up a likelihood criterion, one might directly optimize the squared error of the prediction. In other words, if the class label for $\overline{X_k}$ is $y_k \in \{-1, +1\}$, one might simply attempt to optimize the squared error $\sum_{\overline{X_k} \in \mathcal{D}} (y_k - \text{sign}(\theta_0 + \sum_{i=1}^{d} \theta_i x_i^k))^2$ over all test instances. Here, the function "sign" evaluates to $+1$ or -1, depending on whether its argument is positive or negative. As will be evident in Sect. 10.7, such a model is (approximately) used by neural networks. Similarly, Fisher's linear discriminant, which was discussed at the beginning of this chapter, is also a linear least-squares model (cf. Sect. 11.5.1.1 of Chap. 11) but with a different coding of the class variable. In the next section, a linear model that uses the *maximum margin principle* to separate the two classes, will be discussed.

10.6 Support Vector Machines

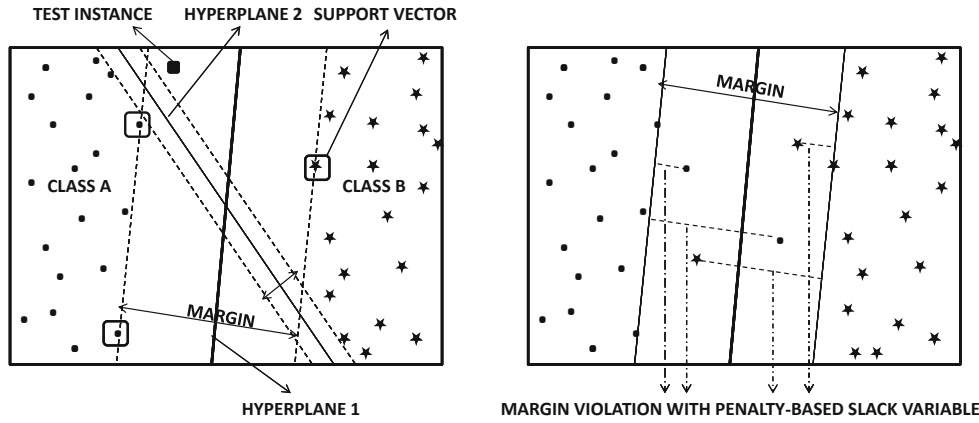
Support vector machines (SVMs) are naturally defined for binary classification of numeric data. The binary-class problem can be generalized to the multiclass case by using a variety of tricks discussed in Sect. 11.2 of Chap. 11. Categorical feature variables can also be addressed by transforming categorical attributes to binary data with the binarization approach discussed in Chap. 2.

It is assumed that the class labels are drawn from $\{-1, 1\}$. As with all linear models, SVMs use separating hyperplanes as the decision boundary between the two classes. In the case of SVMs, the optimization problem of determining these hyperplanes is set up with the notion of *margin*. Intuitively, a *maximum margin hyperplane* is one that cleanly separates the two classes, and for which a large region (or *margin*) exists on each side of the boundary with no training data points in it. To understand this concept, the very special case where the data is *linearly separable* will be discussed first. In linearly separable data, it is possible to construct a linear hyperplane which cleanly separates data points belonging to the two classes. Of course, this special case is relatively unusual because real data is rarely fully separable, and at least a few data points, such as mislabeled data points or outliers, will violate linear separability. Nevertheless, the linearly separable formulation is crucial in understanding the important principle of maximum margin. After discussing the linear separable case, the modifications to the formulation required to enable more general (and realistic) scenarios will be addressed.

10.6.1 Support Vector Machines for Linearly Separable Data

This section will introduce the use of the maximum margin principle in linearly separable data. When the data is linearly separable, there are an infinite number of possible ways of constructing a linear separating hyperplane between the classes. Two examples of such hyperplanes are illustrated in Fig. 10.7a as hyperplane 1 and hyperplane 2. Which of these hyperplanes is better? To understand this, consider the test instance (marked by a square), which is very obviously much closer to class A than class B. The hyperplane 1 will correctly classify it to class A, whereas the hyperplane 2 will incorrectly classify it to class B.

The reason for the varying performance of the two classifiers is that the test instance is placed in a noisy and uncertain boundary region between the two classes, which is not easily *generalizable* from the available training data. In other words, there are few training data points in this uncertain region that are quite like the test instance. In such cases, a separating hyperplane like hyperplane 1, whose minimum perpendicular distance to training points from both classes is as large as possible, is the most robust one for correct classification. This distance can be quantified using the *margin* of the hyperplane.



(a) Hard separation (b) Soft separation

Figure 10.7: Hard and soft SVMs

Consider a hyperplane that cleanly separates two linearly separable classes. The margin of the hyperplane is defined as the sum of its distances to the closest training points belonging to each of the two classes on the opposite side of the hyperplane. A further assumption is that the distance of the separating hyperplane to its closest training point of either class is the same. With respect to the separating hyperplane, it is possible to construct parallel hyperplanes that touch the training data of opposite classes on either side, and have no data point between them. The training data points on these hyperplanes are referred to as the *support vectors*, and the distance between the two hyperplanes is the *margin*. The separating hyperplane, or decision boundary, is precisely in the middle of these two hyperplanes in order to achieve the most accurate classification. The margins for hyperplane 1 and hyperplane 2 are illustrated in Fig. 10.7a by dashed lines. It is evident that the margin for hyperplane 1 is larger than that for hyperplane 2. Therefore, the former hyperplane provides better generalization power for unseen test instances in the "difficult" uncertain region separating the two classes where classification errors are most likely. This is also consistent with our earlier example-based observation about the more accurate classification with hyperplane 1.

How do we determine the maximum margin hyperplane? The way to do this is to set up a nonlinear programming optimization formulation that maximizes the margin by expressing it as a function of the coefficients of the separating hyperplane. The optimal coefficients can be determined by solving this optimization problem. Let the n data points in the training set $\mathcal{D}$ be denoted by $(\overline{X_1}, y_1) \ldots (\overline{X_n}, y_n)$, where $\overline{X_i}$ is a d-dimensional row vector corresponding to the ith data point, and $y_i \in \{-1, +1\}$ is the binary class variable of the ith data point. Then, the separating hyperplane is of the following form:

$$\overline{W} \cdot \overline{X} + b = 0. \tag{10.35}$$

Here, $\overline{W} = (w_1 \ldots w_d)$ is the d-dimensional row vector representing the normal direction to the hyperplane, and b is a scalar, also known as the *bias*. The vector $\overline{W}$ regulates the orientation of the hyperplane and the bias b regulates the distance of the hyperplane from the origin. The $(d + 1)$ coefficients corresponding to $\overline{W}$ and b need to be learned from the training data to maximize the margin of separation between the two classes. Because it

is assumed that the classes are linearly separable, such a hyperplane can also be assumed to exist. All data points $\overline{X_i}$ with $y_i = +1$ will lie on one side of the hyperplane satisfying $\overline{W} \cdot \overline{X_i} + b \geq 0$. Similarly, all points with $y_i = -1$ will lie on the other side of the hyperplane satisfying $\overline{W} \cdot \overline{X_i} + b \leq 0$.

$$\overline{W} \cdot \overline{X_i} + b \geq 0 \quad \forall i : y_i = +1 \tag{10.36}$$

$$\overline{W} \cdot \overline{X_i} + b \leq 0 \quad \forall i : y_i = -1 \tag{10.37}$$

These constraints do not yet incorporate the margin requirements on the data points. A stronger set of constraints are defined using these margin requirements. It may be assumed that the separating hyperplane $\overline{W} \cdot \overline{X} + b = 0$ is located in the center of the two margin-defining hyperplanes. Therefore, the two symmetric hyperplanes touching the support vectors can be expressed by introducing another parameter c that regulates the distance between them.

$$\overline{W} \cdot \overline{X} + b = +c \tag{10.38}$$

$$\overline{W} \cdot \overline{X} + b = -c \tag{10.39}$$

It is possible to assume, without loss of generality, that the variables $\overline{W}$ and b are appropriately scaled, so that the value of c can be set to 1. Therefore, the two separating hyperplanes can be expressed in the following form:

$$\overline{W} \cdot \overline{X} + b = +1 \tag{10.40}$$

$$\overline{W} \cdot \overline{X} + b = -1. \tag{10.41}$$

These constraints are referred to as *margin constraints*. The two hyperplanes segment the data space into three regions. It is assumed that no training data points lie in the uncertain decision boundary region between these two hyperplanes, and all training data points for each class are mapped to one of the two remaining (extreme) regions. This can be expressed as pointwise constraints on the training data points as follows:

$$\overline{W} \cdot \overline{X_i} + b \geq +1 \quad \forall i : y_i = +1 \tag{10.42}$$

$$\overline{W} \cdot \overline{X_i} + b \leq -1 \quad \forall i : y_i = -1. \tag{10.43}$$

Note that the constraints for both the positive and negative classes can be written in the following succinct and algebraically convenient, but rather cryptic, form:

$$y_i(\overline{W} \cdot \overline{X_i} + b) \geq +1 \quad \forall i. \tag{10.44}$$

The distance between the two hyperplanes for the positive and negative instances is also referred to as the margin. As discussed earlier, the goal is to maximize this margin. What is the distance (or margin) between these two parallel hyperplanes? One can use linear algebra to show that the distance between two parallel hyperplanes is the normalized difference between their constant terms, where the normalization factor is the L_2-norm $||\overline{W}|| = \sqrt{\sum_{i=1}^{d} w_i^2}$ of the coefficients. Because the difference between the constant terms of the two aforementioned hyperplanes is 2, it follows that the distance between them is $2/||\overline{W}||$. This is the margin that needs to be maximized with respect to the aforementioned constraints. This form of the objective function is inconvenient because it incorporates a

square root in the denominator of the objective function. However, maximizing $2/||\overline{W}||$ is the same as minimizing $||\overline{W}||^2/2$. This is a convex quadratic programming problem, because the quadratic objective function $||\overline{W}||^2/2$ needs to be minimized subject to a set of linear constraints (Eqs. 10.42–10.43) on the training points. Note that each training data point leads to a constraint, which tends to make the optimization problem rather large, and explains the high computational complexity of SVMs.

Such constrained nonlinear programming problems are solved using a method known as *Lagrangian relaxation*. The broad idea is to associate a nonnegative n-dimensional set of Lagrangian multipliers $\overline{\lambda} = (\lambda_1 \ldots \lambda_n) \geq 0$ for the different constraints. The multiplier λ_i corresponds to the margin constraint of the ith training data point. The constraints are then relaxed, and the objective function is augmented by incorporating a Lagrangian penalty for constraint violation:

$$L_P = \frac{||\overline{W}||^2}{2} - \sum_{i=1}^{n} \lambda_i \left[y_i(\overline{W} \cdot \overline{X_i} + b) - 1 \right]. \tag{10.45}$$

For *fixed* nonnegative values of λ_i, margin constraint violations increase L_p. Therefore, the penalty term pushes the optimized values of $\overline{W}$ and b towards constraint nonviolation for minimization of L_P with respect to $\overline{W}$ and b. Values of $\overline{W}$ and b that satisfy the margin constraints will always result in a nonpositive penalty. Therefore, for any fixed nonnegative value of $\overline{\lambda}$, the minimum value of L_P will always be at most equal to that of the original optimal objective function value $||\overline{W^*}||^2/2$ because of the impact of the non-positive penalty term for any feasible $(\overline{W^*}, b^*)$.

Therefore, if L_P is minimized with respect to $\overline{W}$ and b for any particular $\overline{\lambda}$, and then maximized with respect to nonnegative Lagrangian multipliers $\overline{\lambda}$, the resulting *dual* solution L_D^* will be a lower bound on the optimal objective function $O^* = ||\overline{W^*}||^2/2$ of the SVM formulation. Mathematically, this *weak duality* condition can be expressed as follows:

$$O^* \geq L_D^* = \max_{\overline{\lambda} \geq 0} \min_{\overline{W}, b} L_P. \tag{10.46}$$

Optimization formulations such as SVM are special because the objective function is convex, and the constraints are linear. Such formulations satisfy a property known as *strong duality*. According to this property, the minimax relationship of Eq. 10.46 yields an optimal and feasible solution to the original problem (i.e., $O^* = L_D^*$) in which the Lagrangian penalty term has zero contribution. Such a solution $(\overline{W^*}, b^*, \overline{\lambda^*})$ is referred to as the *saddle point* of the Lagrangian formulation. Note that zero Lagrangian penalty is achieved by a feasible solution only when each training data point $\overline{X_i}$ satisfies $\lambda_i \left[y_i(\overline{W} \cdot \overline{X_i} + b) - 1 \right] = 0$. These conditions are equivalent to the *Kuhn–Tucker optimality conditions*, and they imply that data points $\overline{X_i}$ with $\lambda_i > 0$ are support vectors. The Lagrangian formulation is solved using the following steps:

1. The Lagrangian objective L_P can be expressed more conveniently as a pure maximization problem by eliminating the minimization part from the awkward minimax formulation. This is achieved by eliminating the minimization variables $\overline{W}$ and b with gradient-based optimization conditions on these variables. By setting the gradient of L_P with respect to $\overline{W}$ to 0, we obtain the following:

$$\nabla L_P = \nabla \frac{||\overline{W}||^2}{2} - \nabla \sum_{i=1}^{n} \lambda_i \left[y_i(\overline{W} \cdot \overline{X_i} + b) - 1 \right] = 0 \tag{10.47}$$

$$\overline{W} - \sum_{i=1}^{n} \lambda_i y_i \overline{X_i} = 0. \tag{10.48}$$

Therefore, one can now derive an expression for $\overline{W}$ in terms of the Lagrangian multipliers and the training data points:

$$\overline{W} = \sum_{i=1}^{n} \lambda_i y_i \overline{X_i}. \tag{10.49}$$

Furthermore, by setting the partial derivative of L_P with respect to b to 0, we obtain $\sum_{i=1}^{n} \lambda_i y_i = 0$.

2. The optimization condition $\sum_{i=1}^{n} \lambda_i y_i = 0$ can be used to eliminate the term $-b\sum_{i=1}^{n} \lambda_i y_i$ from L_P. The expression $\overline{W} = \sum_{i=1}^{n} \lambda_i y_i \overline{X_i}$ from Eq. 10.49 can then be substituted in L_P to create a dual problem L_D in terms of only the *maximization* variables $\overline{\lambda}$. Specifically, the maximization objective function L_D for the Lagrangian dual is as follows:

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \overline{X_i} \cdot \overline{X_j}. \tag{10.50}$$

The dual problem maximizes L_D subject to the constraints $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$. Note that L_D is expressed only in terms of λ_i, the class labels, and the pairwise dot products $\overline{X_i} \cdot \overline{X_j}$ between training data points. Therefore, solving for the Lagrangian multipliers requires knowledge of only the class variables and dot products between training instances but it does not require *direct* knowledge of the feature values $\overline{X_i}$. The dot products between training data points can be viewed as a kind of similarity between the points, which can easily be defined for data types beyond numeric domains. This observation is important for generalizing linear SVMs to nonlinear decision boundaries and arbitrary data types with the kernel trick.

3. The value of b can be derived from the constraints in the original SVM formulation, for which the Lagrangian multipliers λ_r are *strictly* positive. For these training points, the margin constraint $y_r(\overline{W} \cdot \overline{X_r} + b) = +1$ is satisfied exactly according to the Kuhn–Tucker conditions. The value of b can be derived from *any* such training point $(\overline{X_r}, y_r)$ as follows:

$$y_r \left[\overline{W} \cdot \overline{X_r} + b\right] = +1 \qquad \forall r : \lambda_r > 0 \tag{10.51}$$

$$y_r \left[(\sum_{i=1}^{n} \lambda_i y_i \overline{X_i} \cdot \overline{X_r}) + b\right] = +1 \qquad \forall r : \lambda_r > 0. \tag{10.52}$$

The second relationship is derived by substituting the expression for $\overline{W}$ in terms of the Lagrangian multipliers according to Eq. 10.49. Note that this relationship is expressed only in terms of Lagrangian multipliers, class labels, and dot products between training instances. The value of b can be solved from this equation. To reduce numerical error, the value of b may be averaged over all the support vectors with $\lambda_r > 0$.

4. For a test instance $\overline{Z}$, its class label $F(\overline{Z})$ is defined by the decision boundary obtained by substituting for $\overline{W}$ in terms of the Lagrangian multipliers (Eq. 10.49):

$$F(\overline{Z}) = \text{sign}\{\overline{W} \cdot \overline{Z} + b\} = \text{sign}\{(\sum_{i=1}^{n} \lambda_i y_i \overline{X_i} \cdot \overline{Z}) + b\}. \tag{10.53}$$

It is interesting to note that $F(\overline{Z})$ can be fully expressed in terms of the dot product between training instances and test instances, class labels, Lagrangian multipliers, and bias b. Because the Lagrangian multipliers λ_i and b can also be expressed in terms of the dot products between training instances, it follows that the classification can be fully performed using knowledge of only the dot product between different instances (training and test), without knowing the exact feature values of either the training or the test instances.

The observations about dot products are crucial in generalizing SVM methods to nonlinear decision boundaries and arbitrary data types with the use of a technique known as the *kernel trick*. This technique simply substitutes dot products with kernel similarities (cf. Sect. 10.6.4).

It is noteworthy from the derivation of $\overline{W}$ (see Eq. 10.49) and the aforementioned derivation of b, that only training data points that are support vectors (with $\lambda_r > 0$) are used to define the solution $\overline{W}$ and b in SVM optimization. As discussed in Chap. 11, this observation is leveraged by scalable SVM classifiers, such as *SVMLight*. Such classifiers shrink the size of the problem by discarding irrelevant training data points that are easily identified to be far away from the separating hyperplanes.

10.6.1.1 Solving the Lagrangian Dual

The Lagrangian dual L_D may be optimized by using the gradient ascent technique in terms of the n-dimensional parameter vector $\overline{\lambda}$.

$$\frac{\partial L_D}{\partial \lambda_i} = 1 - y_i \sum_{j=1}^{n} y_j \lambda_j \overline{X_i} \cdot \overline{X_j} \tag{10.54}$$

Therefore, as in logistic regression, the corresponding gradient-based update equation is as follows:

$$(\lambda_1 \ldots \lambda_n) \leftarrow (\lambda_1 \ldots \lambda_n) + \alpha \left(\frac{\partial L_D}{\partial \lambda_1} \ldots \frac{\partial L_D}{\partial \lambda_n} \right). \tag{10.55}$$

The step size α may be chosen to maximize the improvement in objective function. The initial solution can be chosen to be the vector of zeros, which is also a feasible solution for $\overline{\lambda}$.

One problem with this update is that the constraints $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$ may be violated after an update. Therefore, the gradient vector is projected along the hyperplane $\sum_{i=1}^{n} \lambda_i y_i = 0$ before the update to create a modified gradient vector. Note that the projection of the gradient ∇L_D along the normal to this hyperplane is simply $\overline{H} = (\overline{y} \cdot \nabla L_D)\, \overline{y}$, where $\overline{y}$ is the unit vector $\frac{1}{\sqrt{n}}(y_1 \ldots y_n)$. This component is subtracted from ∇L_D to create a modified gradient vector $\overline{G} = \nabla L_D - \overline{H}$. Because of the projection, updating along the modified gradient vector $\overline{G}$ will not violate the constraint $\sum_{i=1}^{n} \lambda_i y_i = 0$. In addition, any negative values of λ_i after an update are reset to 0.

Note that the constraint $\sum_{i=1}^{n} \lambda_i y_i = 0$ is derived by setting the gradient of L_P with respect to b to 0. In some alternative formulations of SVMs, the bias vector b can be included within $\overline{W}$ by adding a synthetic dimension to the data with a constant value of 1. In such cases, the gradient vector update is simplified to Eq. 10.55 because one no longer needs to worry about the constraint $\sum_{i=1}^{n} \lambda_i y_i = 0$. This alternative formulation of SVMs is discussed in Chap. 13.

10.6.2 Support Vector Machines with Soft Margin for Nonseparable Data

The previous section discussed the scenario where the data points of the two classes are linearly separable. However, perfect linear separability is a rather contrived scenario, and real data sets usually will not satisfy this property. An example of such a data set is illustrated in Fig. 10.7b, where no linear separator may be found. Many real data sets may, however, be approximately separable, where *most* of the data points lie on correct sides of well-chosen separating hyperplanes. In this case, the notion of margin becomes a softer one because training data points are allowed to violate the margin constraints *at the expense of a penalty*. The two margin hyperplanes separate out "most" of the training data points but not all of them. An example is illustrated in Fig. 10.7b.

The level of violation of each margin constraint by training data point $\overline{X_i}$ is denoted by a slack variable $\xi_i \geq 0$. Therefore, the new set of soft constraints on the separating hyperplanes may be expressed as follows:

$$\begin{aligned} \overline{W} \cdot \overline{X_i} + b &\geq +1 - \xi_i \quad \forall i : y_i = +1 \\ \overline{W} \cdot \overline{X_i} + b &\leq -1 + \xi_i \quad \forall i : y_i = -1 \\ \xi_i &\geq 0 \quad \forall i. \end{aligned}$$

These slack variables ξ_i may be interpreted as the distances of the training data points from the separating hyperplanes, as illustrated in Fig. 10.7b, when they lie on the "wrong" side of the separating hyperplanes. The values of the slack variables are 0 when they lie on the correct side of the separating hyperplanes. It is not desirable for too many training data points to have positive values of ξ_i, and therefore such violations are penalized by $C \cdot \xi_i^r$, where C and r are user-defined parameters regulating the level of softness in the model. Small values of C would result in relaxed margins, whereas large values of C would minimize training data errors and result in narrow margins. Setting C to be sufficiently large would disallow any training data error in separable classes, which is the same as setting all slack variables to 0 and defaulting to the hard version of the problem. A popular choice of r is 1, which is also referred to as *hinge loss*. Therefore, the objective function for soft-margin SVMs, with hinge loss, is defined as follows:

$$O = \frac{||\overline{W}||^2}{2} + C \sum_{i=1}^{n} \xi_i. \tag{10.56}$$

As before, this is a convex quadratic optimization problem that can be solved using Lagrangian methods. A similar approach is used to set up the Lagrangian relaxation of the problem with penalty terms and additional multipliers $\beta_i \geq 0$ for the slack constraints $\xi_i \geq 0$:

$$L_P = \frac{||\overline{W}||^2}{2} + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \lambda_i \left[y_i (\overline{W} \cdot \overline{X_i} + b) - 1 + \xi_i \right] - \sum_{i=1}^{n} \beta_i \xi_i. \tag{10.57}$$

A similar approach to the hard SVM case can be used to eliminate the minimization variables $\overline{W}$, ξ_i, and b from the optimization formulation and create a purely maximization dual formulation. This is achieved by setting the gradient of L_P with respect to these variables to 0. By setting the gradients of L_P with respect to $\overline{W}$ and b to 0, it can be respectively shown that the value of $\overline{W}$ is identical to the hard-margin case (Eq. 10.49), and the same

multiplier constraint $\sum_{i=1}^{n} \lambda_i y_i = 0$ is satisfied. This is because the additional slack terms in L_P involving ξ_i do not affect the respective gradients with respect to $\overline{W}$ and b. Furthermore, it can be shown that the objective function L_D of the Lagrangian dual in the soft-margin case is identical to that of the hard-margin case, according to Eq. 10.50, because the linear terms involving each ξ_i evaluate[5] to 0. The *only* change to the dual optimization problem is that the nonnegative Lagrangian multipliers satisfy additional constraints of the form $C - \lambda_i = \beta_i \geq 0$. This constraint is derived by setting the partial derivative of L_P with respect to ξ_i to 0. One way of viewing this additional constraint $\lambda_i \leq C$ is that the influence of any training data point $\overline{X_i}$ on the weight vector $\overline{W} = \sum_{i=1}^{n} \lambda_i y_i \overline{X_i}$ is capped by C because of the softness of the margin. *The dual problem in soft SVMs maximizes L_D (Eq. 10.50) subject to the constraints $0 \leq \lambda_i \leq C$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$.*

The Kuhn–Tucker optimality conditions for the slack nonnegativity constraints are $\beta_i \xi_i = 0$. Because we have already derived $\beta_i = C - \lambda_i$, we obtain $(C - \lambda_i)\xi_i = 0$. In other words, training points $\overline{X_i}$ with $\lambda_i < C$ correspond to zero slack ξ_i and they might either lie on the margin or on the correct side of the margin. However, in this case, the support vectors are defined as data points that satisfy the *soft* SVM constraints exactly and some of them might have nonzero slack. Such points might lie on the margin, between the margin, or on the wrong side of the decision boundary. Points that satisfy $\lambda_i > 0$ are always support vectors. The support vectors that lie on the margin will therefore satisfy $0 < \lambda_i < C$. These points are very useful in solving for b. Consider any such support vector $\overline{X_r}$ with zero slack, which satisfies $0 < \lambda_r < C$. The value of b may be obtained as before:

$$y_r \left[(\sum_{i=1}^{n} \lambda_i y_i \overline{X_i} \cdot \overline{X_r}) + b \right] = +1. \tag{10.58}$$

Note that this expression is the same as for the case of hard SVMs, except that the relevant training points are identified by using the condition $0 < \lambda_r < C$. The gradient-ascent update is also identical to the separable case (cf. Sect. 10.6.1.1), except that any multiplier λ_i exceeding C because of an update needs to be reset to C. The classification of a test instance also uses Eq. 10.53 in terms of Lagrangian multipliers because the relationship between the weight vector and the Lagrangian multipliers is the same in this case. Thus, the soft SVM formulation with hinge loss is strikingly similar to the hard SVM formulation. This similarity is less pronounced for other slack penalty functions such as quadratic loss.

The soft version of SVMs also allows an *unconstrained* primal formulation by eliminating the margin constraints and slack variables simultaneously. This is achieved by substituting $\xi_i = \max\{0, 1 - y_i[\overline{W} \cdot \overline{X_i} + b]\}$ in the primal objective function of Eq. 10.56. This results in an unconstrained optimization (minimization) problem purely in terms of $\overline{W}$ and b:

$$O = \frac{||\overline{W}||^2}{2} + C \sum_{i=1}^{n} \max\{0, 1 - y_i[\overline{W} \cdot \overline{X_i} + b]\}. \tag{10.59}$$

One can use a gradient descent approach, which is analogous to the gradient ascent method used in logistic regression. The partial derivatives of nondifferentiable function O with respect to $w_1, \ldots w_d$ and b are approximated on a casewise basis, depending on whether or not the term inside the maximum function evaluates to a positive quantity. The precise derivation of the gradient descent steps is left as an exercise for the reader. While the dual

[5]The additional term in L_P involving ξ_i is $(C - \beta_i - \lambda_i)\xi_i$. This term evaluates to 0 because the partial derivative of L_P with respect to ξ_i is $(C - \beta_i - \lambda_i)$. This partial derivative must evaluate to 0 for optimality of L_P.

approach is more popular, the primal approach is intuitively simpler, and it is often more efficient when an approximate solution is desired.

10.6.2.1 Comparison with Other Linear Models

The normal vector to a linear separating hyperplane can be viewed as a direction along which the data points of the two classes are best separated. Fisher's linear discriminant also achieves this goal by maximizing the ratio of the between-class scatter to the within-class scatter along an optimally chosen vector. However, an important distinguishing feature of SVMs is that they focus extensively on the *decision boundary* region between the two classes because this is the most uncertain region, which is prone to classification error. Fisher's discriminant focuses on the global separation between the two classes and may not necessarily provide the best separation in the uncertain boundary region. This is the reason that SVMs often have better generalization behavior for noisy data sets that are prone to overfitting.

It is instructive to express logistic regression as a minimization problem by using the negative of the log-likelihood function and then comparing it with SVMs. The coefficients $(\theta_0, \ldots \theta_d)$ in logistic regression are analogous to the coefficients $(b, \overline{W})$ in SVMs. SVMs have a margin component to increase the generalization power of the classifier, just as logistic regression uses regularization. Interestingly, the margin component $||\overline{W}||^2/2$ in SVMs has an identical form to the regularization term $\sum_{i=1}^{d} \theta_i^2/2$ in logistic regression. SVMs have slack penalties just as logistic regression implicitly penalizes the *probability of* mistakes in the log-likelihood function. However, the slack is computed using *margin violations* in SVMs, whereas the penalties in logistic regression are computed as a smooth function of the distances from the *decision boundary*. Specifically, the log-likelihood function in logistic regression creates a smooth loss function of the form $log(1 + e^{-y_i[\theta_0 + \overline{\theta} \cdot \overline{X_i}]})$, whereas the hinge loss $max\{0, 1 - y_i[\overline{W} \cdot \overline{X_i} + b]\}$ in SVMs is not a smooth function. The nature of the misclassification penalty is the only difference between the two models. Therefore, there are several conceptual similarities among these models, but they emphasize different aspects of optimization. SVMs and regularized logistic regression show similar performance in many practical settings with poorly separable classes. However, SVMs and Fisher's discriminant generally perform better than logistic regression for the special case of well-separated classes. All these methods can also be extended to nonlinear decision boundaries in similar ways.

10.6.3 Nonlinear Support Vector Machines

In many cases, linear solvers are not appropriate for problems in which the decision boundary is not linear. To understand this point, consider the data distribution illustrated in Fig. 10.8. It is evident that no linear separating hyperplanes can delineate the two classes. This is because the two classes are separated by the following decision boundary:

$$8(x_1 - 1)^2 + 50(x_2 - 2)^2 = 1. \tag{10.60}$$

Now, if one already had some insight about the nature of the decision boundary, one might transform the training data into the new 4-dimensional space as follows:

$$z_1 = x_1^2$$
$$z_2 = x_1$$
$$z_3 = x_2^2$$
$$z_4 = x_2.$$

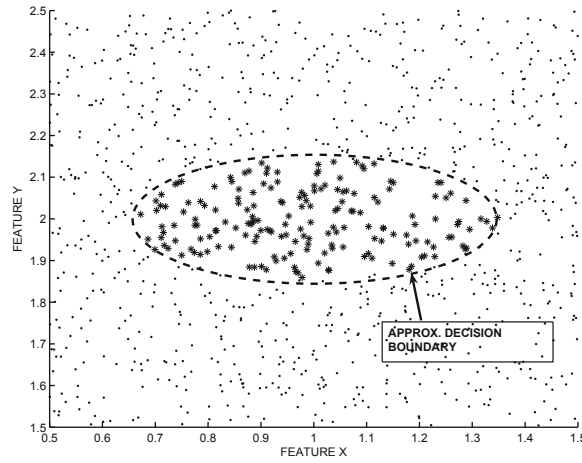


Figure 10.8: Nonlinear decision surface

The decision boundary of Eq. 10.60 can be expressed linearly in terms of the variables $z_1 \ldots z_4$, by expanding Eq. 10.60 in terms of x_1, x_1^2, x_2, and x_2^2:

$$8x_1^2 - 16x_1 + 50x_2^2 - 200x_2 + 207 = 0$$
$$8z_1 - 16z_2 + 50z_3 - 200z_4 + 207 = 0.$$

Thus, each training data point is now expressed in terms of these four newly transformed dimensions, and the classes will be linearly separable in this space. The SVM optimization formulation can then be solved in the transformed space as a linear model, and used to classify test instances that are also transformed to 4-dimensional space. It is important to note that the complexity of the problem effectively increased because of the increase in the size of the hyperplane coefficient vector $\overline{W}$.

In general, it is possible to approximate any polynomial decision boundary by adding an additional set of dimensions for each exponent of the polynomial. High-degree polynomials have significant expressive power in approximating many nonlinear functions well. This kind of transformation can be very effective in cases where one does not know whether the decision boundary is linear or nonlinear. This is because the additional degrees of freedom in the model, in terms of the greater number of coefficients to be learned, can determine the linearity or nonlinearity of the decision boundary in a data-driven way. In our previous example, if the decision boundary had been linear, the coefficients for z_1 and z_3 would automatically have been learned to be almost 0, given enough training data. The price for this additional flexibility is the increased computational complexity of the training problem, and the larger number of coefficients that need to be learned. Furthermore, if enough training data is not available, then this may result in overfitting where even a simple linear decision boundary is incorrectly approximated as a nonlinear one. A different approach, which is sometimes used to learn nonlinear decision boundaries, is known as the "kernel trick." This approach is able to learn arbitrary decision boundaries without performing the transformation explicitly.

10.6.4 The Kernel Trick

The kernel trick leverages the important observation that the SVM formulation can be fully solved in terms of dot products (or similarities) between pairs of data points. One does not need to know the feature values. Therefore, the key is to define the pairwise dot product (or similarity function) directly in the d'-dimensional transformed representation $\Phi(\overline{X})$, with the use of a kernel function $K(\overline{X_i}, \overline{X_j})$.

$$K(\overline{X_i}, \overline{X_j}) = \Phi(\overline{X_i}) \cdot \Phi(\overline{X_j}) \tag{10.61}$$

To effectively solve the SVM, recall that the transformed feature values $\Phi(\overline{X})$ need not be explicitly computed, as long as the dot product (or kernel similarity) $K(\overline{X_i}, \overline{X_j})$ is known. This implies that the term $\overline{X_i} \cdot \overline{X_j}$ may be replaced by the *transformed-space* dot product $K(\overline{X_i}, \overline{X_j})$ in Eq. 10.50, and the term $\overline{X_i} \cdot \overline{Z}$ in Eq. 10.53 can be replaced by $K(\overline{X_i}, \overline{Z})$ to perform SVM classification.

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(\overline{X_i}, \overline{X_j}) \tag{10.62}$$

$$F(\overline{Z}) = \text{sign}\{(\sum_{i=1}^{n} \lambda_i y_i K(\overline{X_i}, \overline{Z})) + b\} \tag{10.63}$$

Note that the bias b is also expressed in terms of dot products according to Eq. 10.58. These modifications are carried over to the update equations discussed in Sect. 10.6.1.1, all of which are expressed in terms of dot products.

Thus, all computations are performed in the *original* space, and the actual transformation $\Phi(\cdot)$ does not need to be known as long as the kernel similarity function $K(\cdot, \cdot)$ is known. By using kernel-based similarity with carefully chosen kernels, arbitrary nonlinear decision boundaries can be approximated. There are different ways of modeling similarity between $\overline{X_i}$ and $\overline{X_j}$. Some common choices of the kernel function are as follows:

Function	Form
Gaussian radial basis kernel	$K(\overline{X_i}, \overline{X_j}) = e^{-\|\overline{X_i} - \overline{X_j}\|^2/2\sigma^2}$
Polynomial kernel	$K(\overline{X_i}, \overline{X_j}) = (\overline{X_i} \cdot \overline{X_j} + c)^h$
Sigmoid kernel	$K(\overline{X_i}, \overline{X_j}) = \tanh(\kappa \overline{X_i} \cdot \overline{X_j} - \delta)$

Many of these kernel functions have parameters associated with them. In general, these parameters may need to be tuned by holding out a portion of the training data, and using it to test the accuracy of different choices of parameters. Many other kernels are possible beyond the ones listed in the table above. Kernels need to satisfy a property known as *Mercer's theorem* to be considered valid. This condition ensures that the $n \times n$ kernel similarity matrix $S = [K(\overline{X_i}, \overline{X_j})]$ is positive semidefinite, and similarities can be expressed as dot products in some transformed space. Why must the kernel similarity matrix always be positive semidefinite for similarities to be expressed as dot products? Note that if the $n \times n$ kernel similarity matrix S can be expressed as the $n \times n$ dot-product matrix AA^T of some $n \times r$ transformed representation A of the points, then for any n-dimensional column vector $\overline{V}$, we have $\overline{V}^T S \overline{V} = (A\overline{V})^T (A\overline{V}) \geq 0$. In other words, S is positive semidefinite. Conversely, if the kernel matrix S is positive semi-definite then it can be expressed as a dot product

with the eigen decomposition $S = Q\Sigma^2 Q^T = (Q\Sigma)(Q\Sigma)^T$, where Σ^2 is an $n \times n$ diagonal matrix of nonnegative eigenvalues and Q is an $n \times n$ matrix containing the eigenvectors of S in columns. The matrix $Q\Sigma$ is the n-dimensional transformed representation of the points, and it also sometimes referred to as the *data-specific Mercer kernel map*. This map is data set-specific, and it is used in many nonlinear dimensionality reduction methods such as kernel *PCA*.

What kind of kernel function works best for the example of Fig. 10.8? In general, there are no predefined rules for selecting kernels. Ideally, if the similarity values $K(\overline{X_i}, \overline{X_j})$ were defined so that a space exists, in which points with this similarity structure are linearly separable, then a linear SVM in the transformed space $\Phi(\cdot)$ will work well.

To explain this point, we will revisit the example of Fig. 10.8. Let $\overline{X2_i}$ and $\overline{X2_j}$ be the d-dimensional vectors derived by squaring each coordinate of $\overline{X_i}$ and $\overline{X_j}$, respectively. In the case of Fig. 10.8, consider the transformation (z_1, z_2, z_3, z_4) in the previous section. It can be shown that the dot product between two transformed data points can be captured by the following kernel function:

$$\textit{Transformed-Dot-Product}(\overline{X_i}, \overline{X_j}) = \overline{X_i} \cdot \overline{X_j} + \overline{X2_i} \cdot \overline{X2_j}. \tag{10.64}$$

This is easy to verify by expanding the aforementioned expression in terms of the transformed variables $z_1 \ldots z_4$ of the two data points. The kernel function *Transformed-Dot-Product*$(\overline{X_i}, \overline{X_j})$ would obtain the same Lagrangian multipliers and decision boundary as obtained with the explicit transformation $z_1 \ldots z_4$. Interestingly, this kernel is closely related to the second-order polynomial kernel.

$$K(\overline{X_i}, \overline{X_j}) = (0.5 + \overline{X_i} \cdot \overline{X_j})^2 \tag{10.65}$$

Expanding the second-order polynomial kernel results in a superset of the additive terms in *Transformed-Dot-Product*$(\overline{X_i}, \overline{X_j})$. The additional terms include a constant term of 0.25 and some inter-dimensional products. These terms provide further modeling flexibility. In the case of the 2-dimensional example of Fig. 10.8, the use of the second-order polynomial kernel is equivalent to using an extra transformed variable $z_5 = \sqrt{2}x_1x_2$ representing the product of the values on the two dimensions and a constant dimension $z_6 = 0.5$. These variables are in addition to the original four variables (z_1, z_2, z_3, z_4). Since these additional variables are redundant in this case, they will not affect the ability to discover the correct decision boundary, although they might cause some overfitting. On the other hand, a variable such as $z_5 = \sqrt{2}x_1x_2$ would have come in handy, if the ellipse of Fig. 10.8 had been arbitrarily oriented with respect to the axis system. A full separation of the classes would not have been possible with a linear classifier on the original four variables (z_1, z_2, z_3, z_4). Therefore, the second-order polynomial kernel can discover more general decision boundaries than the transformation of the previous section. Using even higher-order polynomial kernels can model increasingly complex boundaries but at a greater risk of overfitting.

In general, different kernels have different levels of flexibility. For example, a transformed feature space that is implied by the Gaussian kernel of width σ can be shown to have an infinite number of dimensions by using the polynomial expansion of the exponential term. The parameter σ controls the relative scaling of various dimensions. A smaller value of σ results in a greater ability to model complex boundaries, but it may also cause overfitting. Smaller data sets are more prone to overfitting. Therefore, the optimal values of kernel parameters depend not only on the shape of the decision boundary but also on the size of the training data set. Parameter tuning is important in kernel methods. With proper tuning, many kernel functions can model complex decision boundaries. Furthermore, kernels provide

a natural route for using SVMs in complex data types. This is because kernel methods only need the pairwise similarity between objects, and are agnostic to the feature values of the data points. Kernel functions have been defined for text, images, sequences, and graphs.

10.6.4.1 Other Applications of Kernel Methods

The use of kernel methods is not restricted to SVM methods. These methods can be extended to any technique in which the solutions are directly or indirectly expressed in terms of dot products. Examples include the Fisher's discriminant, logistic regression, linear regression (cf. Sect. 11.5.4 of Chap. 11), dimensionality reduction, and k-means clustering.

1. *Kernel k-means:* The key idea is that the Euclidean distance between a data point $\overline{X}$ and the cluster centroid $\overline{\mu}$ of cluster $\mathcal{C}$ can be computed as a function of the dot product between $\overline{X}$ and the data points in $\mathcal{C}$:

$$||\overline{X}-\overline{\mu}||^2 = ||\overline{X}-\frac{\sum_{\overline{X_i}\in\mathcal{C}}\overline{X_i}}{|\mathcal{C}|}||^2 = \overline{X}\cdot\overline{X}-2\frac{\sum_{\overline{X_i}\in\mathcal{C}}\overline{X}\cdot\overline{X_i}}{|\mathcal{C}|}+\frac{\sum_{\overline{X_i},\overline{X_j}\in\mathcal{C}}\overline{X_i}\cdot\overline{X_j}}{|\mathcal{C}|^2}. \quad (10.66)$$

   In kernel k-means, the dot products $\overline{X_i}\cdot\overline{X_j}$ are replaced with kernel similarity values $K(\overline{X_i},\overline{X_j})$. For the data point $\overline{X}$, the index of its assigned cluster is obtained by selecting the minimum value of the (kernel-based) distance in Eq. 10.66 over all clusters. Note that the cluster centroids in the transformed space do not need to be explicitly maintained over the different k-means iterations, although the cluster assignment indices for each data point need to be maintained for computation of Eq. 10.66. Because of its implicit nonlinear transformation approach, kernel k-means is able to discover arbitrarily shaped clusters like spectral clustering in spite of its use of the spherically biased Euclidean distance.

2. *Kernel PCA:* In conventional *SVD* and *PCA* of an $n\times d$ mean-centered data matrix D, the basis vectors are given by the eigenvectors of D^TD (columnwise dot product matrix), and the coordinates of the transformed points are extracted from the scaled eigenvectors of DD^T (rowwise dot product matrix). While the basis vectors can no longer be derived in kernel *PCA*, the coordinates of the transformed data can be extracted. The rowwise dot product matrix DD^T can be replaced with the kernel similarity matrix $S=[K(\overline{X_i},\overline{X_j})]_{n\times n}$. The similarity matrix is then adjusted for mean-centering of the data in the transformed space as $S\Leftarrow(I-\frac{U}{n})S(I-\frac{U}{n})$, where U is an $n\times n$ matrix containing all 1s (see Exercise 17). The assumption is that the matrix S can be approximately expressed as a dot product of the reduced data points in some k-dimensional transformed space. Therefore, one needs to approximately factorize S into the form AA^T to extract its reduced $n\times k$ embedding A in the transformed space. This is achieved by eigen-decomposition. Let Q_k be the $n\times k$ matrix containing the largest k eigenvectors of S, and Σ_k be the $k\times k$ diagonal matrix containing the square root of the corresponding eigenvalues. Then, it is evident that $S\approx Q_k\Sigma_k^2Q_k^T=(Q_k\Sigma_k)(Q_k\Sigma_k)^T$, and the k-dimensional embeddings of the data points are given[6] by the rows of the $n\times k$ matrix $A=Q_k\Sigma_k$. Note that this is a truncated version of the data-specific Mercer kernel map. This nonlinear embedding is similar to that obtained

[6] The original result [450] uses a more general argument to derive $S'Q_k\Sigma_k^{-1}$ as the $m\times k$ matrix of k-dimensional embedded coordinates of any *out-of-sample* $m\times d$ matrix D'. Here, $S'=D'D^T$ is the $m\times n$ matrix of kernel similarities between out-of-sample points in D' and in-sample points in D. However, when $D'=D$, this expression is (more simply) equivalent to $Q_k\Sigma_k$ by expanding $S'=S\approx Q_k\Sigma_k^2Q_k^T$.