalty under certain scenarios, especially when there are redundant noise variables. L_1-norm SVM is able to delete many noise features by estimating their coefficients by zero, while L_2-norm SVM will use all the features. When there are many noise variables, the L_2-norm SVM suffers severe damage caused by the noise features. Thus, L_1-norm SVM has inherent variable selection property, while this is not the case for L_2-norm SVM. In this book, our focus has been on L_1-norm SVM formulations. Refer to [56] for L_2-norm SVM formulations.

Replacing Control Parameter *C* in Basic SVM (*C*-SVM); $C \geq 0$, by Parameter ν (ν-SVM); $\nu \in [0, 1]$

As we have seen in the formulations of basic SVM (*C*-SVM) presented in earlier sections, *C* is a user-specified penalty parameter. It is a trade-off between margin and mistakes. Proper choice of *C* is crucial for good generalization power of the classifier. Usually, the parameter is selected by trial-and-error with cross-validation.

Tuning of parameter *C* can be quite time-consuming for large datasets. In the scheme proposed by Schölkopf et al. [62], the parameter $C \geq 0$ in the basic SVM is replaced by a parameter $\nu \in [0, 1]$. ν has been shown to be a lower bound on the fraction of support vectors and an upper bound on the fraction of instances having margin errors (instances that lie on the wrong side of the hyperplane). By playing with ν, we can control the fraction of support vectors, and this is advocated to be more intuitive than playing with *C*. However, as compared to *C*-SVM, its formulations are more complicated.

A formulation of ν-SVM is given by:

$$\textit{minimize } \tfrac{1}{2}\mathbf{w}^T\mathbf{w} - \nu\rho + \frac{1}{N}\sum_{i=1}^{N}\zeta_i$$

$$\text{subject to } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq \rho - \zeta_i\,;\; i = 1, \ldots, N \tag{4.98}$$
$$\zeta_i \geq 0;\; i = 1, \ldots, N;\; \rho \geq 0$$

Note that parameter C does not appear in this formulation; instead there is parameter ν. An additional parameter ρ also appears that is a variable of the optimization problem and scales the margin: the margin is now $2\rho/\|\mathbf{w}\|$ (refer to Eqn (4.25)).

The formulation given by (4.98) represents modification of the basic SVM classification (C-SVM) given by (4.48a) to obtain ν-SVM classification. With analogous modifications of the basic SVM regression (ε-SVM regression), we obtain ν-SVM regression.

Sequential Minimization Algorithms

Support vector machines have attracted many researchers in the last two decades due to many interesting properties they enjoy: immunity to overfitting by means of regularization, guarantees on the generalization error, robust training algorithms that are based on well established mathematical programming techniques, and above all, their success in many real-world classification problems. Despite the many advantages, basic SVM suffers from a serious drawback; it requires Quadratic Programming (QP) solver to solve the problem. The amount of computer memory needed for a standard QP solver increases exponentially with the size of the data. Therefore, the issue is whether it is possible for us to scale up the SVM algorithm for huge datasets comprising thousands and millions of instances. Many decomposition techniques have been developed to scale up the SVM algorithm.

Techniques based on decomposition, break down a large optimization problem into smaller problems, with each one involving merely some cautiously selected variables so that efficient optimization is possible. Platt's SMO algorithm (Sequential Minimal Optimization) [63] is an extreme case of the decomposition techniques developed, which works on a set of two data points at a time. Owing to the fact that the solution for a working set of two data points can be arrived at analytically, the SMO algorithm does not invoke standard QP solvers. Due to its analytical foundation, the SMO and its improved versions [64, 65, 66] are particularly simple and at the moment in the widest use. Many free software packages are available, and the ones that are most popular are *SVM light* [67] and *LIBSVM* [68].

Variants based on Trade-off between Complexity and Accuracy

Decomposition techniques handle memory issue alone by dividing a problem into a series of smaller ones. However, these smaller problems are rather time consuming for big datasets. Number of techniques for reduction in the training time have been suggested at the cost of accuracy.

Many variants have been reported in the literature. Some of the popular ones are on follows.

LS-SVM (Least Squares Support Vector Machine): It is a *Least Squares* version of the classical SVM. LS-SVM classification formulation implicitly corresponds to a regression interpretation with binary targets $y^{(i)} = \pm 1$. Proposed by Suykens and Vandewalla [69], its formulation has equality constraints; a set of linear equations has to be solved instead of a quadratic programming problem for classical SVMs.

PSVM (Proximal Support Vector Machine): Developed by Fung and Mangasarian [70], *Proximal* SVM leads to an extremely fast and simple algorithm for generating a classifier that is obtained by solving a set of linear equations. Proximal SVM is comparable with standard SVM in performance.

The key idea of PSVM is that it classifies points by assigning them to the closer of the two parallel planes that are pushed apart as far as possible. These planes are pushed apart by introducing the term $\mathbf{w}^T\mathbf{w} + w_0^2$ in the objective function of classical L_2-norm optimization problem.

LSVM (Lagrangian Support Vector Machine): A fast and simple algorithm, based on an implicit *Lagrangian* formulation of the dual of a simple reformulation of the standard quadratic program of a support vector machine, was proposed by Mangasarian [71]. This algorithm minimizes unconstrained differentiable convex function for classifying N points in a given n-dimensional input space. An iterative Lagrangian Support Vector Machine (LSVM) algorithm is given for solving the modified SVM. This algorithm can resolve problems accurately with millions of points, at a pace greater than SMO (if n is less than 100) without any optimization tools, like linear or quadratic programming solvers.

Multiclass based SVM Algorithms

Originally, the SVM was developed for binary classification; the basic idea to apply SVM technique to multiclass problems is to decompose the multiclass problem into several two-class problems that can be addressed directly using several SVMs (refer to Section 4.9). This decomposition approach gives 'indirect methods' for multiclass classification problems.

Instead of creating several binary classifiers, a more natural way is to distinguish all classes in one single optimization processing. This approach gives 'direct methods' for multiclass classification problems [72].

Weston and Watkins' Multiclass SVM: In the method (the idea is similar to OAA approach) proposed by Weston and Watkins [73], for an M-class problem, a single objective function is designed for training all M-binary classifiers simultaneously and maximizing the margin from each class to the remaining classes. The main disadvantage of this approach is that the computational time may be very high due to the enormous size of the resulting QP. The OAA approach is generally preferred over this method.

Crammer and Singer's Multiclass SVM: Crammer and Singer [74] presented another 'all-together' approach. This approach gives a compact set of constraints; however, the number of variables in its dual problem are high. This value may explode even for small datasets.

Simplified Multiclass SVM (SimMSVM): A simplified method, named SimMSVM [75], relaxes the constraints of Crammer and Singer's approach.

The support vector machine is currently considered to be the best off-the-shelf learning algorithm and has been applied successfully in various domains. Scholkopf and Smola [53] is a classic book on the subject.