and C_- of both classes of training points,

$$C_\pm := \left\{ \sum_{y_i=\pm 1} c_i \mathbf{x}_i \,\middle|\, \sum_{y_i=\pm 1} c_i = 1, c_i \geq 0 \right\}. \tag{7.21}$$

Convex Hull Separation

It can be shown that the maximum margin hyperplane as described above is the one bisecting the shortest line orthogonally connecting C_+ and C_- (Figure 7.5). Formally, this can be seen by considering the optimization problem

$$\begin{aligned} &\underset{\mathbf{c}\in\mathbb{R}^m}{\text{minimize}} \quad \left\| \sum_{y_i=1} c_i \mathbf{x}_i - \sum_{y_i=-1} c_i \mathbf{x}_i \right\|^2, \\ &\text{subject to} \quad \sum_{y_i=1} c_i = 1, \; \sum_{y_i=-1} c_i = 1, c_i \geq 0, \end{aligned} \tag{7.22}$$

and using the normal vector $\mathbf{w} = \sum_{y_i=1} c_i \mathbf{x}_i - \sum_{y_i=-1} c_i \mathbf{x}_i$, scaled to satisfy the canonicality condition (Definition 7.1). The threshold b is explicitly adjusted such that the hyperplane bisects the shortest connecting line (see also Problem 7.7).

7.4 Nonlinear Support Vector Classifiers

Thus far, we have shown why it is that a large margin hyperplane is good from a statistical point of view, and we have demonstrated how to compute it. Although these two points have worked out nicely, there is still a major drawback to the approach: Everything that we have done so far is linear in the data. To allow for much more general decision surfaces, we now use kernels to nonlinearly transform the input data $x_1, \ldots, x_m \in \mathcal{X}$ into a high-dimensional feature space, using a map $\Phi : x_i \mapsto \mathbf{x}_i$; we then do a linear separation there.

To justify this procedure, Cover's Theorem [113] is sometimes alluded to. This theorem characterizes the number of possible linear separations of m points in general position in an N-dimensional space. If $m \leq N+1$, then all 2^m separations are possible — the VC dimension of the function class is $n+1$ (Section 5.5.6). If $m > N+1$, then Cover's Theorem states that the number of linear separations equals

Cover's Theorem

$$2 \sum_{i=0}^{N} \binom{m-1}{i}. \tag{7.23}$$

The more we increase N, the more terms there are in the sum, and thus the larger is the resulting number. This theorem formalizes the intuition that the number of separations increases with the dimensionality. It requires, however, that the points are in general position — therefore, it does not strictly make a statement about the separability of a given dataset in a given feature space. E.g., the feature map might be such that all points lie on a rather restrictive lower-dimensional manifold, which could prevent us from finding points in general position.

There is another way to intuitively understand why the kernel mapping in-

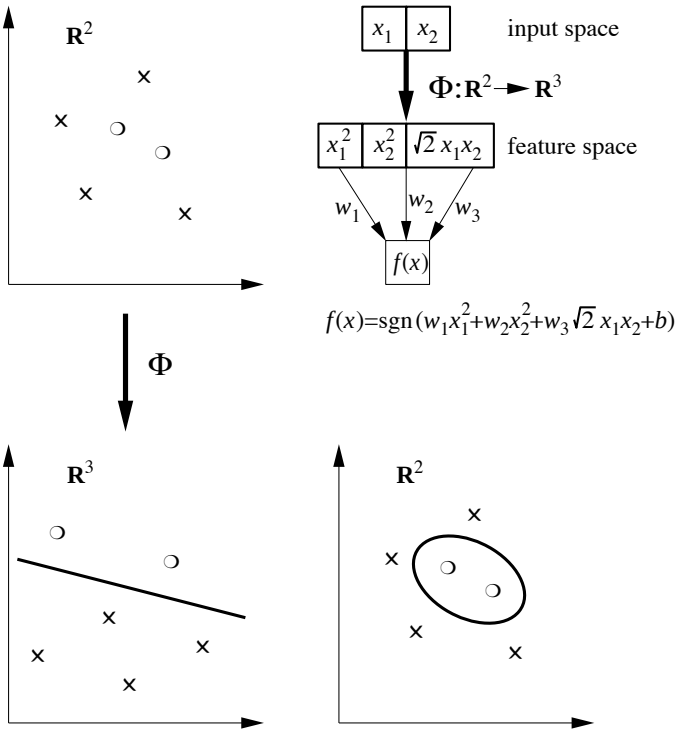

Figure 7.6 By mapping the input data *(top left)* nonlinearly (via Φ) into a higher-dimensional feature space $\mathcal{H}$ (here: $\mathcal{H} = \mathbb{R}^3$), and constructing a separating hyperplane there *(bottom left)*, an SVM *(top right)* corresponds to a nonlinear decision surface in input space (here: $\mathbb{R}^2$, *bottom right*). We use x_1, x_2 to denote the entries of the input vectors, and w_1, w_2, w_3 to denote the entries of the hyperplane normal vector in $\mathcal{H}$.

creases the chances of a separation, in terms of concepts of statistical learning theory. Using a kernel typically amounts to using a larger function class, thus increasing the capacity of the learning machine, and rendering problems separable that are not linearly separable to start with.

"Kernelizing" the Optimal Margin Hyperplane

On the practical level, the modification necessary to perform the algorithm in a high-dimensional feature space are minor. In the above sections, we made no assumptions on the dimensionality of $\mathcal{H}$, the space in which we assumed our patterns belong. We only required $\mathcal{H}$ to be equipped with a dot product. The patterns $\mathbf{x}_i$ that we talked about previously thus need not coincide with the input patterns. They can equally well be the results of mapping the original input patterns x_i into a high-dimensional feature space. Consequently, we take the stance that wherever we wrote $\mathbf{x}$, we actually meant $\Phi(x)$. Maximizing the target function (7.17), and evaluating the decision function (7.20), then requires the computation of dot products $\langle \Phi(x), \Phi(x_i) \rangle$ in a high-dimensional space. These expensive calculations are reduced significantly by using a positive definite kernel k (see Chapter 2), such that

Kernel Trick

$$\langle \Phi(x), \Phi(x_i) \rangle = k(x, x_i), \tag{7.24}$$

leading to decision functions of the form (cf. (7.20))

$$f(x) = \mathrm{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b \right). \tag{7.25}$$

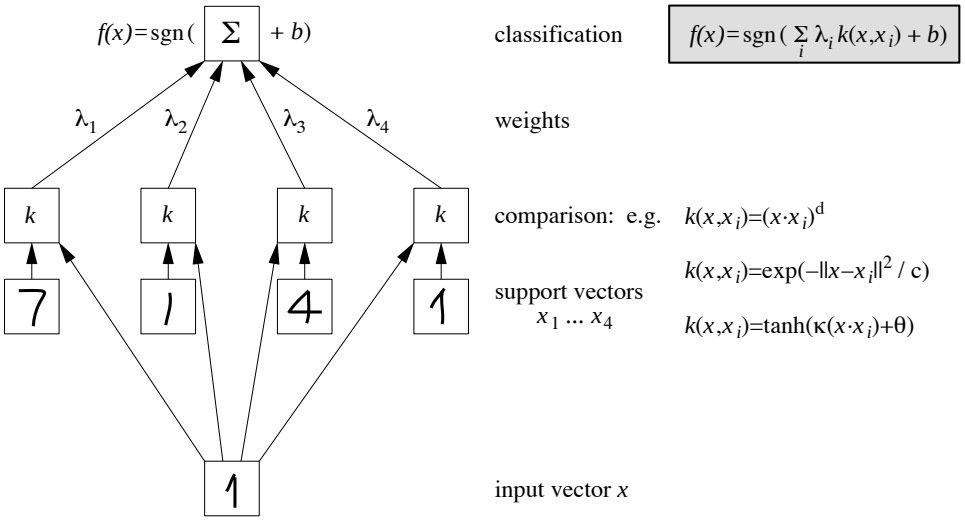

Figure 7.7 Architecture of SVMs. The kernel function k is chosen a priori; it determines the type of classifier (for instance, polynomial classifier, radial basis function classifier, or neural network). All other parameters (number of hidden units, weights, threshold b) are found during training, by solving a quadratic programming problem. The first layer weights x_i are a subset of the training set (the Support Vectors); the second layer weights $\lambda_i = y_i\alpha_i$ are computed from the Lagrange multipliers (cf. (7.25)).

At this point, a small aside regarding terminology is in order. As explained in Chapter 2, the input domain $\mathcal{X}$ need not be a vector space. Therefore, the Support Vectors in (7.25) (i.e., those x_i with $\alpha_i > 0$) are not necessarily vectors. One could choose to be on the safe side, and only refer to the corresponding $\Phi(x_i)$ as SVs. Common usage employs the term in a somewhat loose sense for both, however.

Consequently, everything that has been said about the linear case also applies to nonlinear cases, obtained using a suitable kernel k, instead of the Euclidean dot product (Figure 7.6). By using some of the kernel functions described in Chapter 2, the SV algorithm can construct a variety of learning machines (Figure 7.7), some of which coincide with classical architectures: *polynomial classifiers* of degree d,

Kernels

$$k(x, x_i) = \langle x, x_i \rangle^d, \tag{7.26}$$

radial basis function classifiers with Gaussian kernel of width $c > 0$,

$$k(x, x_i) = \exp\left(-\|x - x_i\|^2/c\right), \tag{7.27}$$

and *neural networks* (e.g., [49, 235]) with tanh activation function,

$$k(x, x_i) = \tanh(\kappa \langle x, x_i \rangle + \Theta). \tag{7.28}$$

The parameters $\kappa > 0$ and $\Theta \in \mathbb{R}$ are the gain and horizontal shift. As we shall see later, the tanh kernel can lead to very good results. Nevertheless, we should mention at this point that from a mathematical point of view, it has certain short-

comings, cf. the discussion following (2.69).

Quadratic Program

To find the decision function (7.25), we solve the following problem (cf. (7.17)):

$$\underset{\boldsymbol{\alpha}}{\text{maximize}}\, W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{7.29}$$

subject to the constraints (7.18) and (7.19).

If k is positive definite, $Q_{ij} := (y_i y_j k(x_i, x_j))_{ij}$ is a positive definite matrix (Problem 7.6), which provides us with a convex problem that can be solved efficiently (cf. Chapter 6). To see this, note that (cf. Proposition 2.16)

$$\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) = \left\langle \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i), \sum_{j=1}^{m} \alpha_j y_j \Phi(x_j) \right\rangle \geq 0, \tag{7.30}$$

for all $\boldsymbol{\alpha} \in \mathbb{R}^m$.

As described in Chapter 2, we can actually use a larger class of kernels without destroying the convexity of the quadratic program. This is due to the fact that the constraint (7.19) excludes certain parts of the space of multipliers α_i. As a result, we only need the kernel to be positive definite on the remaining points. This is precisely guaranteed if we require k to be *conditionally* positive definite (see Definition 2.21). In this case, we have $\boldsymbol{\alpha}^\top Q \boldsymbol{\alpha} \geq 0$ for all coefficient vectors $\boldsymbol{\alpha}$ satisfying (7.19).

Threshold

To compute the threshold b, we take into account that due to the KKT conditions (7.16), $\alpha_j > 0$ implies (using (7.24))

$$\sum_{i=1}^{m} y_i \alpha_i k(x_j, x_i) + b = y_j. \tag{7.31}$$

Thus, the threshold can for instance be obtained by averaging

$$b = y_j - \sum_{i=1}^{m} y_i \alpha_i k(x_j, x_i), \tag{7.32}$$

over all points with $\alpha_j > 0$; in other words, all SVs. Alternatively, one can compute b from the value of the corresponding double dual variable; see Section 10.3 for details. Sometimes it is also useful not to use the "optimal" b, but to change it in order to adjust the number of false positives and false negatives.

Figure 1.7 shows how a simple binary toy problem is solved, using a Support Vector Machine with a radial basis function kernel (7.27). Note that the SVs are the patterns closest to the decision boundary — not only in the feature space, where by construction, the SVs are the patterns closest to the separating hyperplane, but also in the input space depicted in the figure. This feature differentiates SVMs from other types of classifiers. Figure 7.8 shows both the SVs and the centers extracted by k-means, which are the expansion patterns that a classical RBF network approach would employ.

Comparison to RBF Network

In a study comparing the two approaches on the USPS problem of handwritten character recognition, a SVM with a Gaussian kernel outperformed the classical RBF network using Gaussian kernels [482]. A hybrid approach, where the SVM

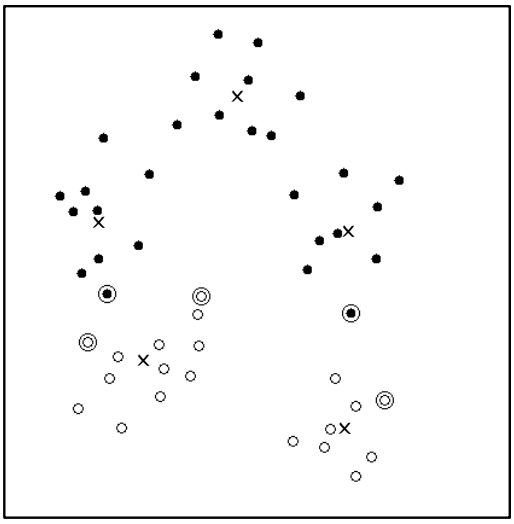

Figure 7.8 RBF centers automatically computed by the Support Vector algorithm (indicated by extra circles), using a Gaussian kernel. The number of SV centers accidentally coincides with the number of identifiable clusters (indicated by crosses found by k-means clustering, with $k = 2$ and $k = 3$ for balls and circles, respectively), but the naive correspondence between clusters and centers is lost; indeed, 3 of the SV centers are circles, and only 2 of them are balls. Note that the SV centers are chosen with respect to the classification task to be solved (from [482]).

algorithm was used to identify the centers (or hidden units) for the RBF network (that is, as a replacement for k-means), exhibited a performance which was in between the previous two. The study concluded that the SVM algorithm yielded two advantages. First, it better identified good expansion patterns, and second, its large margin regularizer led to second-layer weights that generalized better. We should add, however, that using clever engineering, the classical RBF algorithm can be improved to achieve a performance close to the one of SVMs [427].

7.5 Soft Margin Hyperplanes

So far, we have not said much about when the above will actually work. In practice, a separating hyperplane need not exist; and even if it does, it is not always the best solution to the classification problem. After all, an individual outlier in a data set, for instance a pattern which is mislabelled, can crucially affect the hyperplane. We would rather have an algorithm which can tolerate a certain fraction of outliers.

A natural idea might be to ask for the algorithm to return the hyperplane that leads to the *minimal* number of training errors. Unfortunately, it turns out that this is a combinatorial problem. Worse still, the problem is even hard to *approximate*: Ben-David and Simon [34] have recently shown that it is NP-hard to find a hyperplane whose training error is worse by some constant factor than the optimal one. Interestingly, they also show that this can be alleviated by taking into account the concept of the *margin*. By disregarding points that are within some fixed positive margin of the hyperplane, then the problem has polynomial complexity.

Cortes and Vapnik [111] chose a different approach for the SVM, following [40].