7 Pattern Recognition

This chapter is devoted to a detailed description of SV classification (SVC) methods. We have already briefly visited the SVC algorithm in Chapter 1. There will be some overlap with that chapter, but here we give a more thorough treatment.

Overview

We start by describing the classifier that forms the basis for SVC, the separating hyperplane (Section 7.1). Separating hyperplanes can differ in how large a margin of separation they induce between the classes, with corresponding consequences on the generalization error, as discussed in Section 7.2. The "optimal" margin hyperplane is defined in Section 7.3, along with a description of how to compute it. Using the kernel trick of Chapter 2, we generalize to the case where the optimal margin hyperplane is not computed in input space, but in a feature space nonlinearly related to the latter (Section 7.4). This dramatically increases the applicability of the approach, as does the introduction of slack variables to deal with outliers and noise in the data (Section 7.5). Many practical problems require us to classify the data into more than just two classes. Section 7.6 describes how multi-class SV classification systems can be built. Following this, Section 7.7 describes some variations on standard SV classification algorithms, differing in the regularizers and constraints that are used. We conclude with a fairly detailed section on experiments and applications (Section 7.8).

Prerequisites

This chapter requires basic knowledge of kernels, as conveyed in the first half of Chapter 2. To understand details of the optimization problems, it is helpful (but not indispensable) to get some background from Chapter 6. To understand the connections to learning theory, in particular regarding the statistical basis of the regularizer used in SV classification, it would be useful to have read Chapter 5.

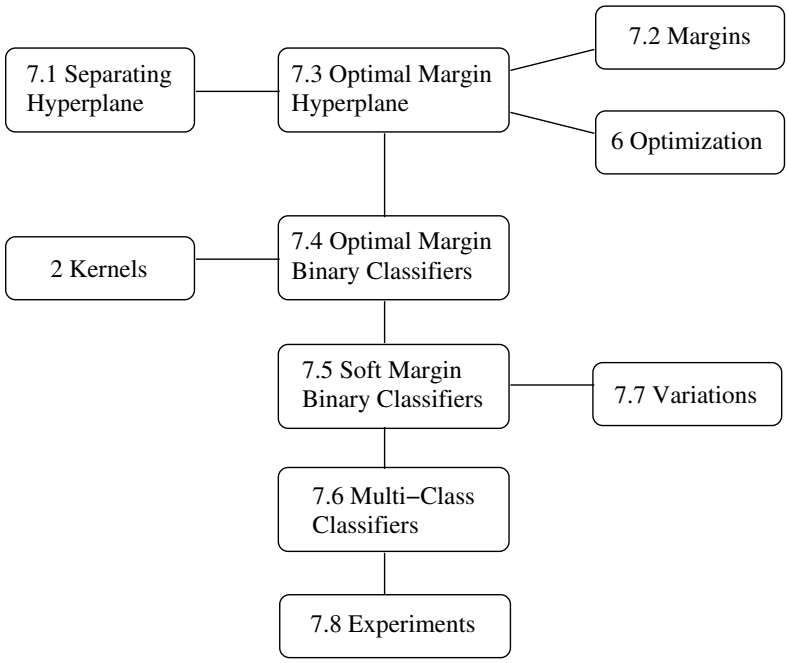
7.1 Separating Hyperplanes

Hyperplane

Suppose we are given a dot product space $\mathcal{H}$, and a set of pattern vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{H}$. Any hyperplane in $\mathcal{H}$ can be written as

$$\{\mathbf{x} \in \mathcal{H} \,|\, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}, \quad \mathbf{w} \in \mathcal{H}, b \in \mathbb{R}. \tag{7.1}$$

In this formulation, $\mathbf{w}$ is a vector orthogonal to the hyperplane: If $\mathbf{w}$ has unit length, then $\langle \mathbf{w}, \mathbf{x} \rangle$ is the length of $\mathbf{x}$ along the direction of $\mathbf{w}$ (Figure 7.1). For general $\mathbf{w}$, this number will be scaled by $\|\mathbf{w}\|$. In any case, the set (7.1) consists



of vectors that all have the same length along $\mathbf{w}$. In other words, these are vectors that project onto the same point on the line spanned by $\mathbf{w}$.

In this formulation, we still have the freedom to multiply $\mathbf{w}$ and b by the same non-zero constant. This superfluous freedom — physicists would call it a "gauge" freedom — can be abolished as follows.

Definition 7.1 (Canonical Hyperplane) *The pair* $(\mathbf{w}, b) \in \mathcal{H} \times \mathbb{R}$ *is called a* canonical *form of the hyperplane (7.1) with respect to* $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{H}$*, if it is scaled such that*

$$\min_{i=1,\ldots,m} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1, \tag{7.2}$$

which amounts to saying that the point closest to the hyperplane has a distance of $1/\|\mathbf{w}\|$ *(Figure 7.2).*

Note that the condition (7.2) still allows two such pairs: given a canonical hyperplane $(\mathbf{w}, b)$, another one satisfying (7.2) is given by $(-\mathbf{w}, -b)$. For the purpose of pattern recognition, these two hyperplanes turn out to be different, as they are oriented differently; they correspond to two *decision functions*,

Decision Function

$$f_{\mathbf{w},b} : \mathcal{H} \to \{\pm 1\}$$
$$\mathbf{x} \mapsto f_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right), \tag{7.3}$$

which are the inverse of each other.

In the absence of class labels $y_i \in \{\pm 1\}$ associated with the $\mathbf{x}_i$, there is no way of distinguishing the two hyperplanes. For a *labelled* dataset, a distinction exists: The two hyperplanes make opposite class assignments. In pattern recognition,

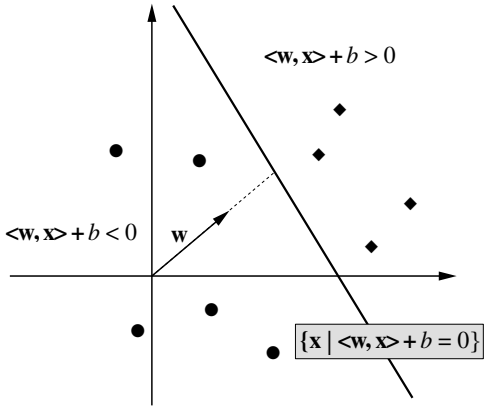


Figure 7.1 A separable classification problem, along with a separating hyperplane, written in terms of an orthogonal weight vector $\mathbf{w}$ and a threshold b. Note that by multiplying both $\mathbf{w}$ and b by the same non-zero constant, we obtain the same hyperplane, represented in terms of different parameters. Figure 7.2 shows how to eliminate this scaling freedom.

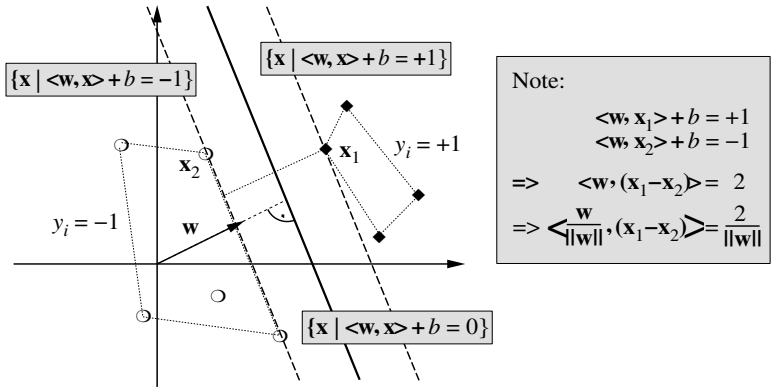


Figure 7.2 By requiring the scaling of $\mathbf{w}$ and b to be such that the point(s) closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of a hyperplane. Note that in this case, the margin, measured perpendicularly to the hyperplane, equals $1/\|\mathbf{w}\|$. This can be seen by considering two opposite points which precisely satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$ (cf. Problem 7.4)

we attempt to find a solution $f_{\mathbf{w},b}$ which *correctly classifies* the labelled examples $(\mathbf{x}_i, y_i) \in \mathcal{H} \times \{\pm 1\}$; in other words, which satisfies $f_{\mathbf{w},b}(\mathbf{x}_i) = y_i$ for all i (in this case, the training set is said to be *separable*), or at least for a large fraction thereof.

The next section will introduce the term *margin*, to denote the distance to a separating hyperplane from the point closest to it. It will be argued that to generalize well, a large margin should be sought. In view of Figure 7.2, this can be achieved by keeping $\|\mathbf{w}\|$ small. Readers who are content with this level of detail may skip the next section and proceed directly to Section 7.3, where we describe how to construct the hyperplane with the largest margin.

7.2 The Role of the Margin

The margin plays a crucial role in the design of SV learning algorithms. Let us start by formally defining it.

Definition 7.2 (Geometrical Margin) *For a hyperplane* $\{\mathbf{x} \in \mathcal{H} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$*, we call*

$$\rho_{(\mathbf{w},b)}(\mathbf{x}, y) := y(\langle \mathbf{w}, \mathbf{x} \rangle + b)/||\mathbf{w}|| \tag{7.4}$$

Geometrical Margin

the geometrical margin *of the point* $(\mathbf{x}, y) \in \mathcal{H} \times \{\pm 1\}$*. The minimum value*

$$\rho_{(\mathbf{w},b)} := \min_{i=1,\ldots,m} \rho_{(\mathbf{w},b)}(\mathbf{x}_i, y_i) \tag{7.5}$$

shall be called the geometrical margin *of* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$*. If the latter is omitted, it is understood that the training set is meant.*

Occasionally, we will omit the qualification *geometrical*, and simply refer to the *margin*.

For a point $(\mathbf{x}, y)$ which is correctly classified, the margin is simply the distance from $\mathbf{x}$ to the hyperplane. To see this, note first that the margin is zero *on* the hyperplane. Second, in the definition, we effectively consider a hyperplane

$$(\hat{\mathbf{w}}, \hat{b}) := (\mathbf{w}/||\mathbf{w}||, b/||\mathbf{w}||), \tag{7.6}$$

which has a unit length weight vector, and then compute the quantity $y(\langle \hat{\mathbf{w}}, \mathbf{x} \rangle + \hat{b})$. The term $\langle \hat{\mathbf{w}}, \mathbf{x} \rangle$, however, simply computes the length of the projection of $\mathbf{x}$ onto the direction orthogonal to the hyperplane, which, after adding the offset $\hat{b}$, equals the distance to it. The multiplication by y ensures that the margin is positive whenever a point is correctly classified. For misclassified points, we thus get a margin which equals the *negative* distance to the hyperplane. Finally, note that for canonical hyperplanes, the margin is $1/||\mathbf{w}||$ (Figure 7.2). The definition of the canonical hyperplane thus ensures that the length of $\mathbf{w}$ now corresponds to a meaningful geometrical quantity.

Margin of Canonical Hyperplanes

It turns out that the margin of a separating hyperplane, and thus the length of the weight vector $\mathbf{w}$, plays a fundamental role in support vector type algorithms. Loosely speaking, if we manage to separate the training data with a large margin, then we have reason to believe that we will do well on the test set. Not surprisingly, there exist a number of explanations for this intuition, ranging from the simple to the rather technical. We will now briefly sketch some of them.

Insensitivity to Pattern Noise

The simplest possible justification for large margins is as follows. Since the training and test data are assumed to have been generated by the same underlying dependence, it seems reasonable to assume that most of the test patterns will lie close (in $\mathcal{H}$) to at least one of the training patterns. For the sake of simplicity, let us consider the case where *all* test points are generated by adding bounded pattern noise (sometimes called input noise) to the training patterns. More precisely, given a training point $(\mathbf{x}, y)$, we will generate test points of the form $(\mathbf{x} + \Delta\mathbf{x}, y)$, where

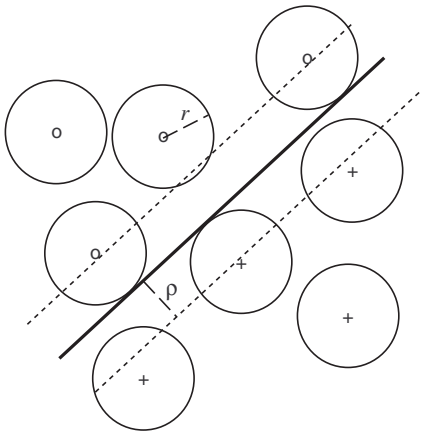


Figure 7.3 Two-dimensional toy example of a classification problem: Separate 'o' from '+' using a hyperplane. Suppose that we add bounded noise to each pattern. If the optimal margin hyperplane has margin ρ, and the noise is bounded by $r < \rho$, then the hyperplane will correctly separate even the noisy patterns. Conversely, if we ran the perceptron algorithm (which finds *some* separating hyperplane, but not necessarily the optimal one) on the noisy data, then we would recover the optimal hyperplane in the limit $r \to \rho$.

$\Delta\mathbf{x} \in \mathcal{H}$ is bounded in norm by some $r > 0$. Clearly, if we manage to separate the training set with a margin $\rho > r$, we will correctly classify *all* test points: Since all training points have a distance of at least ρ to the hyperplane, the test patterns will still be on the correct side (Figure 7.3, cf. also [152]).

If we knew ρ beforehand, then this could actually be turned into an optimal margin classifier training algorithm, as follows. If we use an r which is slightly smaller than ρ, then even the patterns with added noise will be separable with a nonzero margin. In this case, the standard perceptron algorithm can be shown to converge.[1]

Therefore, we can run the perceptron algorithm on the noisy patterns. If the algorithm finds a sufficient number of noisy versions of each pattern, with different perturbations $\Delta\mathbf{x}$, then the resulting hyperplane will not intersect any of the balls depicted in Figure 7.3. As r approaches ρ, the resulting hyperplane should better approximate the maximum margin solution (the figure depicts the limit $r = \rho$). This constitutes a connection between training with pattern noise and maximizing the margin. The latter, in turn, can be thought of as a regularizer, comparable to those discussed earlier (see Chapter 4 and (2.49)). Similar connections to training with noise, for other types of regularizers, have been pointed out before for neural networks [50].

1. Rosenblatt's perceptron algorithm [439] is one of the simplest conceivable iterative procedures for computing a separating hyperplane. In its simplest form, it proceeds as follows. We start with an arbitrary weight vector $\mathbf{w}_0$. At step $n \in \mathbb{N}$, we consider the training example $(\mathbf{x}_n, y_n)$. If it is classified correctly using the current weight vector (i.e., if $\operatorname{sgn}\langle \mathbf{x}_n, \mathbf{w}_{n-1}\rangle = y_n$), we set $\mathbf{w}_n := \mathbf{w}_{n-1}$; otherwise, we set $\mathbf{w}_n := \mathbf{w}_{n-1} + \eta y_i \mathbf{x}_i$ (here, $\eta > 0$ is a learning rate). We thus loop over all patterns repeatedly, until we can complete one full pass through the training set without a single error. The resulting weight vector will thus classify all points correctly. Novikoff [386] proved that this procedure terminates, provided that the training set is separable with a nonzero margin.

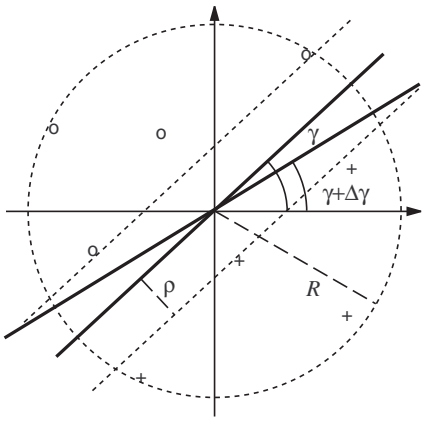


Figure 7.4 Two-dimensional toy example of a classification problem: Separate 'o' from '+' using a hyperplane passing through the origin. Suppose the patterns are bounded in length (distance to the origin) by R, and the classes are separated by an optimal hyperplane (parametrized by the angle γ) with margin ρ. In this case, we can perturb the parameter by some $\Delta\gamma$ with $|\Delta\gamma| < \arcsin \frac{\rho}{R}$, and still correctly separate the data.

Parameter Noise

A similar robustness argument can be made for the dependence of the hyperplane on the parameters $(\mathbf{w}, b)$ (cf. [504]). If all points lie at a distance of at least ρ from the hyperplane, and the patterns are bounded in length, then small perturbations to the hyperplane parameters will not change the classification of the training data (see Figure 7.4).[2] Being able to perturb the parameters of the hyperplane amounts to saying that to store the hyperplane, we need fewer bits than we would for a hyperplane whose *exact* parameter settings are crucial. Interestingly, this is related to what is called the Minimum Description Length principle ([583, 433, 485], cf. also [522, 305, 94]): The best description of the data, in terms of generalization error, should be the one that requires the fewest bits to store.

VC Margin Bound

We now move on to a more technical justification of large margin algorithms. For simplicity, we only deal with hyperplanes that have offset $b = 0$, leaving $f(\mathbf{x}) = \operatorname{sgn} \langle \mathbf{w}, \mathbf{x} \rangle$. The theorem below follows from a result in [24].

Theorem 7.3 (Margin Error Bound) *Consider the set of decision functions* $f(\mathbf{x}) = \operatorname{sgn} \langle \mathbf{w}, \mathbf{x} \rangle$ *with* $||\mathbf{w}|| \leq \Lambda$ *and* $||\mathbf{x}|| \leq R$, *for some* $R, \Lambda > 0$. *Moreover, let* $\rho > 0$, *and* ν *denote the fraction of training examples with margin smaller than* $\rho/||\mathbf{w}||$, *referred to as the* margin error.

Margin Error

For all distributions P *generating the data, with probability at least* $1 - \delta$ *over the drawing of the* m *training patterns, and for any* $\rho > 0$ *and* $\delta \in (0, 1)$, *the probability that a test pattern drawn from* P *will be misclassified is bounded from above, by*

$$\nu + \sqrt{\frac{c}{m}\left(\frac{R^2\Lambda^2}{\rho^2}\ln^2 m + \ln(1/\delta)\right)}. \tag{7.7}$$

Here, c is a universal constant.

2. Note that this would not hold true if we allowed patterns of arbitrary length — this type of restriction of the pattern lengths pops up in various places, such as Novikoff's theorem [386], Vapnik's VC dimension bound for margin classifiers (Theorem 5.5), and Theorem 7.3.

Let us try to understand this theorem. It makes a probabilistic statement about a probability, by giving an upper bound on the probability of test error, which *itself* only holds true with a certain probability, $1-\delta$. Where do these two probabilities come from? The first is due to the fact that the *test* examples are randomly drawn from P; the second is due to the *training* examples being drawn from P. Strictly speaking, the bound does not refer to a *single* classifier that has been trained on some fixed data set at hand, but to an ensemble of classifiers, trained on various instantiations of training sets generated by the same underlying regularity P.

It is beyond the scope of the present chapter to prove this result. The basic ingredients of bounds of this type, commonly referred to as *VC bounds*, are described in Chapter 5; for further details, see Chapter 12, and [562, 491, 504, 125]. Several aspects of the bound are noteworthy. The test error is bounded by a sum of the margin error ν, and a capacity term (the $\sqrt{\ldots}$ term in (7.7)), with the latter tending to zero as the number of examples, m, tends to infinity. The capacity term can be kept small by keeping R and Λ small, and making ρ large. If we assume that R and Λ are fixed a priori, the main influence is ρ. As can be seen from (7.7), a large ρ leads to a small capacity term, but the margin error ν gets larger. A small ρ, on the other hand, will usually cause fewer points to have margins smaller than $\rho/||\mathbf{w}||$, leading to a smaller margin error; but the capacity penalty will increase correspondingly. The overall message: Try to find a hyperplane which is aligned such that even for a large ρ, there are few margin errors.

Maximizing ρ, however, is the same as minimizing the length of $\mathbf{w}$. Hence we might just as well keep ρ fixed, say, equal to 1 (which is the case for canonical hyperplanes), and search for a hyperplane which has a small $||\mathbf{w}||$ and few points with a margin smaller than $1/||\mathbf{w}||$; in other words (Definition 7.2), few points such that $y\langle \mathbf{w}, \mathbf{x}\rangle < 1$.

It should be emphasized that dropping the condition $||\mathbf{w}|| \leq \Lambda$ would prevent us from stating a bound of the kind shown above. We could give an alternative bound, where the capacity depends on the dimensionality of the space $\mathcal{H}$. The crucial advantage of the bound given above is that it is independent of that dimensionality, enabling us to work in very high dimensional spaces. This will become important when we make use of the kernel trick.

It has recently been pointed out that the margin also plays a crucial role in improving asymptotic rates in nonparametric estimation [551]. This topic, however, is beyond the scope of the present book.

Implementation in Hardware

To conclude this section, we note that large margin classifiers also have advantages of a practical nature: An algorithm that can separate a dataset with a certain margin will behave in a benign way when implemented in hardware. Real-world systems typically work only within certain accuracy bounds, and if the classifier is insensitive to small changes in the inputs, it will usually tolerate those inaccuracies.

We have thus accumulated a fair amount of evidence in favor of the following approach: Keep the margin training error small, and the margin large, in order to achieve high generalization ability. In other words, hyperplane decision functions

should be constructed such that they maximize the margin, and at the same time separate the training data with as few exceptions as possible. Sections 7.3 and 7.5 respectively will deal with these two issues.

7.3 Optimal Margin Hyperplanes

Let us now derive the optimization problem to be solved for computing the optimal hyperplane. Suppose we are given a set of examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, $\mathbf{x}_i \in \mathcal{H}$, $y_i \in \{\pm 1\}$. Here and below, the index i runs over $1, \ldots, m$ by default. We assume that there is at least one negative and one positive y_i. We want to find a decision function $f_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right)$ satisfying

$$f_{\mathbf{w},b}(\mathbf{x}_i) = y_i. \tag{7.8}$$

If such a function exists (the non-separable case will be dealt with later), canonicality (7.2) implies

$$y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b\right) \geq 1. \tag{7.9}$$

As an aside, note that out of the two canonical forms of the same hyperplane, $(\mathbf{w}, b)$ and $(-\mathbf{w}, -b)$, only one will satisfy equations (7.8) and (7.11). The existence of class labels thus allows to distinguish two orientations of a hyperplane.

Following the previous section, a separating hyperplane which generalizes well can thus be constructed by solving the following problem:

$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2, \tag{7.10}$$

$$\text{subject to} \quad y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b\right) \geq 1 \text{ for all } i = 1, \ldots, m. \tag{7.11}$$

This is called the *primal optimization problem*.

Problems like this one are the subject of optimization theory. For details on how to solve them, see Chapter 6; for a short intuitive explanation, cf. the remarks following (1.26) in the introductory chapter. We will now derive the so-called *dual problem*, which can be shown to have the same solutions as (7.10). In the present case, it will turn out that it is more convenient to deal with the dual. To derive it, we introduce the Lagrangian,

Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1\right), \tag{7.12}$$

with Lagrange multipliers $\alpha_i \geq 0$. Recall that as in Chapter 1, we use bold face Greek variables to refer to the corresponding vectors of variables, for instance, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$.

The Lagrangian L must be maximized with respect to α_i, and minimized with respect to $\mathbf{w}$ and b (see Theorem 6.26). Consequently, at this saddle point, the

derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b}L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial \mathbf{w}}L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{7.13}$$

which leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{7.14}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{7.15}$$

The solution vector thus has an expansion in terms of training examples. Note that although the solution $\mathbf{w}$ is unique (due to the strict convexity of (7.10), and the convexity of (7.11)), the coefficients α_i need not be.

According to the KKT theorem (Chapter 6), only the Lagrange multipliers α_i that are non-zero at the saddle point, correspond to constraints (7.11) which are precisely met. Formally, for all $i = 1, \ldots, m$, we have

$$\alpha_i[y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1] = 0. \tag{7.16}$$

Support Vectors

The patterns $\mathbf{x}_i$ for which $\alpha_i > 0$ are called *Support Vectors*. This terminology is related to corresponding terms in the theory of convex sets, relevant to convex optimization (e.g., [334, 45]).[3] According to (7.16), they lie exactly on the margin.[4] All remaining examples in the training set are irrelevant: Their constraints (7.11) are satisfied automatically, and they do not appear in the expansion (7.15), since their multipliers satisfy $\alpha_i = 0$.[5]

This leads directly to an upper bound on the generalization ability of optimal margin hyperplanes. To this end, we consider the so-called leave-one-out method (for further details, see Section 12.2) to estimate the expected test error [335, 559]. This procedure is based on the idea that if we leave out one of the training

3. Given any boundary point of a convex set, there always exists a hyperplane separating the point from the interior of the set. This is called a *supporting hyperplane*.

SVs lie on the boundary of the convex hulls of the two classes, thus they possess supporting hyperplanes. The SV optimal hyperplane is the hyperplane which lies in the middle of the two parallel supporting hyperplanes (of the two classes) with maximum distance.

Conversely, from the optimal hyperplane, we can obtain supporting hyperplanes for all SVs of both classes, by shifting it by $1/\|\mathbf{w}\|$ in both directions.

4. Note that this implies the solution $(\mathbf{w}, b)$, where b is computed using $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$ for SVs, *is* in canonical form with respect to the training data. (This makes use of the reasonable assumption that the training set contains both positive and negative examples.)

5. In a statistical mechanics framework, Anlauf and Biehl [12] have put forward a similar argument for the *optimal stability perceptron*, also computed using constrained optimization. There is a large body of work in the physics community on optimal margin classification. Some further references of interest are [310, 191, 192, 394, 449, 141]; other early works include [313].

examples, and train on the remaining ones, then the probability of error on the left out example gives us a fair indication of the true test error. Of course, doing this for a single training example leads to an error of either zero or one, so it does not yet give an estimate of the test error. The leave-one-out method *repeats* this procedure for each individual training example in turn, and averages the resulting errors.

Let us return to the present case. If we leave out a pattern $\mathbf{x}_{i^*}$, and construct the solution from the remaining patterns, the following outcomes are possible (cf. (7.11)):

1. $y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) > 1$. In this case, the pattern is classified correctly and does not lie on the margin. These are patterns that would not have become SVs anyway.

2. $y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) = 1$. In other words, $\mathbf{x}_{i^*}$ exactly meets the constraint (7.11). In this case, the solution $\mathbf{w}$ does not change, even though the coefficients α_i would change: Namely, if $\mathbf{x}_{i^*}$ might have become a Support Vector (i.e., $\alpha_{i^*} > 0$) had it been kept in the training set. In that case, the fact that the solution is the same, no matter whether $\mathbf{x}_{i^*}$ is in the training set or not, means that $\mathbf{x}_{i^*}$ can be written as $\sum_{\mathrm{SVs}} \beta_i y_i \mathbf{x}_i$ with, $\beta_i \geq 0$. Note that condition 2 is *not* equivalent to saying that $\mathbf{x}_{i^*}$ may be written as some linear combination of the remaining Support Vectors: Since the sign of the coefficients in the linear combination is determined by the class of the respective pattern, not any linear combination will do. Strictly speaking, $\mathbf{x}_{i^*}$ must lie in the cone spanned by the $y_i \mathbf{x}_i$, where the $\mathbf{x}_i$ are all Support Vectors.[6] For more detail, see [565] and Section 12.2.

3. $0 < y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) < 1$. In this case, $\mathbf{x}_{i^*}$ lies within the margin, but still on the correct side of the decision boundary. Thus, the solution looks different from the one obtained with $\mathbf{x}_{i^*}$ in the training set (in that case, $\mathbf{x}_{i^*}$ would satisfy (7.11) after training); classification is nevertheless correct.

4. $y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) > 0$. This means that $\mathbf{x}_{i^*}$ is classified incorrectly.

Note that cases 3 and 4 necessarily correspond to examples which would have become SVs if kept in the training set; case 2 potentially includes such situations. Only case 4, however, leads to an error in the leave-one-out procedure. Consequently, we have the following result on the generalization error of optimal margin classifiers [570]:[7]

Leave-One-Out Bound

Proposition 7.4 *The expectation of the number of Support Vectors obtained during training on a training set of size m, divided by m, is an* upper bound *on the expected probability of test error of the SVM trained on training sets of size $m-1$.*[8]

6. Possible non-uniqueness of the solution's expansion in terms of SVs is related to zero Eigenvalues of $(y_i y_j k(x_i, x_j))_{ij}$, cf. Proposition 2.16. Note, however, the above caveat on the distinction between linear combinations, and linear combinations with coefficients of fixed sign.

7. It also holds for the generalized versions of optimal margin classifiers described in the following sections.

8. Note that the leave-one-out procedure performed with m training examples thus yields

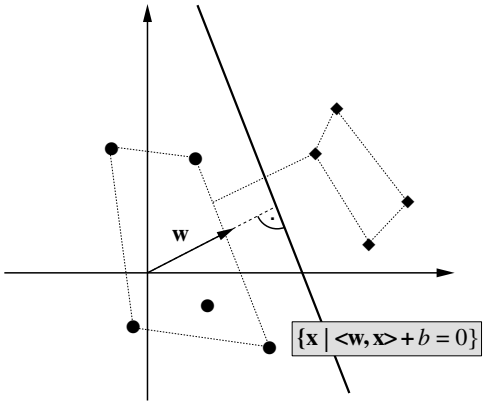


Figure 7.5 The optimal hyperplane (Figure 7.2) is the one bisecting the shortest connection between the convex hulls of the two classes.

A sharper bound can be formulated by making a further distinction in case 2, between SVs that must occur in the solution, and those that can be expressed in terms of the other SVs (see [570, 565, 268, 549] and Section 12.2).

We now return to the optimization problem to be solved. Substituting the conditions for the extremum, (7.14) and (7.15), into the Lagrangian (7.12), we arrive at the dual form of the optimization problem:

Quadratic Program of Optimal Margin Classifier

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle , \tag{7.17}$$

$$\text{subject to} \quad \alpha_i \geq 0, \quad i = 1, \ldots, m, \tag{7.18}$$

$$\text{and} \quad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{7.19}$$

On substitution of the expansion (7.15) into the decision function (7.3), we obtain an expression which can be evaluated in terms of dot products, taken between the pattern to be classified and the Support Vectors,

$$f(\mathbf{x}) = \operatorname{sgn} \left(\sum_{i=1}^{m} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right) . \tag{7.20}$$

To conclude this section, we note that there is an alternative way to derive the dual optimization problem [38]. To describe it, we first form the convex hulls C_+

a bound valid for training sets of size $m - 1$. This difference, however, does not usually mislead us too much. In statistical terms, the leave-one-out error is called *almost unbiased*. Note, moreover, that the statement talks about the *expected probability* of test error — there are thus *two* sources of randomness. One is the expectation over different training sets of size $m - 1$, the other is the probability of test error when one of the SVMs is faced with a test example drawn from the underlying distribution generating the data. For a generalization, see Theorem 12.9.

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-

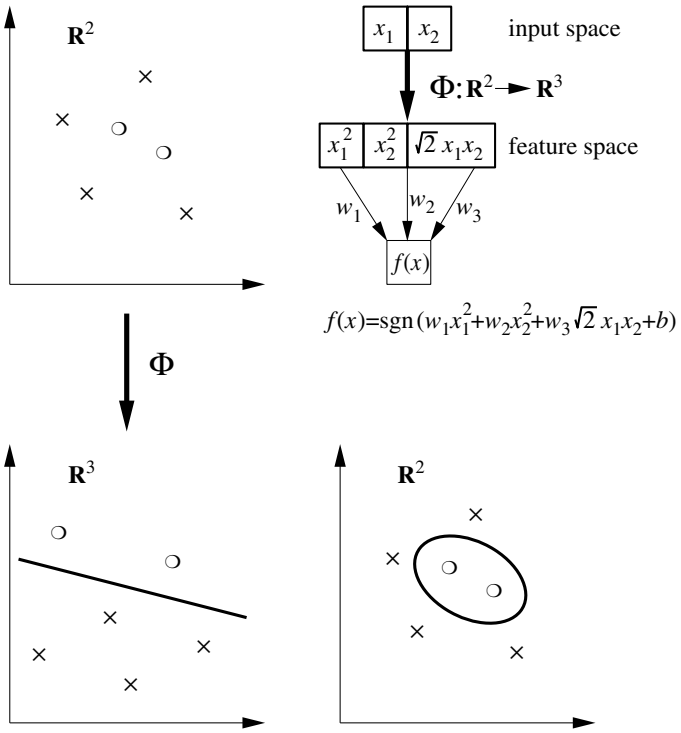


Figure 7.6 By mapping the input data *(top left)* nonlinearly (via Φ) into a higher-dimensional feature space $\mathcal{H}$ (here: $\mathcal{H} = \mathbb{R}^3$), and constructing a separating hyperplane there *(bottom left)*, an SVM *(top right)* corresponds to a nonlinear decision surface in input space (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))

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

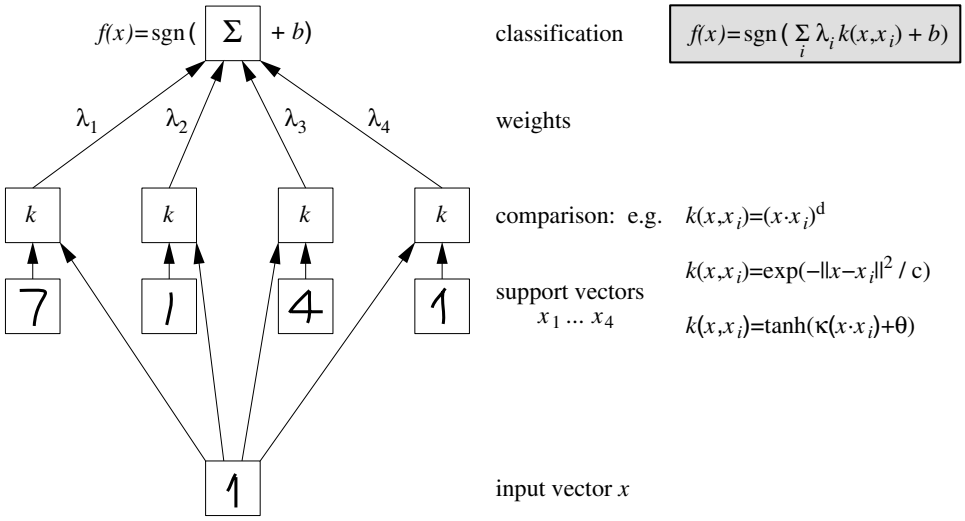


Figure 7.7 Architecture of SVMs. The kernel function k is chosen a priori; it determines the type of classifier (for instance, polynomial classifier, radial basis function classifier, or neural network). All other parameters (number of hidden units, weights, threshold b) are found during training, by 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

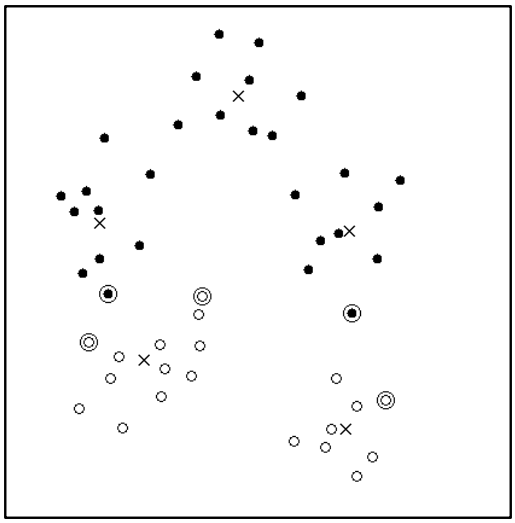


Figure 7.8 RBF centers automatically computed by the Support Vector algorithm (indicated by extra circles), using a Gaussian kernel. The number of SV centers accidentally coincides with the number of identifiable clusters (indicated by crosses found by k-means clustering, with $k = 2$ and $k = 3$ for 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].

Slack Variables

To allow for the possibility of examples violating (7.11), they introduced so-called slack variables,

$$\xi_i \geq 0, \quad \text{where } i = 1, \ldots, m, \tag{7.33}$$

and use relaxed separation constraints (cf. (7.11)),

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m. \tag{7.34}$$

Clearly, by making ξ_i large enough, the constraint on $(\mathbf{x}_i, y_i)$ can always be met. In order not to obtain the trivial solution where all ξ_i take on large values, we thus need to penalize them in the objective function. To this end, a term $\sum_i \xi_i$ is included in (7.10).

C-SVC

In the simplest case, referred to as the *C*-SV classifier, this is done by solving, for some $C > 0$,

$$\underset{\mathbf{w} \in \mathcal{H}, \boldsymbol{\xi} \in \mathbb{R}^m}{\text{minimize}} \quad \tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{m} \sum_{i=1}^{m} \xi_i, \tag{7.35}$$

subject to the constraints (7.33) and (7.34). It is instructive to compare this to Theorem 7.3, considering the case $\rho = 1$. Whenever the constraint (7.34) is met with $\xi_i = 0$, the corresponding point will not be a margin error. All non-zero slacks ξ correspond to margin errors; hence, roughly speaking, the fraction of margin errors in Theorem 7.3 increases with the second term in (7.35). The capacity term, on the other hand, increases with $||\mathbf{w}||$. Hence, for a suitable positive constant C, this approach approximately minimizes the right hand side of the bound.

Note, however, that if many of the ξ_i attain large values (in other words, if the classes to be separated strongly overlap, for instance due to noise), then $\sum_{i=1}^{m} \xi_i$ can be significantly larger than the fraction of margin errors. In that case, there is no guarantee that the hyperplane will generalize well.

As in the separable case (7.15), the solution can be shown to have an expansion

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{7.36}$$

where non-zero coefficients α_i can only occur if the corresponding example $(\mathbf{x}_i, y_i)$ precisely meets the constraint (7.34). Again, the problem only depends on dot products in $\mathcal{H}$, which can be computed by means of the kernel.

The coefficients α_i are found by solving the following quadratic programming problem:

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{7.37}$$

$$\text{subject to } 0 \leq \alpha_i \leq \frac{C}{m} \text{ for all } i = 1, \ldots, m, \tag{7.38}$$

$$\text{and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{7.39}$$

To compute the threshold b, we take into account that due to (7.34), for Support

Vectors x_j for which $\xi_j = 0$, we have (7.31). Thus, the threshold can be obtained by averaging (7.32) over all Support Vectors x_j (recall that they satisfy $\alpha_j > 0$) with $\alpha_j < C$.

In the above formulation, C is a constant determining the trade-off between two conflicting goals: minimizing the training error, and maximizing the margin. Unfortunately, C is a rather unintuitive parameter, and we have no a priori way to select it.[9] Therefore, a modification was proposed in [481], which replaces C by a parameter ν; the latter will turn out to control the number of margin errors and Support Vectors.

ν-SVC

As a primal problem for this approach, termed the ν-SV classifier, we consider

$$\underset{\mathbf{w}\in\mathcal{H},\boldsymbol{\xi}\in\mathbb{R}^m,\rho,b\in\mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w},\boldsymbol{\xi},\rho) = \frac{1}{2}\|\mathbf{w}\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i \tag{7.40}$$

$$\text{subject to} \quad y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq \rho - \xi_i \tag{7.41}$$

$$\text{and} \quad \xi_i \geq 0, \quad \rho \geq 0. \tag{7.42}$$

Note that no constant C appears in this formulation; instead, there is a parameter ν, and also an additional variable ρ to be optimized. To understand the role of ρ, note that for $\boldsymbol{\xi} = 0$, the constraint (7.41) simply states that the two classes are separated by the *margin* $2\rho/\|\mathbf{w}\|$ (cf. Problem 7.4).

Margin Error

To explain the significance of ν, let us first recall the term *margin error*: by this, we denote points with $\xi_i > 0$. These are points which are either errors, or lie within the margin. Formally, the fraction of margin errors is

$$R^{\rho}_{\text{emp}}[g] := \frac{1}{m}\left|\{i | y_i g(x_i) < \rho\}\right|. \tag{7.43}$$

Here, g is used to denote the argument of the sgn in the decision function (7.25): $f = \text{sgn} \circ g$. We are now in a position to state a result that explains the significance of ν.

ν-Property

Proposition 7.5 ([481]) *Suppose we run ν-SVC with k on some data with the result that $\rho > 0$. Then*

(i) ν is an upper bound on the fraction of margin errors.

(ii) ν is a lower bound on the fraction of SVs.

(iii) Suppose the data $(x_1, y_1), \ldots, (x_m, y_m)$ were generated iid from a distribution $\mathrm{P}(x, y) = \mathrm{P}(x)\mathrm{P}(y|x)$, such that neither $\mathrm{P}(x, y = 1)$ nor $\mathrm{P}(x, y = -1)$ contains any discrete component. Suppose, moreover, that the kernel used is analytic and non-constant. With probability 1, asymptotically, ν equals both the fraction of SVs and the fraction of errors.

The proof can be found in Section A.2.

Before we get into the technical details of the dual derivation, let us take a look

9. As a default value, we use $C/m = 10$ unless stated otherwise.

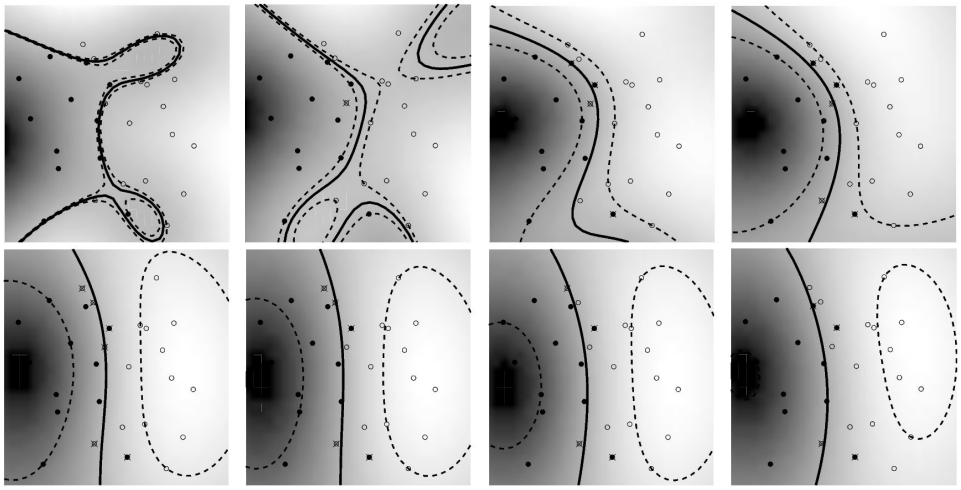


Figure 7.9 Toy problem (task: separate circles from disks) solved using ν-SV classification, with parameter values ranging from $\nu = 0.1$ (top left) to $\nu = 0.8$ (bottom right). The larger we make ν, the more points are allowed to lie inside the margin (depicted by dotted lines). Results are shown for a Gaussian kernel, $k(x, x') = \exp(-\|x - x'\|^2)$.

Table 7.1 Fractions of errors and SVs, along with the margins of class separation, for the toy example in Figure 7.9.
Note that ν upper bounds the fraction of errors and lower bounds the fraction of SVs, and that increasing ν, i.e., allowing more errors, increases the margin.

ν	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
fraction of errors	0.00	0.07	0.25	0.32	0.39	0.50	0.61	0.71
fraction of SVs	0.29	0.36	0.43	0.46	0.57	0.68	0.79	0.86
margin $\rho/\|\mathbf{w}\|$	0.005	0.018	0.115	0.156	0.364	0.419	0.461	0.546

at a toy example illustrating the influence of ν (Figure 7.9). The corresponding fractions of SVs and margin errors are listed in table 7.1.

Derivation of the Dual

The derivation of the ν-SVC dual is similar to the above SVC formulations, only slightly more complicated. We consider the Lagrangian

$$L(\mathbf{w}, \boldsymbol{\xi}, b, \rho, \boldsymbol{\alpha}, \boldsymbol{\beta}, \delta) = \frac{1}{2}\|\mathbf{w}\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i \\ - \sum_{i=1}^{m}(\alpha_i(y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - \rho + \xi_i) + \beta_i\xi_i) - \delta\rho, \tag{7.44}$$

using multipliers $\alpha_i, \beta_i, \delta \geq 0$. This function has to be minimized with respect to the primal variables $\mathbf{w}, \boldsymbol{\xi}, b, \rho$, and maximized with respect to the dual variables $\boldsymbol{\alpha}, \boldsymbol{\beta}, \delta$. To eliminate the former, we compute the corresponding partial derivatives

and set them to 0, obtaining the following conditions:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{7.45}$$

$$\alpha_i + \beta_i = 1/m, \tag{7.46}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{7.47}$$

$$\sum_{i=1}^{m} \alpha_i - \delta = \nu. \tag{7.48}$$

Again, in the *SV expansion* (7.45), the α_i that are non-zero correspond to a constraint (7.41) which is precisely met.

Substituting (7.45) and (7.46) into L, using $\alpha_i, \beta_i, \delta \geq 0$, and incorporating kernels for dot products, leaves us with the following quadratic optimization problem for ν-SV classification:

Quadratic Program for ν-SVC

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{7.49}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq \frac{1}{m}, \tag{7.50}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{7.51}$$

$$\sum_{i=1}^{m} \alpha_i \geq \nu. \tag{7.52}$$

As above, the resulting decision function can be shown to take the form

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i k(x, x_i) + b \right). \tag{7.53}$$

Compared with the C-SVC dual (7.37), there are two differences. First, there is an additional constraint (7.52).[10] Second, the linear term $\sum_{i=1}^{m} \alpha_i$ no longer appears in the objective function (7.49). This has an interesting consequence: (7.49) is now quadratically homogeneous in $\boldsymbol{\alpha}$. It is straightforward to verify that the same decision function is obtained if we start with the primal function

$$\tau(\mathbf{w}, \boldsymbol{\xi}, \rho) = \frac{1}{2}\|\mathbf{w}\|^2 + C\left(-\nu\rho + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right), \tag{7.54}$$

10. The additional constraint makes it more challenging to come up with efficient training algorithms for large datasets. So far, two approaches have been proposed which work well. One of them slightly modifies the primal problem in order to avoid the *other* equality constraint (related to the offset b) [98]. The other one is a direct generalization of a corresponding algorithm for C-SVC, which reduces the problem for each chunk to a linear system, and which does not suffer any disadvantages from the additional constraint [407, 408]. See also Sections 10.3.2, 10.4.3, and 10.6.3 for further details.

i.e., if one does use C, cf. Problem 7.16.

To compute the threshold b and the margin parameter ρ, we consider two sets $S_\pm$, of identical size $s > 0$, containing SVs x_i with $0 < \alpha_i < 1$ and $y_i = \pm 1$, respectively. Then, due to the KKT conditions, (7.41) becomes an equality with $\xi_i = 0$. Hence, in terms of kernels,

$$b = -\frac{1}{2s} \sum_{x \in S_+ \cup S_-} \sum_{j=1}^{m} \alpha_j y_j k(x, x_j), \tag{7.55}$$

$$\rho = \frac{1}{2s} \Big(\sum_{x \in S_+} \sum_{j=1}^{m} \alpha_j y_j k(x, x_j) - \sum_{x \in S_-} \sum_{j=1}^{m} \alpha_j y_j k(x, x_j) \Big). \tag{7.56}$$

Note that for the decision function, only b is actually required.

Connection ν-SVC — C-SVC

A connection to standard SV classification, and a somewhat surprising interpretation of the regularization parameter C, is described by the following result:

Proposition 7.6 (Connection ν-SVC — C-SVC [481]) *If ν-SV classification leads to $\rho > 0$, then C-SV classification, with C set a priori to $1/\rho$, leads to the same decision function.*

Proof If we minimize (7.40), and then fix ρ to minimize only over the remaining variables, nothing will change. Hence the solution $\mathbf{w}_0, b_0, \boldsymbol{\xi}_0$ minimizes (7.35), for $C = 1$, subject to (7.41). To recover the constraint (7.34), we rescale to the set of variables $\mathbf{w}' = \mathbf{w}/\rho, b' = b/\rho, \boldsymbol{\xi}' = \boldsymbol{\xi}/\rho$. This leaves us with the objective function (7.35), up to a constant scaling factor ρ^2, using $C = 1/\rho$. ■

For further details on the connection between ν-SVMs and C-SVMs, see [122, 38]. A complete account has been given by Chang and Lin [98], who show that for a given problem and kernel, there is an interval $[\nu_{\min}, \nu_{\max}]$ of admissible values for ν, with $0 \leq \nu_{\min} \leq \nu_{\max} \leq 1$. The boundaries of the interval are computed by considering $\sum_i \alpha_i$ as returned by the C-SVM in the limits $C \to \infty$ and $C \to 0$, respectively.

It has been noted that ν-SVMs have an interesting interpretation in terms of *reduced convex hulls* [122, 38] (cf. (7.21)). If a problem is non-separable, the convex hulls will no longer be disjoint. Therefore, it no longer makes sense to search for the shortest line connecting them, and the approach of (7.22) will fail. In this situation, it seems natural to reduce the convex hulls in size, by limiting the size of the coefficients c_i in (7.21) to some value $\nu \in (0, 1)$. Intuitively, this amounts to limiting the influence of individual points — note that in the original problem (7.22), two single points can already determine the solution. It is possible to show that the ν-SVM formulation solves the problem of finding the hyperplane orthogonal to the closest line connecting the *reduced* convex hulls [122].

Robustness and Outliers

We now move on to another aspect of soft margin classification. When we introduced the slack variables, we did not attempt to justify the fact that in the objective function, we used a penalizer $\sum_{i=1}^m \xi_i$. Why not use another penalizer, such as $\sum_{i=1}^m \xi_i^p$, for some $p \geq 0$ [111]? For instance, $p = 0$ would yield a penalizer

that exactly *counts* the number of margin errors. Unfortunately, however, it is also a penalizer that leads to a combinatorial optimization problem. Penalizers yielding optimization problems that are particularly convenient, on the other hand, are obtained for $p = 1$ and $p = 2$. By default, we use the former, as it possesses an additional property which is statistically attractive. As the following proposition shows, linearity of the target function in the slack variables ξ_i leads to a certain "outlier" resistance of the estimator. As above, we use the shorthand $\mathbf{x}_i$ for $\Phi(x_i)$.

Proposition 7.7 (Resistance of SV classification [481]) *Suppose* $\mathbf{w}$ *can be expressed in terms of the SVs which are not at bound,*

$$\mathbf{w} = \sum_{i=1}^{m} \gamma_i \mathbf{x}_i \tag{7.57}$$

with $\gamma_i \neq 0$ *only if* $\alpha_i \in (0, 1/m)$ *(where the* α_i *are the coefficients of the dual solution). Then local movements of any margin error* $\mathbf{x}_m$ *parallel to* $\mathbf{w}$ *do not change the hyperplane.*[11]

The proof can be found in Section A.2. For further results in support of the $p = 1$ case, see [527].

Note that the assumption (7.57) is not as restrictive as it may seem. Even though the SV expansion of the solution, $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$, often contains many multipliers α_i which are at bound, it is nevertheless quite conceivable, especially when discarding the requirement that the coefficients be bounded, that we can obtain an expansion (7.57) in terms of a subset of the original vectors.

For instance, if we have a 2-D problem that we solve directly in input space, i.e., with $k(x, x') = \langle x, x' \rangle$, then it suffices to have two linearly independent SVs which are not at bound, in order to express $\mathbf{w}$. This holds true regardless of whether or not the two classes overlap, even if there are many SVs which are at the upper bound. Further information on resistance and robustness of SVMs can be found in Sections 3.4 and 9.3.

We have introduced SVs as those training examples x_i for which $\alpha_i > 0$. In some cases, it is useful to further distinguish different types of SVs. For reference purposes, we give a list of different types of SVs (Table 7.2).

In Section 7.3, we used the KKT conditions to argue that in the hard margin case, the SVs lie exactly on the margin. Using an identical argument for the soft margin case, we see that in this instance, in-bound SVs lie on the margin (Problem 7.9).

Note that in the hard margin case, where $\alpha_{\max} = \infty$, every SV is an in-bound SV. Note, moreover, that for kernels that produce full-rank Gram matrices, such as the Gaussian (Theorem 2.18), in theory every SV is essential (provided there are no duplicate patterns in the training set).[12]

11. Note that the perturbation of the point is carried out in feature space. What it precisely corresponds to in input space therefore depends on the specific kernel chosen.
12. In practice, Gaussian Gram matrices usually have some eigenvalues that are close to 0.

Table 7.2 Overview of different types of SVs. In each case, the condition on the Lagrange multipliers α_i (corresponding to an SV x_i) is given. In the table, $\alpha_{\max}$ stands for the upper bound in the optimization problem; for instance, $\alpha_{\max} = \frac{C}{m}$ in (7.38) and $\alpha_{\max} = \frac{1}{m}$ in (7.50).

Type of SV	Definition	Properties
(standard) SV	$0 < \alpha_i$	lies on or in margin
in-bound SV	$0 < \alpha_i < \alpha_{\max}$	lies on margin
bound SV	$\alpha_i = \alpha_{\max}$	usually lies in margin ("margin error")
essential SV	appears in all possible expansions of solution	becomes margin error when left out (Section 7.3)

7.6 Multi-Class Classification

So far, we have talked about binary classification, where the class labels can only take two values: ± 1. Many real-world problems, however, have more than two classes — an example being the widely studied optical character recognition (OCR) problem. We will now review some methods for dealing with this issue.

7.6.1 One Versus the Rest

To get *M-class classifiers*, it is common to construct a set of binary classifiers $f^1, \ldots, f^M$, each trained to separate one class from the rest, and combine them by doing the multi-class classification according to the maximal output before applying the sgn function; that is, by taking

$$\operatorname*{argmax}_{j=1,\ldots,M} g^j(x), \quad \text{where } g^j(x) = \sum_{i=1}^{m} y_i \alpha_i^j k(x, x_i) + b^j \tag{7.58}$$

(note that $f^j(x) = \operatorname{sgn}(g^j(x))$, cf. (7.25)).

Reject Decisions

The values $g^j(x)$ can also be used for *reject decisions*. To see this, we consider the difference between the two largest $g^j(x)$ as a measure of confidence in the classification of x. If that measure falls short of a threshold θ, the classifier rejects the pattern and does not assign it to a class (it might instead be passed on to a human expert). This has the consequence that on the remaining patterns, a lower error rate can be achieved. Some benchmark comparisons report a quantity referred to as the *punt error*, which denotes the fraction of test patterns that must be rejected in order to achieve a certain accuracy (say 1% error) on the remaining test samples. To compute it, the value of θ is adjusted on the *test* set [64].

The main shortcoming of (7.58), sometimes called the *winner-takes-all* approach, is that it is somewhat heuristic. The binary classifiers used are obtained by training on different binary classification problems, and thus it is unclear whether their

real-valued outputs (before thresholding) are on comparable scales.[13] This can be a problem, since situations often arise where *several* binary classifiers assign the pattern to their respective class (or where *none* does); in this case, *one* class must be chosen by comparing the real-valued outputs.

In addition, binary one-versus-the-rest classifiers have been criticized for dealing with rather asymmetric problems. For instance, in digit recognition, the classifier trained to recognize class '7' is usually trained on many more negative than positive examples. We can deal with these asymmetries by using values of the regularization constant C which differ for the respective classes (see Problem 7.10). It has nonetheless been argued that the following approach, which is more symmetric from the outset, can be advantageous.

7.6.2 Pairwise Classification

In pairwise classification, we train a classifier for each possible pair of classes [178, 463, 233, 311]. For M classes, this results in $(M-1)M/2$ binary classifiers. This number is usually larger than the number of one-versus-the-rest classifiers; for instance, if $M = 10$, we need to train 45 binary classifiers rather than 10 as in the method above. Although this suggests large training times, the individual problems that we need to train on are significantly smaller, and if the training algorithm scales superlinearly with the training set size, it is actually possible to save time.

Similar considerations apply to the runtime execution speed. When we try to classify a test pattern, we evaluate all 45 binary classifiers, and classify according to which of the classes gets the highest number of votes. A vote for a given class is defined as a classifier putting the pattern into that class.[14] The individual classifiers, however, are usually smaller in size (they have fewer SVs) than they would be in the one-versus-the-rest 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.

Nevertheless, if M is large, and we evaluate the $(M-1)M/2$ classifiers, then the resulting system may be slower than the corresponding one-versus-the-rest SVM. To illustrate this weakness, consider the following hypothetical situation: Suppose, in a digit recognition task, that after evaluating the first few binary classifiers, both digit 7 and digit 8 seem extremely unlikely (they already "lost" on several classifiers). In that case, it would seem pointless to evaluate the 7-vs-8 classifier. This idea can be cast into a precise framework by embedding the binary classifiers into a directed acyclic graph. Each classification run then corresponds to a directed traversal of that graph, and classification can be much faster [411].

13. Note, however, that some effort has gone into developing methods for transforming the real-valued outputs into class probabilities [521, 486, 410].
14. Some care has to be exercised in tie-breaking. For further detail, see [311].

7.6.3 Error-Correcting Output Coding

The method of error-correcting output codes was developed in [142], and later adapted to the case of SVMs [5]. In a nutshell, the idea is as follows. Just as we can generate a binary problem from a multiclass problem by separating one class from the rest — digit 0 from digits 1 through 9, say — we can generate a large number of further binary problems by splitting the original set of classes into two subsets. For instance, we could separate the even digits from the odd ones, or we could separate digits 0 through 4 from 5 through 9. It is clear that if we design a set of binary classifiers $f^1, \ldots, f^L$ in the right way, then the binary responses will completely determine the class of a test patterns. Each class corresponds to a unique vector in $\{\pm 1\}^L$; for M classes, we thus get a so-called *decoding matrix* $M \in \{\pm 1\}^{M \times L}$. What happens if the binary responses are inconsistent with each other; if, for instance, the problem is noisy, or the training sample is too small to estimate the binary classifiers reliably? Formally, this means that we will obtain a vector of responses $f^1(x), \ldots, f^L(x)$ which does not occur in the matrix M. To deal with these cases, [142] proposed designing a clever set of binary problems, which yields robustness against some errors. Here, the closest match between the vector of responses and the rows of the matrix is determined using the Hamming distance (the number of entries where the two vectors differ; essentially, the L_∞ distance). Now imagine a situation where the code is such that the minimal Hamming distance is three. In this case, we can *guarantee* that we will correctly classify all test examples which lead to at most one error amongst the binary classifiers.

This method produces very good results in multi-class tasks; nevertheless, it has been pointed out that it does not make use of a crucial quantity in classifiers: the margin. Recently [5], a version was developed that replaces the Hamming-based decoding with a more sophisticated scheme that takes margins into account. Recommendations are also made regarding how to design good codes for margin classifiers, such as SVMs.

7.6.4 Multi-Class Objective Functions

Arguably the most elegant multi-class algorithm, and certainly the method most closely aligned with Vapnik's principle of always trying to solve problems *directly*, entails modifying the SVM objective function in such a way that it simultaneously allows the computation of a multi-class classifier. For instance [593, 58], we can modify (7.35) and use the following quadratic program:

$$\underset{\mathbf{w}_r \in \mathcal{H}, \boldsymbol{\xi}^r \in \mathbb{R}^m, b_r \in \mathbb{R}}{\text{minimize}} \ \frac{1}{2} \sum_{r=1}^{M} \|\mathbf{w}_r\|^2 + \frac{C}{m} \sum_{i=1}^{m} \sum_{r \neq y_i} \xi_i^r, \tag{7.59}$$

$$\text{subject to } \langle \mathbf{w}_{y_i}, \mathbf{x}_i \rangle + b_{y_i} \geq \langle \mathbf{w}_r, \mathbf{x}_i \rangle + b_r + 2 - \xi_i^r,$$

$$\xi_i^r \geq 0, \tag{7.60}$$

where $m \in \{1, \ldots, M\} \setminus y_i$, and $y_i \in \{1, \ldots, M\}$ is the multi-class label of the pattern $\mathbf{x}_i$ (cf. Problem 7.17).

In terms of accuracy, the results obtained with this approach are comparable to those obtained with the widely used one-versus-the-rest approach. Unfortunately, the optimization problem is such that it has to deal with *all* SVs at the same time. In the other approaches, the individual binary classifiers usually have much smaller SV sets, with beneficial effects on the training time. For further multiclass approaches, see [160, 323]. Generalizations to *multi-label* problems, where patterns are allowed to belong to several classes at the same time, are discussed in [162].

Overall, it is fair to say that there is probably no multi-class approach that generally outperforms the others. For practical problems, the choice of approach will depend on constraints at hand. Relevant factors include the required accuracy, the time available for development and training, and the nature of the classification problem (e.g., for a problem with very many classes, it would not be wise to use (7.59)). That said, a simple one-against-the-rest approach often produces acceptable results.

7.7 Variations on a Theme

There are a number of variations of the standard SV classification algorithm, such as the elegant *leave-one-out machine* [589, 592] (see also Section 12.2.2 below), the idea of *Bayes point machines* [451, 239, 453, 545, 392], and extensions to *feature selection* [70, 224, 590]. Due to lack of space, we only describe one of the variations; namely, *linear programming machines*.

Linear Programming Machines

As we have seen above, the SVM approach automatically leads to a decision function of the form (7.25). Let us rewrite it as $f(x) = \operatorname{sgn}(g(x))$, with

$$g(x) = \sum_{i=1}^{m} \upsilon_i k(x, x_i) + b. \tag{7.61}$$

In Chapter 4, we showed that this form of the solution is essentially a consequence of the form of the regularizer $\|\mathbf{w}\|^2$ (Theorem 4.2). The idea of linear programming (LP) machines is to use the kernel expansion as an ansatz for the solution, but to use a different regularizer, namely the ℓ_1 norm of the coefficient vector [343, 344, 74, 184, 352, 37, 591, 593, 39]. The main motivation for this is that this regularizer is known to induce sparse expansions (see Chapter 4).

ℓ_1 Regularizer

This amounts to the objective function

$$R_{\text{reg}}[g] := \frac{1}{m}\|\boldsymbol{\upsilon}\|_1 + C\, R_{\text{emp}}[g], \tag{7.62}$$

where $\|\boldsymbol{\upsilon}\|_1 = \sum_{i=1}^{m} |\upsilon_i|$ denotes the ℓ_1 norm in coefficient space, using the soft margin empirical risk,

$$R_{\text{emp}}[g] = \frac{1}{m} \sum_i \xi_i, \tag{7.63}$$

with slack terms

$$\xi_i = \max\{1 - y_i g(x_i), 0\}. \tag{7.64}$$

We thus obtain a linear programming problem;

$$\begin{aligned} \underset{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b\in\mathbb{R}}{\text{minimize}} \quad & \tfrac{1}{m}\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + C\sum_{i=1}^{m}\xi_i, \\ \text{subject to} \quad & y_i g(x_i) \geq 1 - \xi_i, \\ & \alpha_i, \alpha_i^*, \xi_i \geq 0. \end{aligned} \tag{7.65}$$

Here, we have dealt with the ℓ_1-norm by splitting each component v_i into its positive and negative part: $v_i = \alpha_i - \alpha_i^*$ in (7.61). The solution differs from (7.25) in that it is no longer necessarily the case that each expansion pattern has a weight $\alpha_i y_i$, whose sign equals its class label. This property would have to be enforced separately (Problem 7.19). Moreover, it is also no longer the case that the expansion patterns lie on or beyond the margin — in LP machines, they can basically be anywhere.

ν-LPMs

LP machines can also benefit from the ν-trick. In this case, the programming problem can be shown to take the following form [212]:

$$\begin{aligned} \underset{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b,\rho\in\mathbb{R}}{\text{minimize}} \quad & \tfrac{1}{m}\sum_{i=1}^{m}\xi_i - \nu\rho, \\ \text{subject to} \quad & \tfrac{1}{m}\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) = 1, \\ & y_i g(\mathbf{x}_i) \geq \rho - \xi_i, \\ & \alpha_i, \alpha_i^*, \xi_i, \rho \geq 0. \end{aligned} \tag{7.66}$$

We will not go into further detail at this point. Additional information on linear programming machines from a regularization point of view is given in Section 4.9.2.

7.8 Experiments

7.8.1 Digit Recognition Using Different Kernels

Handwritten digit recognition has long served as a test bed for evaluating and benchmarking classifiers [318, 64, 319]. Thus, it was imperative in the early days of SVM research to evaluate the SV method on widely used digit recognition tasks. In this section we report results on the US Postal Service (USPS) database (described in Section A.1). We shall return to the character recognition problem in Chapter 11, where we consider the larger MNIST database.

As described above, the difference between C-SVC and ν-SVC lies only in the fact that we have to select a different parameter a priori. If we are able to do this

Table 7.3 Performance on the USPS set, for three different types of classifier, constructed with the Support Vector algorithm by choosing different functions k in (7.25) and (7.29). Error rates on the test set are given; and for each of the ten-class-classifiers, we also show the average number of Support Vectors of the ten two-class-classifiers. The normalization factor of 256 is tailored to the dimensionality of the data, which is 16×16.

polynomial: $k(x, x') = \left(\langle x, x' \rangle / 256\right)^d$

d	1	2	3	4	5	6	7
raw error/%	8.9	4.7	4.0	4.2	4.5	4.5	4.7
av. # of SVs	282	237	274	321	374	422	491

RBF: $k(x, x') = \exp\left(-\|x - x'\|^2/(256\, c)\right)$

c	4.0	2.0	1.2	0.8	0.5	0.2	0.1
raw error/%	5.3	5.0	4.9	4.3	4.4	4.4	4.5
av. # of SVs	266	240	233	235	251	366	722

sigmoid: $k(x, x') = \tanh(2\langle x, x' \rangle / 256 + \Theta)$

$-\Theta$	0.8	0.9	1.0	1.1	1.2	1.3	1.4
raw error/%	6.3	4.8	4.1	4.3	4.3	4.4	4.8
av. # of SVs	206	242	254	267	278	289	296

well, we obtain identical performance. The experiments reported were carried out before the development of ν-SVC, and thus all use C-SVC code.

In the present study, we put particular emphasis on comparing different types of SV classifiers obtained by choosing different kernels. We report results for polynomial kernels (7.26), Gaussian radial basis function kernels (7.27), and sigmoid kernels (7.28), summarized in Table 7.3. In all three cases, error rates around 4% can be achieved.

Kernel Scaling

Note that in practical applications, it is usually helpful to scale the argument of the kernel, such that the numerical values do not get extremely small or large as the dimension of the data increases. This helps avoid large roundoff errors, and prevents over- and underflow. In the present case, the scaling was done by including the factor 256 in Table 7.3.

The results show that the Support Vector algorithm allows the construction of a range of learning machines, all of which perform well. The similar performance for the three different functions k suggests that among these cases, the choice of the set of decision functions is less important than capacity control in the chosen type of structure. This phenomenon is well-known for the Parzen window density estimator in $\mathbb{R}^N$ (e.g., [226])

$$p(x) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{\omega^N} k\left(\frac{x - x_i}{\omega}\right). \tag{7.67}$$

It is of great importance in this case to choose an appropriate value of the band-

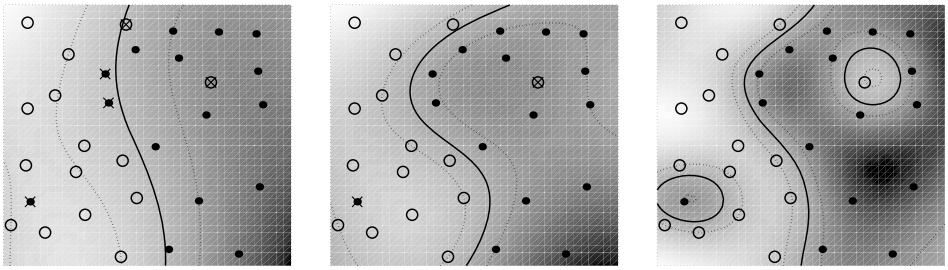


Figure 7.10 2D toy example of a binary classification problem solved using a soft margin SVC. In all cases, a Gaussian kernel (7.27) is used. From left to right, we decrease the kernel width. Note that for a large width, the decision boundary is almost linear, and the data set cannot be separated without error (see text). Solid lines represent decision boundaries; dotted lines depict the edge of the margin (where (7.34) becomes an equality with $\xi_i = 0$).

width parameter ω for a given amount of data. Similar parallels can be drawn to the solution of ill-posed problems; for a discussion, see [561].

Figure 7.10 shows a toy example using a Gaussian kernel (7.27), illustrating that it is crucial to pick the right kernel parameter. In all cases, the same value of C was used, but the kernel width c was varied. For large values of c, the classifier is almost linear, and it cannot separate the data set without errors. For a small width (right), the data set is practically *memorized*. For an intermediate width (middle), a trade-off is made between allowing *some* training errors and using a "simple" decision boundary.

Parameter Choice

In practice, both the kernel parameters and the value of C (or ν) are often chosen using *cross validation*. To this end, we first split the data set into p parts of equal size, say, $p = 10$. We then perform ten training runs. Each time, we leave out one of the ten parts and use it as an independent validation set for optimizing the parameters. In the simplest case, we choose the parameters which work best, on average over the ten runs. It is common practice, however, to then train on the full training set, using these average parameters. There are some problems with this. First, it amounts to optimizing the parameters on the same set as the one used for training, which can lead to overfitting. Second, the optimal parameter settings for data sets of size m and $\frac{9}{10}m$, respectively, do not usually coincide. Typically, the smaller set will require a slightly stronger regularization. This could mean a wider Gaussian kernel, a smaller polynomial degree, a smaller C, or a larger ν. Even worse, it is theoretically possible that there is a so-called phase transition (e.g., [393]) in the learning curve between the two sample sizes. This means that the generalization error as a function of the sample size could change dramatically between $\frac{9}{10}m$ and m. Having said all this, practitioners often do not care about these theoretical precautions, and use the unchanged parameters with excellent results. For further detail, see Section 12.2.

In some cases, one can try to avoid the whole procedure by using an educated guess. Below, we list several methods.

- Use parameter settings that have worked well for similar problems. Here, some care has to be exercised in the scaling of kernel parameters. For instance, when using an RBF kernel, c must be rescaled to ensure that $\|x_i - x_j\|^2/c$ roughly lies in the same range, even if the scaling and dimension of the data are different.
- For many problems, there is some prior expectation regarding the typical error rate. Let us assume we are looking at an image classification task, and we have already tried three other approaches, all of which yielded around 5% test error. Using ν–SV classifiers, we can incorporate this knowledge by choosing a value for ν which is in that range, say $\nu = 5\%$. The reason for this guess is that we know (Proposition 7.5) that the margin error is then below 5%, which in turn implies that the training error is below 5%. The training error will typically be smaller than the test error, thus it is consistent that it should be upper bounded by the 5% test error.
- In a slightly less elegant way, one can try to mimic this procedure for C-SV classifiers. To this end, we start off with a large value of C, and reduce it until the number of Lagrange multipliers that are at the upper bound (in other words, the number of margin errors) is in a suitable range (say, somewhat below 5%). Compared to the above procedure for choosing ν, the disadvantage is that this entails a number of training runs. We can also monitor the number of actual training errors during the training runs, but since not every margin error is a training error, this is often less sensitive. Indeed, the difference between training error and test error can often be quite substantial. For instance, on the USPS set, most of the results reported here were obtained with systems that had essentially zero training error.
- One can put forward scaling arguments which indicate that $C \propto 1/R^2$, where R is a measure for the range of the data in feature space that scales like the length of the points in $\mathcal{H}$. Examples thereof are the standard deviation of the distance of the points to their mean, the radius of the smallest sphere containing the data (cf. (5.61) and (8.17)), or, in some cases, the maximum (or mean) length $k(x_i, x_i)$ over all data points (see Problem 7.25).
- Finally, we can use theoretical tools such as VC bounds (see, for instance, Figure 5.5) or leave-one-out bounds (Section 12.2).

Having seen that different types of SVCs lead to similar performance, the question arises as to how these performances compare with other approaches. Table 7.4 gives a summary of a number of results on the USPS set. Note that the best SVM result is 3.0%; it uses additional techniques that we shall explain in chapters 11 and 13. It is known that the USPS test set is rather difficult — the human error rate is 2.5% [79]. For a discussion, see [496]. Note, moreover, that some of the results reported in the literature for the USPS set were obtained with an enhanced training set: For instance, the study of Drucker et al. [148] used an enlarged training set of size 9709, containing some additional machine-printed digits, and found that this improves the accuracy on the test set. Similarly, Bottou and Vapnik [65] used a training set of size 9840. Since there are no machine-printed digits in the com-

Table 7.4 Summary of error rates on the USPS set. Note that two variants of this database are used in the literature; one of them (denoted by USPS$^+$) is enhanced by a set of machine-printed characters which have been found to improve the test error. Note that the virtual SV systems perform best out of all systems trained on the original USPS set.

Classifier	Training set	Test error	Reference
Linear SVM	USPS	8.9%	[470]
Relevance Vector Machine	USPS	5.1%	Chapter 16
Hard margin SVM	USPS	4.6%	[62]
SVM	USPS	4.0%	[470]
Hyperplane on KPCA features	USPS	4.0%	Chapter 14
KFD	USPS	3.7%	Chapter 15
Virtual SVM	USPS	3.2%	Chapter 11
Virtual SVM, local kernel	USPS	3.0%	Chapter 13
Nearest neighbor	USPS$^+$	5.9%	[496]
LeNet1	USPS$^+$	5.0%	[318]
Local learning Approach	USPS$^+$	3.3%	[65]
Boosted Neural Net	USPS$^+$	2.6%	[148]
Tangent distance	USPS$^+$	2.6%	[496]
Human error rate	—	2.5%	[79]

monly used test set (size 2007), this addition distorts the original learning problem to a situation where results become somewhat hard to interpret. For our experiments, we only had the original 7291 training examples at our disposal. Of all the systems trained on this original set, the SVM system of Chapter 13 performs best.

7.8.2 Universality of the Support Vector Set

In the present section, we report empirical evidence that the SV set contains all the information necessary to solve a given classification task: Using the Support Vector algorithm to train three different types of handwritten digit classifiers, we observe that these types of classifiers construct their decision surface from small, strongly overlapping subsets of the database.

Overlap of SV Sets

To study the Support Vector sets for three different types of SV classifiers, we use the optimal kernel parameters on the USPS set according to Table 7.3. Table 7.5 shows that all three classifiers use around 250 Support Vectors per two-class-classifier (less than 4% of the training set), of which there are 10. The *total* number of different Support Vectors of the ten-class-classifiers is around 1600. It is less than 2500 (10 times the above 250), since for instance a particular vector that has been used as a positive SV (i.e., $y_i = +1$ in (7.25)) for digit 7, might at the same time be a negative SV ($y_i = -1$) for digit 1.

Table 7.6 shows that the SV sets of the different classifiers have about 90% overlap. This surprising result has been reproduced on the MNIST OCR set [467].

Table 7.5 First row: Total number of different SVs in three different ten-class-classifiers (i.e., number of elements of the union of the ten two-class-classifier SV sets), obtained by choosing different functions k in (7.25) and (7.29); Second row: Average number of SVs per two-class-classifier (USPS database size: 7291) (from [470]).

	Polynomial	RBF	Sigmoid
total # of SVs	1677	1498	1611
average # of SVs	274	235	254

Table 7.6 Percentage of the SV set of [column] contained in the SV set of [row]; for ten-class classifiers *(top)*, and binary recognizers for digit class 7 *(bottom)* (USPS set) (from [470]).

	Polynomial	RBF	Sigmoid
Polynomial	100	93	94
RBF	83	100	87
Sigmoid	90	93	100

	Polynomial	RBF	Sigmoid
Polynomial	100	84	93
RBF	89	100	92
Sigmoid	93	86	100

Using a leave-one-out procedure similar to Proposition 7.4, Vapnik and Watkins have put forward a theoretical argument for shared SVs. We state it in the following form: If the SV set of three SV classifiers had no overlap, we could obtain a fourth classifier which has zero test error.

Voting Argument for Shared SVs

To see why this is the case, note that if a pattern is left out of the training set, it will always be classified correctly by voting between the three SV classifiers trained on the remaining examples: Otherwise, it would have been an SV of at least two of them, if kept in the training set. The expectation of the number of patterns which are SVs of at least two of the three classifiers, divided by the training set size, thus forms an upper bound on the expected test error of the voting system. Regarding error rates, it would thus in fact be desirable to be able to construct classifiers with different SV sets. An alternative explanation, studying the effect of the input density on the kernel, was recently proposed by Williams [597]. Finally, we add that the result is also plausible in view of the similar regularization characteristics of the different kernels that were used (see Chapter 4).

Training on SV Sets

As described in Section 7.3, the Support Vector set contains all the information a given classifier needs for constructing the decision function. Due to the overlap in the Support Vector sets of different classifiers, we can even train classifiers on the Support Vector set of *another* classifier; the latter having a different kernel to the former. Table 7.7 shows that this leads to results comparable to those after training

Table 7.7 Training classifiers on the Support Vector sets of other classifiers, leads to performances on the test set (USPS problem) which are as good as the results for training on the full database (number of errors on the 2007-element test set are shown, for two-class classifiers separating digit 7 from the rest). Additionally, the results for training on a random subset of the database of size 200 are displayed.

kernel	trained on: *size:*	poly-SVs *178*	rbf-SVs *189*	tanh-SVs *177*	full db *7291*	rnd. subs. *200*
Poly		13	13	12	13	23
RBF		17	13	17	15	27
tanh		15	13	13	15	25

on the whole database. In Section 11.3, we will use this finding as a motivation for a method to make SVMs transformation invariant, to obtain *virtual SV* machines.

What do these results concerning the nature of Support Vectors tell us? Learning can be viewed as inferring regularities from a set of training examples. Much research has been devoted to the study of various learning algorithms, which allow the extraction of these underlying regularities. No matter how different the outward appearance of these algorithms is, they must all rely on intrinsic regularities of the data. If the learning has been successful, these intrinsic regularities are captured in the values of certain parameters of a learning machine; for a polynomial classifier, these parameters are the coefficients of a polynomial, for a neural network, they are weights, biases, and gains, and for a radial basis function classifier, they are weights, centers, and widths. This variety of different representations of the intrinsic regularities, however, conceals the fact that they all stem from a common root. This is why SVMs with different kernel functions identify the same subset of the training examples as crucial for the regularity to be learned.

7.8.3 Other Applications

SVMs have been successfully applied in other computer vision tasks, which relate to the OCR problems discussed above. Examples include object and face detection and recognition, as well as image retrieval [57, 467, 399, 419, 237, 438, 99, 75].

Another area where SVMs have been used with success is that of *text categorization*. Being a high-dimensional problem, text categorization has been found to be well suited for SVMs. A popular benchmark is the Reuters-22173 text corpus. The news agency Reuters collected 21450 news stories from 1997, and partitioned and indexed them into 135 different categories. The feature typically used to classify Reuters documents are 10^4-dimensional vectors containing word frequencies within a document (sometimes called the "bag-of-words" representation of texts, as it completely discards the information on word ordering). Using this coding, SVMs have led to excellent results, see [155, 265, 267, 150, 333, 542, 149, 326].

Since the use of classification techniques is ubiquitous throughout technology,

we cannot give an exhaustive listing of all successful SVM applications. We thus conclude the list with some of the more exotic applications, such as in High-Energy-Physics [19, 558], in the monitoring of household appliances [390], in protein secondary structure prediction [249], and, with rather intriguing results, in the design of decision feedback equalizers (DFE) in telephony [105].

7.9 Summary

This chapter introduced SV pattern recognition algorithms. The crucial idea is to use kernels to reduce a complex classification task to one that can be solved with separating hyperplanes. We discussed what kind of hyperplane should be constructed in order to get good generalization performance, leading to the idea of large margins. It turns out that the concept of large margins can be justified in a number of different ways, including arguments based on statistical learning theory, and compression schemes. We described in detail how the optimal margin hyperplane can be obtained as the solution of a quadratic programming problem. We started with the linear case, where the hyperplane is constructed in the space of the inputs, and then moved on to the case where we use a kernel function to compute dot products, in order to compute the hyperplane in a feature space.

Two further extensions greatly increase the applicability of the approach. First, to deal with noisy data, we introduced so-called slack variables in the optimization problem. Second, for problems that have more than just two classes, we described a number of generalizations of the binary SV classifiers described initially.

Finally, we reported applications and benchmark comparisons for the widely used USPS handwritten digit task. SVMs turn out to work very well in this field, as well as in a variety of other domains mentioned briefly.

7.10 Problems

7.1 (Weight Vector Scaling •) *Show that instead of the "*1*" on the right hand side of the separation constraint (7.11), we can use any positive number* $\gamma > 0$*, without changing the optimal margin hyperplane solution. What changes in the soft margin case?*

7.2 (Dual Perceptron Algorithm [175] ••) *Kernelize the perceptron algorithm described in footnote 1. Which of the patterns will appear in the expansion of the solution?*

7.3 (Margin of Optimal Margin Hyperplanes [62] ••) *Prove that the geometric margin* ρ *of the optimal margin hyperplane can be computed from the solution* $\boldsymbol{\alpha}$ *via*

$$\rho^{-2} = \sum_{i=1}^{m} \alpha_i. \tag{7.68}$$

Also prove that

$$\rho^{-2} = 2W(\boldsymbol{\alpha}) = ||\mathbf{w}||^2. \tag{7.69}$$

Note that for these relations to hold true, $\boldsymbol{\alpha}$ needs to be the solution of (7.29).

7.4 (Relationship Between $||\mathbf{w}||$ and the Geometrical Margin •) *(i) Consider a separating hyperplane in canonical form. Prove that the margin, measured perpendicularly to the hyperplane, equals $1/||\mathbf{w}||$, by considering two opposite points which precisely satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$.*

(ii) How does the corresponding statement look for the case of ν-SVC? Use the constraint (7.41), and assume that all slack variables are 0.

7.5 (Compression Bound for Large Margin Classification ○○○) *Formalize the ideas stated in Section 7.2: Assuming that the data are separable and lie in a ball of radius R, how many bits are necessary to encode the labels of the data by encoding the parameters of a hyperplane? Formulate a generalization error bound in terms of the compression ratio by using the analysis of Vapnik [561, Section 4.6]. Compare the resulting bound with Theorem 7.3. Take into account the eigenvalues of the Gram matrix, using the ideas of from [604] (cf. Section 12.4).*

7.6 (Positive Definiteness of the SVC Hessian •) *From Definition 2.4, prove that the matrix $Q_{ij} := (y_i y_j k(x_i, x_j))_{ij}$ is positive definite.*

7.7 (Geometric Interpretation of Duality in SVC [38] ••) *Prove that the programming problem (7.10), (7.11) has the same solution as (7.22), provided the threshold b is adjusted such that the hyperplane bisects the shortest connection of the two convex hulls. Hint: Show that the latter is the dual of the former. Interpret the result geometrically.*

7.8 (Number of Points Required to Define a Hyperplane •) *From (7.22), argue that no matter what the dimensionality of the space, there can always be situations where two training points suffice to determine the optimal hyperplane.*

7.9 (In-Bound SVs in Soft Margin SVMs •) *Prove that in-bound SVs lie exactly on the margin. Hint: Use the KKT conditions, and proceed analogously to Section 7.3, where it was shown that in the hard margin case, all SVs lie exactly on the margin.*

Argue, moreover, that bound SVs can lie both on or in the margin, and that they will "usually" lie in the margin.

7.10 (Pattern-Dependent Regularization •) *Derive a version of the soft margin classification algorithm which uses different regularization constants C_i for each training example. Start from (7.35), replace the second term by $\frac{1}{m}\sum_{i=1}^{m} C_i \xi_i$, and derive the dual. Discuss both the mathematical form of the result, and possible applications (cf. [462]).*

7.11 (Uncertain Labels ••) *In this chapter, we have been concerned mainly with the case where the patterns are assigned to one of two classes, i.e., $y \in \{\pm 1\}$. Consider now the*

case where the assignment is not strict, i.e., $y \in [-1, 1]$. Modify the soft margin variants of the SV algorithm, (7.34), (7.35) and (7.41), (7.40), such that

- *whenever $y = 0$, the corresponding pattern has effectively no influence*
- *if all labels are in $\{\pm 1\}$, the original algorithm is recovered*
- *if $|y| < 1$, then the corresponding pattern has less influence than it would have for $|y| = 1$.*

7.12 (SVMs vs. Parzen Windows ○○○) *Develop algorithms that approximate the SVM (soft or hard margin) solution by starting from the Parzen Windows algorithm (Figure 1.1) and sparsifying the expansion of the solution.*

7.13 (Squared Error SVC [111] ●●) *Derive a version of the soft margin classification algorithm which penalizes the errors quadratically. Start from (7.35), replace the second term by $\frac{1}{m}\sum_{i=1}^{m}\xi_i^2$, and derive the dual. Compare the result to the usual C-SVM, both in terms of algorithmic differences and in terms of robustness properties. Which algorithm would you expect to work better for Gaussian-like noise, which one for noise with longer tails (and thus more outliers) (cf. Chapter 3)?*

7.14 (C-SVC with Group Error Penalty ●●) *Suppose the training data are partitioned into ℓ groups,*

$$\begin{array}{c} (\mathbf{x}_1^1, y_1^1), \ldots, (\mathbf{x}_1^{m_1}, y_1^{m_1}) \\ \vdots \qquad\qquad \vdots \\ (\mathbf{x}_\ell^1, y_\ell^1), \ldots, (\mathbf{x}_\ell^{m_\ell}, y_\ell^{m_\ell}), \end{array} \tag{7.70}$$

where $\mathbf{x}_i^j \in \mathcal{H}$ and $y_i^j \in \{\pm 1\}$ (it is understood that the index i runs over $\{1, 2, \ldots, \ell\}$ and the index j runs over $\{1, 2, \ldots, m_i\}$).

Suppose, moreover, that we would like to count a point as misclassified already if one point belonging to the same group is misclassified.

Design an SV algorithm where each group's penalty equals the slack of the worst point in that group.

Hint: Use the objective function

$$\frac{1}{2}\|\mathbf{w}\|^2 + \sum_i C_i \xi_i, \tag{7.71}$$

and the constraints

$$y_i^j \cdot (\left\langle \mathbf{w}, \mathbf{x}_i^j \right\rangle + b) \geq 1 - \xi_i, \tag{7.72}$$

$$\xi_i \geq 0. \tag{7.73}$$

Show that the dual problem consists of maximizing

$$W(\boldsymbol{\alpha}) = \sum_{i,j} \alpha_i^j - \frac{1}{2} \sum_{i,j,i',j'} \alpha_i^j \alpha_{i'}^{j'} y_i^j y_{i'}^{j'} \left\langle \mathbf{x}_i^j, \mathbf{x}_{i'}^{j'} \right\rangle, \tag{7.74}$$

subject to

$$0 = \sum_{i,j} \alpha_i^j y_i^j,\ 0 \le \alpha_i^j,\ \textit{and}\ \sum_j \alpha_i^j \le C_i. \tag{7.75}$$

Argue that typically, only one point per group will become an SV.
Show that C-SVC is a special case of this algorithm.

7.15 (ν-SVC with Group Error Penalty ●●●) *Derive a ν-version of the algorithm in Problem 7.14.*

7.16 (C-SVC vs. ν-SVC ●●) *As a modification of ν-SVC (Section 7.5), compute the dual of $\tau(\mathbf{w}, \boldsymbol{\xi}, \rho) = \|\mathbf{w}\|^2/2 + C(-\nu\rho + (1/m)\sum_{i=1}^m \xi_i)$ (note that in ν-SVC, $C = 1$ is used). Argue that due to the homogeneity of the objective function, the dual solution gets scaled by C, however, the decision function will not change. Hence we may set $C = 1$.*

7.17 (Multi-class vs. Binary SVC [593] ●●) *(i) Prove that the multi-class SVC formulation of (7.59) specializes to the binary C-SVC (7.35) in the case $k = 2$, by using $\mathbf{w}_1 = -\mathbf{w}_2$, $b_1 = -b_2$, and $\xi_i = \frac{1}{2}\xi_i^r$ for pattern $\mathbf{x}_i$ in class r. (ii) Derive the dual of (7.59).*

7.18 (Multi-Class ν-SVC ○○○) *Derive a ν-version of the approach described in Section 7.6.4.*

7.19 (LPM with Constrained Signs ●) *Modify the LPM algorithm such that it is guaranteed that each expansion pattern will have a coefficient v_i whose sign equals the class label y_i. Hint: Do not introduce additional constraints, but eliminate the α_i^* variables and use a different ansatz for the solution.*

7.20 (Multi-Class LPM [593] ●●) *In analogy to Section 7.6.4, develop a multi-class version of the LP machine (Section 7.7).*

7.21 (Version Space [368, 239, 451, 238] ●●●) *Consider hyperplanes passing through the origin, $\{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle = 0\}$, with weight vectors $\mathbf{w} \in \mathcal{H}, \|\mathbf{w}\| = 1$. The set of all such hyperplanes forms a unit sphere in weight space. Each training example $(\mathbf{x}, y) \in \mathcal{H} \times \{\pm 1\}$ splits the sphere into two halves: one that correctly classifies $(\mathbf{x}, y)$, i.e., $\operatorname{sgn} \langle \mathbf{w}, \mathbf{x} \rangle = y$, and one that does not. Each training example thus corresponds to a hemisphere (or, equivalently, an oriented great circle) in weight space, and a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ corresponds to the intersection of m hemispheres, called the* version space.

1. *Discuss how the distances between the training example and the hyperplane in the two representations are related.*
2. *Discuss the relationship to the idea of the Hough transform [255]. The Hough transform is sometimes used in image processing to detect lines. In a nutshell, each point gets to cast votes in support for all potential lines that are consistent with it, and at the end, the lines can be read off the histogram of votes.*
3. *Prove that if all $\mathbf{x}_i$ have the same norm, the maximum margin weight vector corresponds to the center of the largest $m - 1$-dimensional sphere that fits into version space.*

4. *Construct situations where the center of the above largest sphere will generalize poorly, and compare it to the center of mass of version space, called the* Bayes point.

5. *If you disregard the labels of the training examples, there is no longer a single area on the unit sphere which is distinguished from the others due to its corresponding to the correct labelling. Instead, the sphere is split into a number of cells. Argue that the expectation of the natural logarithm of this number equals the VC entropy (Section 5.5.6).*

7.22 (Kernels on Sets ○○○) *Use the construction of Proposition 2.19 to define a kernel that compares two points $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$ by comparing the version spaces (see Problem 7.21) of the labelled examples $(\mathbf{x}, 1)$ and $(\mathbf{x}', 1)$. Define a prior distribution P on the unit sphere in $\mathcal{H}$, and discuss the implications of its choice for the induced kernel. What can you say about the connection between this kernel and the kernel $\langle \mathbf{x}, \mathbf{x}' \rangle$?*

7.23 (Training Algorithms for ν-SVC ○○○) *Try to come up with efficient training algorithms for ν-SVC, building on the material presented in Chapter 10.*

(i) Design a simple chunking algorithm that gradually removes all non-SVs.

(ii) Design a decomposition algorithm.

(iii) Is it possible to modify the SMO algorithm such that it deals with the additional equality constraint that ν-SVC comes with? What is the smallest set of patterns that you can optimize over without violating the two equality constraints? Can you design a generalized SMO algorithm for this case?

7.24 (Prior Class Probabilities ●●) *Suppose that it is known a priori that π_+ and π_- are the probabilities that a pattern belongs to the class ± 1, respectively. Discuss ways of modifying the simple classification algorithm described in Section 1.2 to take this information into account.*

7.25 (Choosing C ●●) *Suppose that R is a measure for the range of the data in feature space that scales like the length of the points in $\mathcal{H}$ (cf. Section 7.8.1). Argue that C should scale like $1/R^2$.*[15] *Hint: consider scaling the data by some $\gamma > 0$. How do you have to scale C such that $f(x) = \langle \mathbf{w}, \Phi(x_j) \rangle + b$ (where $\mathbf{w} = \sum_i \alpha_i y_i \Phi(x_i)$) remains invariant ($j \in [m]$)?*[16] *Discuss measures R that can be used. Why does $R := \max_j k(x_j, x_j)$ not make sense for the Gaussian RBF kernel?*

Moreover, argue that in the asymptotic regime, the upper bound on the α_j should scale with $1/m$, justifying the use of m in (7.38).

7.26 (Choosing C, Part II ○○○) *Problem 7.25 does not take into account the class labels, and hence also not the potential overlap of the two classes. Note that this is different in the ν-approach, which automatically scales the margin with the noise. Can you modify the recommendation in Problem 7.25 to get a selection criterion for C which takes into account the labels, e.g., in the form of prior information on the noise level?*

15. Thanks to Olivier Chapelle for this suggestion.
16. Note that in the ν-parametrization, this scale invariance comes for free.