7 Sparse Kernel Machines

In the previous chapter, we explored a variety of learning algorithms based on non-linear kernels. One of the significant limitations of many such algorithms is that the kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$ must be evaluated for all possible pairs $\mathbf{x}_n$ and $\mathbf{x}_m$ of training points, which can be computationally infeasible during training and can lead to excessive computation times when making predictions for new data points. In this chapter we shall look at kernel-based algorithms that have *sparse* solutions, so that predictions for new inputs depend only on the kernel function evaluated at a subset of the training data points.

We begin by looking in some detail at the *support vector machine* (SVM), which became popular in some years ago for solving problems in classification, regression, and novelty detection. An important property of support vector machines is that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum. Because the discussion of support vector machines makes extensive use of Lagrange multipliers, the reader is encouraged to review the key concepts covered in Appendix E. Additional information on support vector machines can be found in Vapnik (1995), Burges (1998), Cristianini and Shawe-Taylor (2000), Müller *et al.* (2001), Schölkopf and Smola (2002), and Herbrich (2002).

The SVM is a decision machine and so does not provide posterior probabilities. We have already discussed some of the benefits of determining probabilities in Section 1.5.4. An alternative sparse kernel technique, known as the *relevance vector machine* (RVM), is based on a Bayesian formulation and provides posterior probabilistic outputs, as well as having typically much sparser solutions than the SVM.

Section 7.2

7.1. Maximum Margin Classifiers

We begin our discussion of support vector machines by returning to the two-class classification problem using linear models of the form

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b \tag{7.1}$$

where $\boldsymbol{\phi}(\mathbf{x})$ denotes a fixed feature-space transformation, and we have made the bias parameter b explicit. Note that we shall shortly introduce a dual representation expressed in terms of kernel functions, which avoids having to work explicitly in feature space. The training data set comprises N input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$, with corresponding target values $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$, and new data points $\mathbf{x}$ are classified according to the sign of $y(\mathbf{x})$.

We shall assume for the moment that the training data set is linearly separable in feature space, so that by definition there exists at least one choice of the parameters $\mathbf{w}$ and b such that a function of the form (7.1) satisfies $y(\mathbf{x}_n) > 0$ for points having $t_n = +1$ and $y(\mathbf{x}_n) < 0$ for points having $t_n = -1$, so that $t_n y(\mathbf{x}_n) > 0$ for all training data points.

There may of course exist many such solutions that separate the classes exactly. In Section 4.1.7, we described the perceptron algorithm that is guaranteed to find a solution in a finite number of steps. The solution that it finds, however, will be dependent on the (arbitrary) initial values chosen for $\mathbf{w}$ and b as well as on the order in which the data points are presented. If there are multiple solutions all of which classify the training data set exactly, then we should try to find the one that will give the smallest generalization error. The support vector machine approaches this problem through the concept of the *margin*, which is defined to be the smallest distance between the decision boundary and any of the samples, as illustrated in Figure 7.1.

In support vector machines the decision boundary is chosen to be the one for which the margin is maximized. The maximum margin solution can be motivated using *computational learning theory*, also known as *statistical learning theory*. However, a simple insight into the origins of maximum margin has been given by Tong and Koller (2000) who consider a framework for classification based on a hybrid of generative and discriminative approaches. They first model the distribution over input vectors $\mathbf{x}$ for each class using a Parzen density estimator with Gaussian kernels

Section 7.1.5

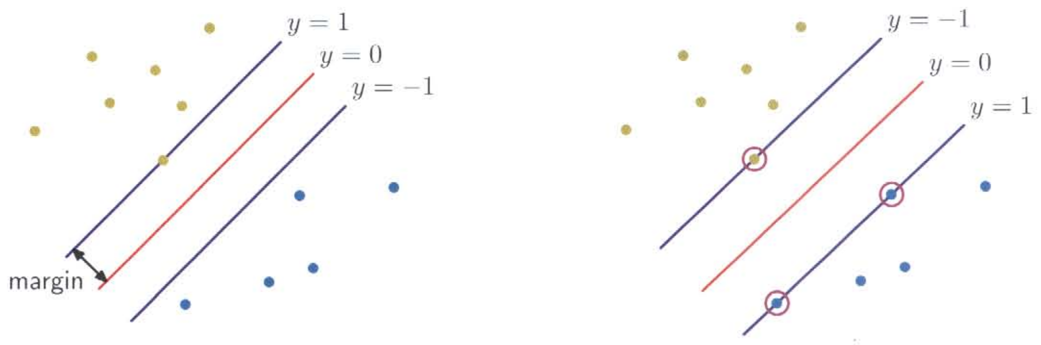

Figure 7.1 The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles.

having a common parameter σ^2. Together with the class priors, this defines an optimal misclassification-rate decision boundary. However, instead of using this optimal boundary, they determine the best hyperplane by minimizing the probability of error relative to the learned density model. In the limit $\sigma^2 \to 0$, the optimal hyperplane is shown to be the one having maximum margin. The intuition behind this result is that as σ^2 is reduced, the hyperplane is increasingly dominated by nearby data points relative to more distant ones. In the limit, the hyperplane becomes independent of data points that are not support vectors.

We shall see in Figure 10.13 that marginalization with respect to the prior distribution of the parameters in a Bayesian approach for a simple linearly separable data set leads to a decision boundary that lies in the middle of the region separating the data points. The large margin solution has similar behaviour.

Recall from Figure 4.1 that the perpendicular distance of a point $\mathbf{x}$ from a hyperplane defined by $y(\mathbf{x}) = 0$ where $y(\mathbf{x})$ takes the form (7.1) is given by $|y(\mathbf{x})|/\|\mathbf{w}\|$. Furthermore, we are only interested in solutions for which all data points are correctly classified, so that $t_n y(\mathbf{x}_n) > 0$ for all n. Thus the distance of a point $\mathbf{x}_n$ to the decision surface is given by

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}. \tag{7.2}$$

The margin is given by the perpendicular distance to the closest point $\mathbf{x}_n$ from the data set, and we wish to optimize the parameters $\mathbf{w}$ and b in order to maximize this distance. Thus the maximum margin solution is found by solving

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n \left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b \right) \right] \right\} \tag{7.3}$$

where we have taken the factor $1/\|\mathbf{w}\|$ outside the optimization over n because $\mathbf{w}$

does not depend on n. Direct solution of this optimization problem would be very complex, and so we shall convert it into an equivalent problem that is much easier to solve. To do this we note that if we make the rescaling $\mathbf{w} \rightarrow \kappa\mathbf{w}$ and $b \rightarrow \kappa b$, then the distance from any point $\mathbf{x}_n$ to the decision surface, given by $t_n y(\mathbf{x}_n)/\|\mathbf{w}\|$, is unchanged. We can use this freedom to set

$$t_n \left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b\right) = 1 \tag{7.4}$$

for the point that is closest to the surface. In this case, all data points will satisfy the constraints

$$t_n \left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b\right) \geqslant 1, \qquad n = 1, \ldots, N. \tag{7.5}$$

This is known as the canonical representation of the decision hyperplane. In the case of data points for which the equality holds, the constraints are said to be *active*, whereas for the remainder they are said to be *inactive*. By definition, there will always be at least one active constraint, because there will always be a closest point, and once the margin has been maximized there will be at least two active constraints. The optimization problem then simply requires that we maximize $\|\mathbf{w}\|^{-1}$, which is equivalent to minimizing $\|\mathbf{w}\|^2$, and so we have to solve the optimization problem

$$\underset{\mathbf{w},b}{\arg\min}\ \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.6}$$

subject to the constraints given by (7.5). The factor of $1/2$ in (7.6) is included for later convenience. This is an example of a *quadratic programming* problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints. It appears that the bias parameter b has disappeared from the optimization. However, it is determined implicitly via the constraints, because these require that changes to $\|\mathbf{w}\|$ be compensated by changes to b. We shall see how this works shortly.

Appendix E

In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geqslant 0$, with one multiplier a_n for each of the constraints in (7.5), giving the Lagrangian function

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b) - 1\right\} \tag{7.7}$$

where $\mathbf{a} = (a_1, \ldots, a_N)^{\mathrm{T}}$. Note the minus sign in front of the Lagrange multiplier term, because we are minimizing with respect to $\mathbf{w}$ and b, and maximizing with respect to $\mathbf{a}$. Setting the derivatives of $L(\mathbf{w}, b, \mathbf{a})$ with respect to $\mathbf{w}$ and b equal to zero, we obtain the following two conditions

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.8}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{7.9}$$

Eliminating $\mathbf{w}$ and b from $L(\mathbf{w}, b, \mathbf{a})$ using these conditions then gives the *dual representation* of the maximum margin problem in which we maximize

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \qquad (7.10)$$

with respect to $\mathbf{a}$ subject to the constraints

$$a_n \geqslant 0, \qquad n = 1, \ldots, N, \qquad (7.11)$$

$$\sum_{n=1}^{N} a_n t_n = 0. \qquad (7.12)$$

Here the kernel function is defined by $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$. Again, this takes the form of a quadratic programming problem in which we optimize a quadratic function of $\mathbf{a}$ subject to a set of inequality constraints. We shall discuss techniques for solving such quadratic programming problems in Section 7.1.1.

The solution to a quadratic programming problem in M variables in general has computational complexity that is $O(M^3)$. In going to the dual formulation we have turned the original optimization problem, which involved minimizing (7.6) over M variables, into the dual problem (7.10), which has N variables. For a fixed set of basis functions whose number M is smaller than the number N of data points, the move to the dual problem appears disadvantageous. However, it allows the model to be reformulated using kernels, and so the maximum margin classifier can be applied efficiently to feature spaces whose dimensionality exceeds the number of data points, including infinite feature spaces. The kernel formulation also makes clear the role of the constraint that the kernel function $k(\mathbf{x}, \mathbf{x}')$ be positive definite, because this ensures that the Lagrangian function $\widetilde{L}(\mathbf{a})$ is bounded below, giving rise to a well-defined optimization problem.

In order to classify new data points using the trained model, we evaluate the sign of $y(\mathbf{x})$ defined by (7.1). This can be expressed in terms of the parameters $\{a_n\}$ and the kernel function by substituting for $\mathbf{w}$ using (7.8) to give

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \qquad (7.13)$$

Joseph-Louis Lagrange
1736–1813

Although widely considered to be a French mathematician, Lagrange was born in Turin in Italy. By the age of nineteen, he had already made important contributions mathematics and had been appointed as Professor at the Royal Artillery School in Turin. For many years, Euler worked hard to persuade Lagrange to move to Berlin, which he eventually did in 1766 where he succeeded Euler as Director of Mathematics at the Berlin Academy. Later he moved to Paris, narrowly escaping with his life during the French revolution thanks to the personal intervention of Lavoisier (the French chemist who discovered oxygen) who himself was later executed at the guillotine. Lagrange made key contributions to the calculus of variations and the foundations of dynamics.

In Appendix E, we show that a constrained optimization of this form satisfies the *Karush-Kuhn-Tucker* (KKT) conditions, which in this case require that the following three properties hold

$$a_n \geqslant 0 \tag{7.14}$$

$$t_n y(\mathbf{x}_n) - 1 \geqslant 0 \tag{7.15}$$

$$a_n \{t_n y(\mathbf{x}_n) - 1\} = 0. \tag{7.16}$$

Thus for every data point, either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$. Any data point for which $a_n = 0$ will not appear in the sum in (7.13) and hence plays no role in making predictions for new data points. The remaining data points are called *support vectors*, and because they satisfy $t_n y(\mathbf{x}_n) = 1$, they correspond to points that lie on the maximum margin hyperplanes in feature space, as illustrated in Figure 7.1. This property is central to the practical applicability of support vector machines. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

Having solved the quadratic programming problem and found a value for $\mathbf{a}$, we can then determine the value of the threshold parameter b by noting that any support vector $\mathbf{x}_n$ satisfies $t_n y(\mathbf{x}_n) = 1$. Using (7.13) this gives

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1 \tag{7.17}$$

where $\mathcal{S}$ denotes the set of indices of the support vectors. Although we can solve this equation for b using an arbitrarily chosen support vector $\mathbf{x}_n$, a numerically more stable solution is obtained by first multiplying through by t_n, making use of $t_n^2 = 1$, and then averaging these equations over all support vectors and solving for b to give

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right) \tag{7.18}$$

where $N_{\mathcal{S}}$ is the total number of support vectors.

For later comparison with alternative models, we can express the maximum-margin classifier in terms of the minimization of an error function, with a simple quadratic regularizer, in the form

$$\sum_{n=1}^{N} E_\infty(y(\mathbf{x}_n)t_n - 1) + \lambda \|\mathbf{w}\|^2 \tag{7.19}$$

where $E_\infty(z)$ is a function that is zero if $z \geqslant 0$ and ∞ otherwise and ensures that the constraints (7.5) are satisfied. Note that as long as the regularization parameter satisfies $\lambda > 0$, its precise value plays no role.

Figure 7.2 shows an example of the classification resulting from training a support vector machine on a simple synthetic data set using a Gaussian kernel of the

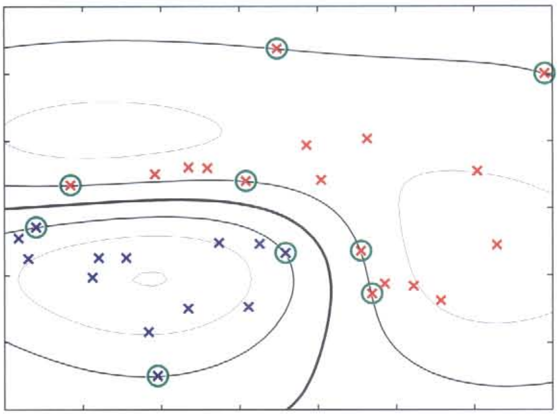

Figure 7.2 Example of synthetic data from two classes in two dimensions showing contours of constant $y(\mathbf{x})$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

form (6.23). Although the data set is not linearly separable in the two-dimensional data space $\mathbf{x}$, it is linearly separable in the nonlinear feature space defined implicitly by the nonlinear kernel function. Thus the training data points are perfectly separated in the original data space.

This example also provides a geometrical insight into the origin of sparsity in the SVM. The maximum margin hyperplane is defined by the location of the support vectors. Other data points can be moved around freely (so long as they remain outside the margin region) without changing the decision boundary, and so the solution will be independent of such data points.

7.1.1 Overlapping class distributions

So far, we have assumed that the training data points are linearly separable in the feature space $\phi(\mathbf{x})$. The resulting support vector machine will give exact separation of the training data in the original input space $\mathbf{x}$, although the corresponding decision boundary will be nonlinear. In practice, however, the class-conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization.

We therefore need a way to modify the support vector machine so as to allow some of the training points to be misclassified. From (7.19) we see that in the case of separable classes, we implicitly used an error function that gave infinite error if a data point was misclassified and zero error if it was classified correctly, and then optimized the model parameters to maximize the margin. We now modify this approach so that data points are allowed to be on the 'wrong side' of the margin boundary, but with a penalty that increases with the distance from that boundary. For the subsequent optimization problem, it is convenient to make this penalty a linear function of this distance. To do this, we introduce *slack variables*, $\xi_n \geqslant 0$ where $n = 1, \ldots, N$, with one slack variable for each training data point (Bennett, 1992; Cortes and Vapnik, 1995). These are defined by $\xi_n = 0$ for data points that are on or inside the correct margin boundary and $\xi_n = |t_n - y(\mathbf{x}_n)|$ for other points. Thus a data point that is on the decision boundary $y(\mathbf{x}_n) = 0$ will have $\xi_n = 1$, and points

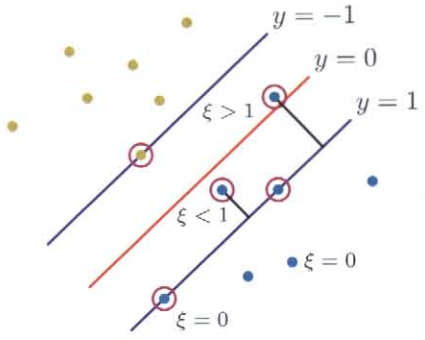

Figure 7.3 Illustration of the slack variables $\xi_n \geqslant 0$. Data points with circles around them are support vectors.

with $\xi_n > 1$ will be misclassified. The exact classification constraints (7.5) are then replaced with

$$t_n y(\mathbf{x}_n) \geqslant 1 - \xi_n, \qquad n = 1, \ldots, N \tag{7.20}$$

in which the slack variables are constrained to satisfy $\xi_n \geqslant 0$. Data points for which $\xi_n = 0$ are correctly classified and are either on the margin or on the correct side of the margin. Points for which $0 < \xi_n \leqslant 1$ lie inside the margin, but on the correct side of the decision boundary, and those data points for which $\xi_n > 1$ lie on the wrong side of the decision boundary and are misclassified, as illustrated in Figure 7.3. This is sometimes described as relaxing the hard margin constraint to give a *soft margin* and allows some of the training set data points to be misclassified. Note that while slack variables allow for overlapping class distributions, this framework is still sensitive to outliers because the penalty for misclassification increases linearly with ξ.

Our goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. We therefore minimize

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.21}$$

where the parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin. Because any point that is misclassified has $\xi_n > 1$, it follows that $\sum_n \xi_n$ is an upper bound on the number of misclassified points. The parameter C is therefore analogous to (the inverse of) a regularization coefficient because it controls the trade-off between minimizing training errors and controlling model complexity. In the limit $C \to \infty$, we will recover the earlier support vector machine for separable data.

We now wish to minimize (7.21) subject to the constraints (7.20) together with $\xi_n \geqslant 0$. The corresponding Lagrangian is given by

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n \{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n \tag{7.22}$$

where $\{a_n \geqslant 0\}$ and $\{\mu_n \geqslant 0\}$ are Lagrange multipliers. The corresponding set of KKT conditions are given by

Appendix E

$$\begin{aligned} a_n &\geqslant 0 && (7.23)\\ t_n y(\mathbf{x}_n) - 1 + \xi_n &\geqslant 0 && (7.24)\\ a_n\left(t_n y(\mathbf{x}_n) - 1 + \xi_n\right) &= 0 && (7.25)\\ \mu_n &\geqslant 0 && (7.26)\\ \xi_n &\geqslant 0 && (7.27)\\ \mu_n \xi_n &= 0 && (7.28)\end{aligned}$$

where $n = 1, \ldots, N$.

We now optimize out $\mathbf{w}$, b, and $\{\xi_n\}$ making use of the definition (7.1) of $y(\mathbf{x})$ to give

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.29}$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} a_n t_n = 0 \tag{7.30}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \quad \Rightarrow \quad a_n = C - \mu_n. \tag{7.31}$$

Using these results to eliminate $\mathbf{w}$, b, and $\{\xi_n\}$ from the Lagrangian, we obtain the dual Lagrangian in the form

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.32}$$

which is identical to the separable case, except that the constraints are somewhat different. To see what these constraints are, we note that $a_n \geqslant 0$ is required because these are Lagrange multipliers. Furthermore, (7.31) together with $\mu_n \geqslant 0$ implies $a_n \leqslant C$. We therefore have to minimize (7.32) with respect to the dual variables $\{a_n\}$ subject to

$$0 \leqslant a_n \leqslant C \tag{7.33}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{7.34}$$

for $n = 1, \ldots, N$, where (7.33) are known as *box constraints*. This again represents a quadratic programming problem. If we substitute (7.29) into (7.1), we see that predictions for new data points are again made by using (7.13).

We can now interpret the resulting solution. As before, a subset of the data points may have $a_n = 0$, in which case they do not contribute to the predictive model (7.13). The remaining data points constitute the support vectors. These have $a_n > 0$ and hence from (7.25) must satisfy

$$t_n y(\mathbf{x}_n) = 1 - \xi_n. \tag{7.35}$$

If $a_n < C$, then (7.31) implies that $\mu_n > 0$, which from (7.28) requires $\xi_n = 0$ and hence such points lie on the margin. Points with $a_n = C$ can lie inside the margin and can either be correctly classified if $\xi_n \leqslant 1$ or misclassified if $\xi_n > 1$.

To determine the parameter b in (7.1), we note that those support vectors for which $0 < a_n < C$ have $\xi_n = 0$ so that $t_n y(\mathbf{x}_n) = 1$ and hence will satisfy

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1. \tag{7.36}$$

Again, a numerically stable solution is obtained by averaging to give

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right) \tag{7.37}$$

where $\mathcal{M}$ denotes the set of indices of data points having $0 < a_n < C$.

An alternative, equivalent formulation of the support vector machine, known as the *ν-SVM*, has been proposed by Schölkopf *et al.* (2000). This involves maximizing

$$\widetilde{L}(\mathbf{a}) = -\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.38}$$

subject to the constraints

$$0 \leqslant a_n \leqslant 1/N \tag{7.39}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{7.40}$$

$$\sum_{n=1}^{N} a_n \geqslant \nu. \tag{7.41}$$

This approach has the advantage that the parameter ν, which replaces C, can be interpreted as both an upper bound on the fraction of *margin errors* (points for which $\xi_n > 0$ and hence which lie on the wrong side of the margin boundary and which may or may not be misclassified) and a lower bound on the fraction of support vectors. An example of the ν-SVM applied to a synthetic data set is shown in Figure 7.4. Here Gaussian kernels of the form $\exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$ have been used, with $\gamma = 0.45$.

Although predictions for new inputs are made using only the support vectors, the training phase (i.e., the determination of the parameters $\mathbf{a}$ and b) makes use of the whole data set, and so it is important to have efficient algorithms for solving

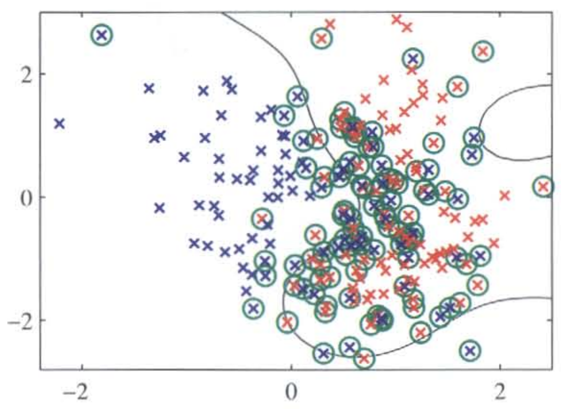

Figure 7.4 Illustration of the ν-SVM applied to a nonseparable data set in two dimensions. The support vectors are indicated by circles.

the quadratic programming problem. We first note that the objective function $\widetilde{L}(\mathbf{a})$ given by (7.10) or (7.32) is quadratic and so any local optimum will also be a global optimum provided the constraints define a convex region (which they do as a consequence of being linear). Direct solution of the quadratic programming problem using traditional techniques is often infeasible due to the demanding computation and memory requirements, and so more practical approaches need to be found. The technique of *chunking* (Vapnik, 1982) exploits the fact that the value of the Lagrangian is unchanged if we remove the rows and columns of the kernel matrix corresponding to Lagrange multipliers that have value zero. This allows the full quadratic programming problem to be broken down into a series of smaller ones, whose goal is eventually to identify all of the nonzero Lagrange multipliers and discard the others. Chunking can be implemented using *protected conjugate gradients* (Burges, 1998). Although chunking reduces the size of the matrix in the quadratic function from the number of data points squared to approximately the number of nonzero Lagrange multipliers squared, even this may be too big to fit in memory for large-scale applications. *Decomposition methods* (Osuna *et al.*, 1996) also solve a series of smaller quadratic programming problems but are designed so that each of these is of a fixed size, and so the technique can be applied to arbitrarily large data sets. However, it still involves numerical solution of quadratic programming subproblems and these can be problematic and expensive. One of the most popular approaches to training support vector machines is called *sequential minimal optimization*, or *SMO* (Platt, 1999). It takes the concept of chunking to the extreme limit and considers just two Lagrange multipliers at a time. In this case, the subproblem can be solved analytically, thereby avoiding numerical quadratic programming altogether. Heuristics are given for choosing the pair of Lagrange multipliers to be considered at each step. In practice, SMO is found to have a scaling with the number of data points that is somewhere between linear and quadratic depending on the particular application.

We have seen that kernel functions correspond to inner products in feature spaces that can have high, or even infinite, dimensionality. By working directly in terms of the kernel function, without introducing the feature space explicitly, it might therefore seem that support vector machines somehow manage to avoid the curse of di-

Section 1.4

mensionality. This is not the case, however, because there are constraints amongst the feature values that restrict the effective dimensionality of feature space. To see this consider a simple second-order polynomial kernel that we can expand in terms of its components

$$\begin{aligned} k(\mathbf{x},\mathbf{z}) &= \left(1+\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2 = (1+x_1z_1+x_2z_2)^2 \\ &= 1+2x_1z_1+2x_2z_2+x_1^2z_1^2+2x_1z_1x_2z_2+x_2^2z_2^2 \\ &= (1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2)(1,\sqrt{2}z_1,\sqrt{2}z_2,z_1^2,\sqrt{2}z_1z_2,z_2^2)^{\mathrm{T}} \\ &= \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{z}). \end{aligned} \tag{7.42}$$

This kernel function therefore represents an inner product in a feature space having six dimensions, in which the mapping from input space to feature space is described by the vector function $\boldsymbol{\phi}(\mathbf{x})$. However, the coefficients weighting these different features are constrained to have specific forms. Thus any set of points in the original two-dimensional space $\mathbf{x}$ would be constrained to lie exactly on a two-dimensional nonlinear manifold embedded in the six-dimensional feature space.

We have already highlighted the fact that the support vector machine does not provide probabilistic outputs but instead makes classification decisions for new input vectors. Veropoulos *et al.* (1999) discuss modifications to the SVM to allow the trade-off between false positive and false negative errors to be controlled. However, if we wish to use the SVM as a module in a larger probabilistic system, then probabilistic predictions of the class label t for new inputs $\mathbf{x}$ are required.

To address this issue, Platt (2000) has proposed fitting a logistic sigmoid to the outputs of a previously trained support vector machine. Specifically, the required conditional probability is assumed to be of the form

$$p(t=1|\mathbf{x}) = \sigma\left(Ay(\mathbf{x})+B\right) \tag{7.43}$$

where $y(\mathbf{x})$ is defined by (7.1). Values for the parameters A and B are found by minimizing the cross-entropy error function defined by a training set consisting of pairs of values $y(\mathbf{x}_n)$ and t_n. The data used to fit the sigmoid needs to be independent of that used to train the original SVM in order to avoid severe over-fitting. This two-stage approach is equivalent to assuming that the output $y(\mathbf{x})$ of the support vector machine represents the log-odds of $\mathbf{x}$ belonging to class $t=1$. Because the SVM training procedure is not specifically intended to encourage this, the SVM can give a poor approximation to the posterior probabilities (Tipping, 2001).

7.1.2 Relation to logistic regression

As with the separable case, we can re-cast the SVM for nonseparable distributions in terms of the minimization of a regularized error function. This will also allow us to highlight similarities, and differences, compared to the logistic regression model.

Section 4.3.2

We have seen that for data points that are on the correct side of the margin boundary, and which therefore satisfy $y_nt_n \geqslant 1$, we have $\xi_n = 0$, and for the

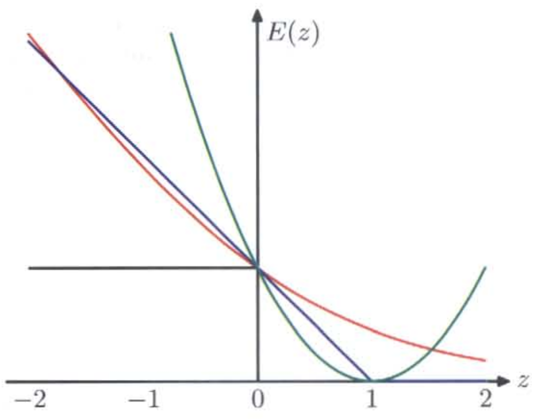

Figure 7.5 Plot of the 'hinge' error function used in support vector machines, shown in blue, along with the error function for logistic regression, rescaled by a factor of $1/\ln(2)$ so that it passes through the point $(0,1)$, shown in red. Also shown are the misclassification error in black and the squared error in green.

remaining points we have $\xi_n = 1 - y_n t_n$. Thus the objective function (7.21) can be written (up to an overall multiplicative constant) in the form

$$\sum_{n=1}^{N} E_{\mathrm{SV}}(y_n t_n) + \lambda \|\mathbf{w}\|^2 \tag{7.44}$$

where $\lambda = (2C)^{-1}$, and $E_{\mathrm{SV}}(\cdot)$ is the *hinge* error function defined by

$$E_{\mathrm{SV}}(y_n t_n) = [1 - y_n t_n]_+ \tag{7.45}$$

where $[\cdot]_+$ denotes the positive part. The hinge error function, so-called because of its shape, is plotted in Figure 7.5. It can be viewed as an approximation to the misclassification error, i.e., the error function that ideally we would like to minimize, which is also shown in Figure 7.5.

When we considered the logistic regression model in Section 4.3.2, we found it convenient to work with target variable $t \in \{0, 1\}$. For comparison with the support vector machine, we first reformulate maximum likelihood logistic regression using the target variable $t \in \{-1, 1\}$. To do this, we note that $p(t = 1|y) = \sigma(y)$ where $y(\mathbf{x})$ is given by (7.1), and $\sigma(y)$ is the logistic sigmoid function defined by (4.59). It follows that $p(t = -1|y) = 1 - \sigma(y) = \sigma(-y)$, where we have used the properties of the logistic sigmoid function, and so we can write

$$p(t|y) = \sigma(yt). \tag{7.46}$$

Exercise 7.6

From this we can construct an error function by taking the negative logarithm of the likelihood function that, with a quadratic regularizer, takes the form

$$\sum_{n=1}^{N} E_{\mathrm{LR}}(y_n t_n) + \lambda \|\mathbf{w}\|^2. \tag{7.47}$$

where

$$E_{\mathrm{LR}}(yt) = \ln\left(1 + \exp(-yt)\right). \tag{7.48}$$

For comparison with other error functions, we can divide by $\ln(2)$ so that the error function passes through the point $(0, 1)$. This rescaled error function is also plotted in Figure 7.5 and we see that it has a similar form to the support vector error function. The key difference is that the flat region in $E_{\text{SV}}(yt)$ leads to sparse solutions.

Both the logistic error and the hinge loss can be viewed as continuous approximations to the misclassification error. Another continuous error function that has sometimes been used to solve classification problems is the squared error, which is again plotted in Figure 7.5. It has the property, however, of placing increasing emphasis on data points that are correctly classified but that are a long way from the decision boundary on the correct side. Such points will be strongly weighted at the expense of misclassified points, and so if the objective is to minimize the misclassification rate, then a monotonically decreasing error function would be a better choice.

7.1.3 Multiclass SVMs

The support vector machine is fundamentally a two-class classifier. In practice, however, we often have to tackle problems involving $K > 2$ classes. Various methods have therefore been proposed for combining multiple two-class SVMs in order to build a multiclass classifier.

One commonly used approach (Vapnik, 1998) is to construct K separate SVMs, in which the k^{th} model $y_k(\mathbf{x})$ is trained using the data from class $\mathcal{C}_k$ as the positive examples and the data from the remaining $K - 1$ classes as the negative examples. This is known as the *one-versus-the-rest* approach. However, in Figure 4.2 we saw that using the decisions of the individual classifiers can lead to inconsistent results in which an input is assigned to multiple classes simultaneously. This problem is sometimes addressed by making predictions for new inputs $\mathbf{x}$ using

$$y(\mathbf{x}) = \max_k y_k(\mathbf{x}). \tag{7.49}$$

Unfortunately, this heuristic approach suffers from the problem that the different classifiers were trained on different tasks, and there is no guarantee that the real-valued quantities $y_k(\mathbf{x})$ for different classifiers will have appropriate scales.

Another problem with the one-versus-the-rest approach is that the training sets are imbalanced. For instance, if we have ten classes each with equal numbers of training data points, then the individual classifiers are trained on data sets comprising 90% negative examples and only 10% positive examples, and the symmetry of the original problem is lost. A variant of the one-versus-the-rest scheme was proposed by Lee *et al.* (2001) who modify the target values so that the positive class has target $+1$ and the negative class has target $-1/(K-1)$.

Weston and Watkins (1999) define a single objective function for training all K SVMs simultaneously, based on maximizing the margin from each to remaining classes. However, this can result in much slower training because, instead of solving K separate optimization problems each over N data points with an overall cost of $O(KN^2)$, a single optimization problem of size $(K-1)N$ must be solved giving an overall cost of $O(K^2N^2)$.

Another approach is to train $K(K-1)/2$ different 2-class SVMs on all possible pairs of classes, and then to classify test points according to which class has the highest number of 'votes', an approach that is sometimes called *one-versus-one*. Again, we saw in Figure 4.2 that this can lead to ambiguities in the resulting classification. Also, for large K this approach requires significantly more training time than the one-versus-the-rest approach. Similarly, to evaluate test points, significantly more computation is required.

The latter problem can be alleviated by organizing the pairwise classifiers into a directed acyclic graph (not to be confused with a probabilistic graphical model) leading to the *DAGSVM* (Platt *et al.*, 2000). For K classes, the DAGSVM has a total of $K(K-1)/2$ classifiers, and to classify a new test point only $K-1$ pairwise classifiers need to be evaluated, with the particular classifiers used depending on which path through the graph is traversed.

A different approach to multiclass classification, based on error-correcting output codes, was developed by Dietterich and Bakiri (1995) and applied to support vector machines by Allwein *et al.* (2000). This can be viewed as a generalization of the voting scheme of the one-versus-one approach in which more general partitions of the classes are used to train the individual classifiers. The K classes themselves are represented as particular sets of responses from the two-class classifiers chosen, and together with a suitable decoding scheme, this gives robustness to errors and to ambiguity in the outputs of the individual classifiers. Although the application of SVMs to multiclass classification problems remains an open issue, in practice the one-versus-the-rest approach is the most widely used in spite of its ad-hoc formulation and its practical limitations.

There are also *single-class* support vector machines, which solve an unsupervised learning problem related to probability density estimation. Instead of modelling the density of data, however, these methods aim to find a smooth boundary enclosing a region of high density. The boundary is chosen to represent a quantile of the density, that is, the probability that a data point drawn from the distribution will land inside that region is given by a fixed number between 0 and 1 that is specified in advance. This is a more restricted problem than estimating the full density but may be sufficient in specific applications. Two approaches to this problem using support vector machines have been proposed. The algorithm of Schölkopf *et al.* (2001) tries to find a hyperplane that separates all but a fixed fraction ν of the training data from the origin while at the same time maximizing the distance (margin) of the hyperplane from the origin, while Tax and Duin (1999) look for the smallest sphere in feature space that contains all but a fraction ν of the data points. For kernels $k(\mathbf{x}, \mathbf{x}')$ that are functions only of $\mathbf{x} - \mathbf{x}'$, the two algorithms are equivalent.

7.1.4 SVMs for regression

Section 3.1.4

We now extend support vector machines to regression problems while at the same time preserving the property of sparseness. In simple linear regression, we

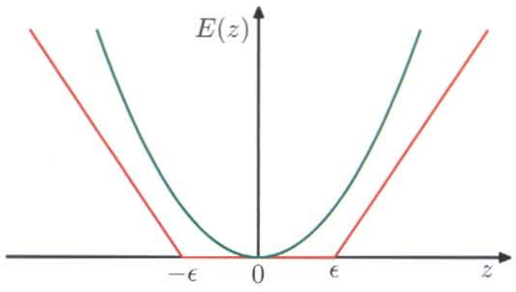

Figure 7.6 Plot of an ϵ-insensitive error function (in red) in which the error increases linearly with distance beyond the insensitive region. Also shown for comparison is the quadratic error function (in green).

minimize a regularized error function given by

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2. \tag{7.50}$$

To obtain sparse solutions, the quadratic error function is replaced by an *ϵ-insensitive error function* (Vapnik, 1995), which gives zero error if the absolute difference between the prediction $y(\mathbf{x})$ and the target t is less than ϵ where $\epsilon > 0$. A simple example of an ϵ-insensitive error function, having a linear cost associated with errors outside the insensitive region, is given by

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0, & \text{if } |y(\mathbf{x}) - t| < \epsilon; \\ |y(\mathbf{x}) - t| - \epsilon, & \text{otherwise} \end{cases} \tag{7.51}$$

and is illustrated in Figure 7.6.

We therefore minimize a regularized error function given by

$$C\sum_{n=1}^{N} E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.52}$$

where $y(\mathbf{x})$ is given by (7.1). By convention the (inverse) regularization parameter, denoted C, appears in front of the error term.

As before, we can re-express the optimization problem by introducing slack variables. For each data point $\mathbf{x}_n$, we now need two slack variables $\xi_n \geqslant 0$ and $\widehat{\xi}_n \geqslant 0$, where $\xi_n > 0$ corresponds to a point for which $t_n > y(\mathbf{x}_n) + \epsilon$, and $\widehat{\xi}_n > 0$ corresponds to a point for which $t_n < y(\mathbf{x}_n) - \epsilon$, as illustrated in Figure 7.7.

The condition for a target point to lie inside the ϵ-tube is that $y_n - \epsilon \leqslant t_n \leqslant y_n + \epsilon$, where $y_n = y(\mathbf{x}_n)$. Introducing the slack variables allows points to lie outside the tube provided the slack variables are nonzero, and the corresponding conditions are

$$t_n \leqslant y(\mathbf{x}_n) + \epsilon + \xi_n \tag{7.53}$$

$$t_n \geqslant y(\mathbf{x}_n) - \epsilon - \widehat{\xi}_n. \tag{7.54}$$

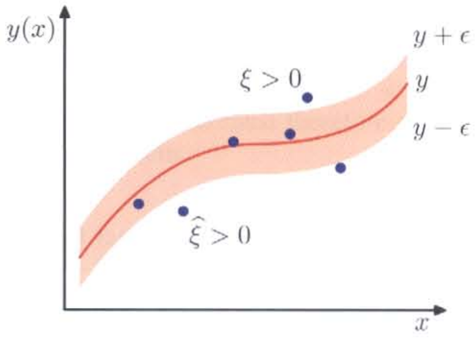

Figure 7.7 Illustration of SVM regression, showing the regression curve together with the ϵ-insensitive 'tube'. Also shown are examples of the slack variables ξ and $\widehat{\xi}$. Points above the ϵ-tube have $\xi > 0$ and $\widehat{\xi} = 0$, points below the ϵ-tube have $\xi = 0$ and $\widehat{\xi} > 0$, and points inside the ϵ-tube have $\xi = \widehat{\xi} = 0$.

The error function for support vector regression can then be written as

$$C\sum_{n=1}^{N}(\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.55}$$

which must be minimized subject to the constraints $\xi_n \geqslant 0$ and $\widehat{\xi}_n \geqslant 0$ as well as (7.53) and (7.54). This can be achieved by introducing Lagrange multipliers $a_n \geqslant 0$, $\widehat{a}_n \geqslant 0$, $\mu_n \geqslant 0$, and $\widehat{\mu}_n \geqslant 0$ and optimizing the Lagrangian

$$\begin{aligned} L &= C\sum_{n=1}^{N}(\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}(\mu_n\xi_n + \widehat{\mu}_n\widehat{\xi}_n) \\ &\quad - \sum_{n=1}^{N} a_n(\epsilon + \xi_n + y_n - t_n) - \sum_{n=1}^{N}\widehat{a}_n(\epsilon + \widehat{\xi}_n - y_n + t_n). \end{aligned} \tag{7.56}$$

We now substitute for $y(\mathbf{x})$ using (7.1) and then set the derivatives of the Lagrangian with respect to $\mathbf{w}$, b, ξ_n, and $\widehat{\xi}_n$ to zero, giving

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad\Rightarrow\quad \mathbf{w} = \sum_{n=1}^{N}(a_n - \widehat{a}_n)\boldsymbol{\phi}(\mathbf{x}_n) \tag{7.57}$$

$$\frac{\partial L}{\partial b} = 0 \quad\Rightarrow\quad \sum_{n=1}^{N}(a_n - \widehat{a}_n) = 0 \tag{7.58}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \quad\Rightarrow\quad a_n + \mu_n = C \tag{7.59}$$

$$\frac{\partial L}{\partial \widehat{\xi}_n} = 0 \quad\Rightarrow\quad \widehat{a}_n + \widehat{\mu}_n = C. \tag{7.60}$$

Exercise 7.7

Using these results to eliminate the corresponding variables from the Lagrangian, we see that the dual problem involves maximizing

$$\begin{aligned}\widetilde{L}(\mathbf{a}, \widehat{\mathbf{a}}) &= -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) \\ &\quad -\epsilon\sum_{n=1}^{N}(a_n + \widehat{a}_n) + \sum_{n=1}^{N}(a_n - \widehat{a}_n)t_n \end{aligned} \tag{7.61}$$

with respect to $\{a_n\}$ and $\{\widehat{a}_n\}$, where we have introduced the kernel $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}')$. Again, this is a constrained maximization, and to find the constraints we note that $a_n \geqslant 0$ and $\widehat{a}_n \geqslant 0$ are both required because these are Lagrange multipliers. Also $\mu_n \geqslant 0$ and $\widehat{\mu}_n \geqslant 0$ together with (7.59) and (7.60), require $a_n \leqslant C$ and $\widehat{a}_n \leqslant C$, and so again we have the box constraints

$$0 \leqslant a_n \leqslant C \tag{7.62}$$

$$0 \leqslant \widehat{a}_n \leqslant C \tag{7.63}$$

together with the condition (7.58).

Substituting (7.57) into (7.1), we see that predictions for new inputs can be made using

$$y(\mathbf{x}) = \sum_{n=1}^{N}(a_n - \widehat{a}_n)k(\mathbf{x}, \mathbf{x}_n) + b \tag{7.64}$$

which is again expressed in terms of the kernel function.

The corresponding Karush-Kuhn-Tucker (KKT) conditions, which state that at the solution the product of the dual variables and the constraints must vanish, are given by

$$a_n(\epsilon + \xi_n + y_n - t_n) = 0 \tag{7.65}$$

$$\widehat{a}_n(\epsilon + \widehat{\xi}_n - y_n + t_n) = 0 \tag{7.66}$$

$$(C - a_n)\xi_n = 0 \tag{7.67}$$

$$(C - \widehat{a}_n)\widehat{\xi}_n = 0. \tag{7.68}$$

From these we can obtain several useful results. First of all, we note that a coefficient a_n can only be nonzero if $\epsilon + \xi_n + y_n - t_n = 0$, which implies that the data point either lies on the upper boundary of the ϵ-tube ($\xi_n = 0$) or lies above the upper boundary ($\xi_n > 0$). Similarly, a nonzero value for $\widehat{a}_n$ implies $\epsilon + \widehat{\xi}_n - y_n + t_n = 0$, and such points must lie either on or below the lower boundary of the ϵ-tube.

Furthermore, the two constraints $\epsilon + \xi_n + y_n - t_n = 0$ and $\epsilon + \widehat{\xi}_n - y_n + t_n = 0$ are incompatible, as is easily seen by adding them together and noting that ξ_n and $\widehat{\xi}_n$ are nonnegative while ϵ is strictly positive, and so for every data point $\mathbf{x}_n$, either a_n or $\widehat{a}_n$ (or both) must be zero.

The support vectors are those data points that contribute to predictions given by (7.64), in other words those for which either $a_n \neq 0$ or $\widehat{a}_n \neq 0$. These are points that lie on the boundary of the ϵ-tube or outside the tube. All points within the tube have

$a_n = \widehat{a}_n = 0$. We again have a sparse solution, and the only terms that have to be evaluated in the predictive model (7.64) are those that involve the support vectors.

The parameter b can be found by considering a data point for which $0 < a_n < C$, which from (7.67) must have $\xi_n = 0$, and from (7.65) must therefore satisfy $\epsilon + y_n - t_n = 0$. Using (7.1) and solving for b, we obtain

$$\begin{aligned} b &= t_n - \epsilon - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) \\ &= t_n - \epsilon - \sum_{m=1}^{N}(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) \end{aligned} \tag{7.69}$$

where we have used (7.57). We can obtain an analogous result by considering a point for which $0 < \widehat{a}_n < C$. In practice, it is better to average over all such estimates of b.

As with the classification case, there is an alternative formulation of the SVM for regression in which the parameter governing complexity has a more intuitive interpretation (Schölkopf *et al.*, 2000). In particular, instead of fixing the width ϵ of the insensitive region, we fix instead a parameter ν that bounds the fraction of points lying outside the tube. This involves maximizing

$$\begin{aligned} \widetilde{L}(\mathbf{a}, \widehat{\mathbf{a}}) = & -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) \\ & + \sum_{n=1}^{N}(a_n - \widehat{a}_n)t_n \end{aligned} \tag{7.70}$$

subject to the constraints

$$0 \leqslant a_n \leqslant C/N \tag{7.71}$$

$$0 \leqslant \widehat{a}_n \leqslant C/N \tag{7.72}$$

$$\sum_{n=1}^{N}(a_n - \widehat{a}_n) = 0 \tag{7.73}$$

$$\sum_{n=1}^{N}(a_n + \widehat{a}_n) \leqslant \nu C. \tag{7.74}$$

It can be shown that there are at most νN data points falling outside the insensitive tube, while at least νN data points are support vectors and so lie either on the tube or outside it.

Appendix A

The use of a support vector machine to solve a regression problem is illustrated using the sinusoidal data set in Figure 7.8. Here the parameters ν and C have been chosen by hand. In practice, their values would typically be determined by cross-validation.

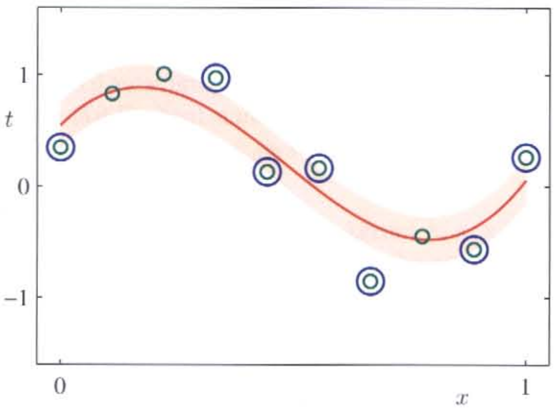

Figure 7.8 Illustration of the ν-SVM for regression applied to the sinusoidal synthetic data set using Gaussian kernels. The predicted regression curve is shown by the red line, and the ϵ-insensitive tube corresponds to the shaded region. Also, the data points are shown in green, and those with support vectors are indicated by blue circles.

7.1.5 Computational learning theory

Historically, support vector machines have largely been motivated and analysed using a theoretical framework known as *computational learning theory*, also sometimes called *statistical learning theory* (Anthony and Biggs, 1992; Kearns and Vazirani, 1994; Vapnik, 1995; Vapnik, 1998). This has its origins with Valiant (1984) who formulated the *probably approximately correct*, or PAC, learning framework. The goal of the PAC framework is to understand how large a data set needs to be in order to give good generalization. It also gives bounds for the computational cost of learning, although we do not consider these here.

Suppose that a data set $\mathcal{D}$ of size N is drawn from some joint distribution $p(\mathbf{x}, \mathbf{t})$ where $\mathbf{x}$ is the input variable and $\mathbf{t}$ represents the class label, and that we restrict attention to 'noise free' situations in which the class labels are determined by some (unknown) deterministic function $\mathbf{t} = \mathbf{g}(\mathbf{x})$. In PAC learning we say that a function $\mathbf{f}(\mathbf{x}; \mathcal{D})$, drawn from a space $\mathcal{F}$ of such functions on the basis of the training set $\mathcal{D}$, has good generalization if its expected error rate is below some pre-specified threshold ϵ, so that

$$\mathbb{E}_{\mathbf{x},\mathbf{t}}\left[I\left(\mathbf{f}(\mathbf{x}; \mathcal{D}) \neq \mathbf{t}\right)\right] < \epsilon \tag{7.75}$$

where $I(\cdot)$ is the indicator function, and the expectation is with respect to the distribution $p(\mathbf{x}, \mathbf{t})$. The quantity on the left-hand side is a random variable, because it depends on the training set $\mathcal{D}$, and the PAC framework requires that (7.75) holds, with probability greater than $1 - \delta$, for a data set $\mathcal{D}$ drawn randomly from $p(\mathbf{x}, \mathbf{t})$. Here δ is another pre-specified parameter, and the terminology 'probably approximately correct' comes from the requirement that with high probability (greater than $1 - \delta$), the error rate be small (less than ϵ). For a given choice of model space $\mathcal{F}$, and for given parameters ϵ and δ, PAC learning aims to provide bounds on the minimum size N of data set needed to meet this criterion. A key quantity in PAC learning is the *Vapnik-Chervonenkis dimension*, or VC dimension, which provides a measure of the complexity of a space of functions, and which allows the PAC framework to be extended to spaces containing an infinite number of functions.

The bounds derived within the PAC framework are often described as worstcase, because they apply to *any* choice for the distribution $p(\mathbf{x}, \mathbf{t})$, so long as both the training and the test examples are drawn (independently) from the same distribution, and for *any* choice for the function $\mathbf{f}(\mathbf{x})$ so long as it belongs to $\mathcal{F}$. In real-world applications of machine learning, we deal with distributions that have significant regularity, for example in which large regions of input space carry the same class label. As a consequence of the lack of any assumptions about the form of the distribution, the PAC bounds are very conservative, in other words they strongly over-estimate the size of data sets required to achieve a given generalization performance. For this reason, PAC bounds have found few, if any, practical applications.

One attempt to improve the tightness of the PAC bounds is the *PAC-Bayesian* framework (McAllester, 2003), which considers a distribution over the space $\mathcal{F}$ of functions, somewhat analogous to the prior in a Bayesian treatment. This still considers any possible choice for $p(\mathbf{x}, \mathbf{t})$, and so although the bounds are tighter, they are still very conservative.

7.2. Relevance Vector Machines

Support vector machines have been used in a variety of classification and regression applications. Nevertheless, they suffer from a number of limitations, several of which have been highlighted already in this chapter. In particular, the outputs of an SVM represent decisions rather than posterior probabilities. Also, the SVM was originally formulated for two classes, and the extension to $K > 2$ classes is problematic. There is a complexity parameter C, or ν (as well as a parameter ϵ in the case of regression), that must be found using a hold-out method such as cross-validation. Finally, predictions are expressed as linear combinations of kernel functions that are centred on training data points and that are required to be positive definite.

The *relevance vector machine* or RVM (Tipping, 2001) is a Bayesian sparse kernel technique for regression and classification that shares many of the characteristics of the SVM whilst avoiding its principal limitations. Additionally, it typically leads to much sparser models resulting in correspondingly faster performance on test data whilst maintaining comparable generalization error.

In contrast to the SVM we shall find it more convenient to introduce the regression form of the RVM first and then consider the extension to classification tasks.

7.2.1 RVM for regression

The relevance vector machine for regression is a linear model of the form studied in Chapter 3 but with a modified prior that results in sparse solutions. The model defines a conditional distribution for a real-valued target variable t, given an input vector $\mathbf{x}$, which takes the form

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}), \beta^{-1}) \tag{7.76}$$