CHAPTER 21 Support Vector Machines

In this chapter we describe Support Vector Machines (SVMs), a classification method based on maximum margin linear discriminants, that is, the goal is to find the optimal hyperplane that maximizes the gap or margin between the classes. Further, we can use the kernel trick to find the optimal nonlinear decision boundary between classes, which corresponds to a hyperplane in some high-dimensional "nonlinear" space.

21.1 SUPPORT VECTORS AND MARGINS

Let $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be a classification dataset, with n points in a d-dimensional space. Further, let us assume that there are only two class labels, that is, $y_i \in \{+1, -1\}$, denoting the positive and negative classes.

Hyperplanes

A hyperplane in d dimensions is given as the set of all points $\mathbf{x} \in \mathbb{R}^d$ that satisfy the equation $h(\mathbf{x}) = 0$, where $h(\mathbf{x})$ is the *hyperplane function*, defined as follows:

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \qquad (21.1) \\ &= w_1x_1 + w_2x_2 + \cdots + w_dx_d + b \end{aligned}$$

Here, $\mathbf{w}$ is a d dimensional *weight vector* and b is a scalar, called the *bias*. For points that lie on the hyperplane, we have

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0 \qquad (21.2)$$

The hyperplane is thus defined as the set of all points such that $\mathbf{w}^T\mathbf{x} = -b$. To see the role played by b, assuming that $w_1 \neq 0$, and setting $x_i = 0$ for all $i > 1$, we can obtain the offset where the hyperplane intersects the first axis, as by Eq. (21.2), we have

$$w_1x_1 = -b \quad \text{or} \quad x_1 = \frac{-b}{w_1}$$

In other words, the point $(\frac{-b}{w_1}, 0, \ldots, 0)$ lies on the hyperplane. In a similar manner, we can obtain the offset where the hyperplane intersects each of the axes, which is given as $\frac{-b}{w_i}$ (provided $w_i \neq 0$).

Separating Hyperplane

A hyperplane splits the original d-dimensional space into two *half-spaces*. A dataset is said to be *linearly separable* if each half-space has points only from a single class. If the input dataset is linearly separable, then we can find a *separating* hyperplane $h(\mathbf{x}) = 0$, such that for all points labeled $y_i = -1$, we have $h(\mathbf{x}_i) < 0$, and for all points labeled $y_i = +1$, we have $h(\mathbf{x}_i) > 0$. In fact, the hyperplane function $h(\mathbf{x})$ serves as a linear classifier or a linear discriminant, which predicts the class y for any given point $\mathbf{x}$, according to the decision rule:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases} \tag{21.3}$$

Let $\mathbf{a}_1$ and $\mathbf{a}_2$ be two arbitrary points that lie on the hyperplane. From Eq. (21.2) we have

$$h(\mathbf{a}_1) = \mathbf{w}^T\mathbf{a}_1 + b = 0$$
$$h(\mathbf{a}_2) = \mathbf{w}^T\mathbf{a}_2 + b = 0$$

Subtracting one from the other we obtain

$$\mathbf{w}^T(\mathbf{a}_1 - \mathbf{a}_2) = 0$$

This means that the weight vector $\mathbf{w}$ is orthogonal to the hyperplane because it is orthogonal to any arbitrary vector $(\mathbf{a}_1 - \mathbf{a}_2)$ on the hyperplane. In other words, the weight vector $\mathbf{w}$ specifies the direction that is normal to the hyperplane, which fixes the orientation of the hyperplane, whereas the bias b fixes the offset of the hyperplane in the d-dimensional space. Because both $\mathbf{w}$ and $-\mathbf{w}$ are normal to the hyperplane, we remove this ambiguity by requiring that $h(\mathbf{x}_i) > 0$ when $y_i = 1$, and $h(\mathbf{x}_i) < 0$ when $y_i = -1$.

Distance of a Point to the Hyperplane

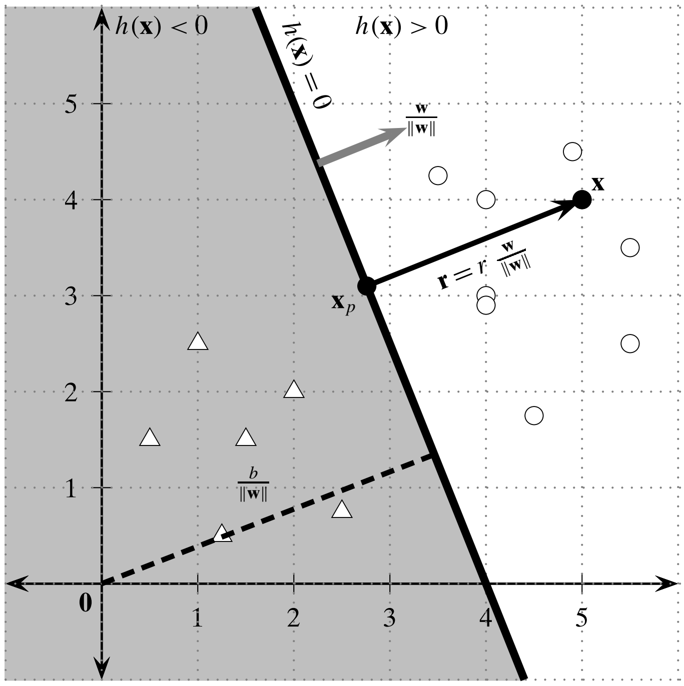
Consider a point $\mathbf{x} \in \mathbb{R}^d$, such that $\mathbf{x}$ does not lie on the hyperplane. Let $\mathbf{x}_p$ be the orthogonal projection of $\mathbf{x}$ on the hyperplane, and let $\mathbf{r} = \mathbf{x} - \mathbf{x}_p$, then as shown in Figure 21.1 we can write $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{x}_p + \mathbf{r}$$
$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{21.4}$$

where r is the *directed distance* of the point $\mathbf{x}$ from $\mathbf{x}_p$, that is, r gives the offset of $\mathbf{x}$ from $\mathbf{x}_p$ in terms of the unit weight vector $\frac{\mathbf{w}}{\|\mathbf{w}\|}$. The offset r is positive if $\mathbf{r}$ is in the same direction as $\mathbf{w}$, and r is negative if $\mathbf{r}$ is in a direction opposite to $\mathbf{w}$.

Plugging Eq. (21.4) into the hyperplane function [Eq. (21.1)], we get

$$\begin{aligned} h(\mathbf{x}) &= h\left(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) \\ &= \mathbf{w}^T\left(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b \end{aligned}$$



$h(\mathbf{x}) < 0$
$h(\mathbf{x}) > 0$
$h(\mathbf{x}) = 0$
$\frac{\mathbf{w}}{\|\mathbf{w}\|}$
$\mathbf{x}$
$\mathbf{r} = r\frac{\mathbf{w}}{\|\mathbf{w}\|}$
$\mathbf{x}_p$
$\frac{b}{\|\mathbf{w}\|}$
0
1
2
3
4
5

Figure 21.1. Geometry of a separating hyperplane in 2D. Points labeled +1 are shown as circles, and those labeled −1 are shown as triangles. The hyperplane $h(\mathbf{x}) = 0$ divides the space into two half-spaces. The shaded region comprises all points $\mathbf{x}$ satisfying $h(\mathbf{x}) < 0$, whereas the unshaded region consists of all points satisfying $h(\mathbf{x}) > 0$. The unit weight vector $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ (in gray) is orthogonal to the hyperplane. The directed distance of the origin to the hyperplane is $\frac{b}{\|\mathbf{w}\|}$.

$$
\begin{aligned}
&= \underbrace{\mathbf{w}^T\mathbf{x}_p + b}_{h(\mathbf{x}_p)} + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|}\\
&= \underbrace{h(\mathbf{x}_p)}_{0} + r\,\|\mathbf{w}\|\\
&= r\,\|\mathbf{w}\|
\end{aligned}
$$

The last step follows from the fact that $h(\mathbf{x}_p) = 0$ because $\mathbf{x}_p$ lies on the hyperplane. Using the result above, we obtain an expression for the directed distance of a point to the hyperplane:

$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$$

To obtain distance, which must be non-negative, we can conveniently multiply r by the class label y of the point because when $h(\mathbf{x}) < 0$, the class is −1, and when $h(\mathbf{x}) > 0$ the class is +1. The distance of a point $\mathbf{x}$ from the hyperplane $h(\mathbf{x}) = 0$ is thus given as

$$\delta = y\,r = \frac{y\,h(\mathbf{x})}{\|\mathbf{w}\|} \tag{21.5}$$

In particular, for the origin $\mathbf{x} = \mathbf{0}$, the directed distance is

$$r = \frac{h(\mathbf{0})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T\mathbf{0} + b}{\|\mathbf{w}\|} = \frac{b}{\|\mathbf{w}\|}$$

as illustrated in Figure 21.1.

Example 21.1. Consider the example shown in Figure 21.1. In this 2-dimensional example, the hyperplane is just a line, defined as the set of all points $\mathbf{x} = (x_1, x_2)^T$ that satisfy the following equation:

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = w_1x_1 + w_2x_2 + b = 0$$

Rearranging the terms we get

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

where $-\frac{w_1}{w_2}$ is the slope of the line, and $-\frac{b}{w_2}$ is the intercept along the second dimension.

Consider any two points on the hyperplane, say $\mathbf{p} = (p_1, p_2) = (4, 0)$, and $\mathbf{q} = (q_1, q_2) = (2, 5)$. The slope is given as

$$-\frac{w_1}{w_2} = \frac{q_2 - p_2}{q_1 - p_1} = \frac{5-0}{2-4} = -\frac{5}{2}$$

which implies that $w_1 = 5$ and $w_2 = 2$. Given any point on the hyperplane, say $(4, 0)$, we can compute the offset b directly as follows:

$$b = -5x_1 - 2x_2 = -5 \cdot 4 - 2 \cdot 0 = -20$$

Thus, $\mathbf{w} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$ is the weight vector, and $b = -20$ is the bias, and the equation of the hyperplane is given as

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \begin{pmatrix} 5 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 20 = 0$$

One can verify that the distance of the origin $\mathbf{0}$ from the hyperplane is given as

$$\delta = y\,r = -1\,r = \frac{-b}{\|\mathbf{w}\|} = \frac{-(-20)}{\sqrt{29}} = 3.71$$

Margin and Support Vectors of a Hyperplane

Given a training dataset of labeled points, $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ with $y_i \in \{+1, -1\}$, and given a separating hyperplane $h(\mathbf{x}) = 0$, for each point $\mathbf{x}_i$ we can find its distance to the hyperplane by Eq. (21.5):

$$\delta_i = \frac{y_i\, h(\mathbf{x}_i)}{\|\mathbf{w}\|} = \frac{y_i\,(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

Over all the n points, we define the *margin* of the linear classifier as the minimum distance of a point from the separating hyperplane, given as

$$\delta^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|} \right\} \tag{21.6}$$

Note that $\delta^* \neq 0$, since $h(\mathbf{x})$ is assumed to be a separating hyperplane, and Eq. (21.3) must be satisfied.

All the points (or vectors) that achieve this minimum distance are called *support vectors* for the hyperplane. In other words, a support vector $\mathbf{x}^*$ is a point that lies precisely on the margin of the classifier, and thus satisfies the condition

$$\delta^* = \frac{y^*(\mathbf{w}^T\mathbf{x}^* + b)}{\|\mathbf{w}\|}$$

where y^* is the class label for $\mathbf{x}^*$. The numerator $y^*(\mathbf{w}^T\mathbf{x}^* + b)$ gives the absolute distance of the support vector to the hyperplane, whereas the denominator $\|\mathbf{w}\|$ makes it a relative distance in terms of $\mathbf{w}$.

Canonical Hyperplane

Consider the equation of the hyperplane [Eq. (21.2)]. Multiplying on both sides by some scalar s yields an equivalent hyperplane:

$$s\, h(\mathbf{x}) = s\, \mathbf{w}^T\mathbf{x} + s\, b = (s\mathbf{w})^T\mathbf{x} + (sb) = 0$$

To obtain the unique or *canonical* hyperplane, we choose the scalar s such that the absolute distance of a support vector from the hyperplane is 1. That is,

$$sy^*(\mathbf{w}^T\mathbf{x}^* + b) = 1$$

which implies

$$s = \frac{1}{y^*(\mathbf{w}^T\mathbf{x}^* + b)} = \frac{1}{y^*h(\mathbf{x}^*)} \tag{21.7}$$

Henceforth, we will assume that any separating hyperplane is canonical. That is, it has already been suitably rescaled so that $y^*h(\mathbf{x}^*) = 1$ for a support vector $\mathbf{x}^*$, and the margin is given as

$$\delta^* = \frac{y^*h(\mathbf{x}^*)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

For the canonical hyperplane, for each support vector $\mathbf{x}_i^*$ (with label y_i^*), we have $y_i^*h(\mathbf{x}_i^*) = 1$, and for any point that is not a support vector we have $y_ih(\mathbf{x}_i) > 1$, because, by definition, it must be farther from the hyperplane than a support vector. Over all the n points in the dataset $\mathbf{D}$, we thus obtain the following set of inequalities:

$$y_i\,(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \text{ for all points } \mathbf{x}_i \in \mathbf{D} \tag{21.8}$$

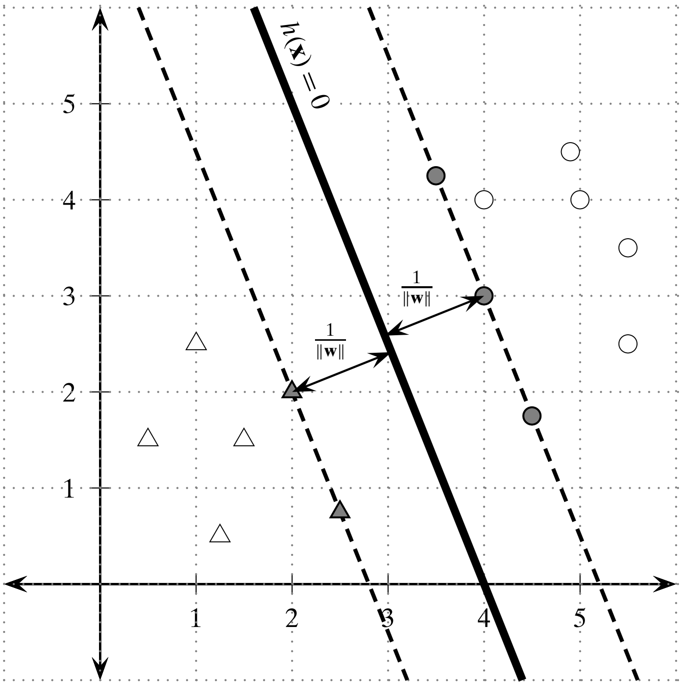


Figure 21.2. Margin of a separating hyperplane: $\frac{1}{\|\mathbf{w}\|}$ is the margin, and the shaded points are the support vectors.

Example 21.2. Figure 21.2 gives an illustration of the support vectors and the margin of a hyperplane. The equation of the separating hyperplane is

$$h(\mathbf{x}) = \begin{pmatrix}5\\2\end{pmatrix}^T \mathbf{x} - 20 = 0$$

Consider the support vector $\mathbf{x}^* = (2,2)^T$, with class $y^* = -1$. To find the canonical hyperplane equation, we have to rescale the weight vector and bias by the scalar s, obtained using Eq. (21.7):

$$s = \frac{1}{y^* h(\mathbf{x}^*)} = \frac{1}{-1\left(\begin{pmatrix}5\\2\end{pmatrix}^T \begin{pmatrix}2\\2\end{pmatrix} - 20\right)} = \frac{1}{6}$$

Thus, the rescaled weight vector is

$$\mathbf{w} = \frac{1}{6}\begin{pmatrix}5\\2\end{pmatrix} = \begin{pmatrix}5/6\\2/6\end{pmatrix}$$

and the rescaled bias is

$$b = \frac{-20}{6}$$

The canonical form of the hyperplane is therefore

$$h(\mathbf{x}) = \begin{pmatrix}5/6\\2/6\end{pmatrix}^T \mathbf{x} - 20/6 = \begin{pmatrix}0.833\\0.333\end{pmatrix}^T \mathbf{x} - 3.33$$

and the margin of the canonical hyperplane is

$$\delta^* = \frac{y^*\, h(\mathbf{x}^*)}{\|\mathbf{w}\|} = \frac{1}{\sqrt{\left(\frac{5}{6}\right)^2 + \left(\frac{2}{6}\right)^2}} = \frac{6}{\sqrt{29}} = 1.114$$

In this example there are five support vectors (shown as shaded points), namely, $(2,2)^T$ and $(2.5, 0.75)^T$ with class $y = -1$ (shown as triangles), and $(3.5, 4.25)^T$, $(4,3)^T$, and $(4.5, 1.75)^T$ with class $y = +1$ (shown as circles), as illustrated in Figure 21.2.

21.2 SVM: LINEAR AND SEPARABLE CASE

Given a dataset $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, let us assume for the moment that the points are linearly separable, that is, there exists a separating hyperplane that perfectly classifies each point. In other words, all points labeled $y_i = +1$ lie on one side ($h(\mathbf{x}) > 0$) and all points labeled $y_i = -1$ lie on the other side ($h(\mathbf{x}) < 0$) of the hyperplane. It is obvious that in the linearly separable case, there are in fact an infinite number of such separating hyperplanes. Which one should we choose?

Maximum Margin Hyperplane

The fundamental idea behind SVMs is to choose the canonical hyperplane, specified by the weight vector $\mathbf{w}$ and the bias b, that yields the maximum margin among all possible separating hyperplanes. If δ_h^* represents the margin for hyperplane $h(\mathbf{x}) = 0$, then the goal is to find the optimal hyperplane h^*:

$$h^* = \arg\max_h \left\{\delta_h^*\right\} = \arg\max_{\mathbf{w},b} \left\{\frac{1}{\|\mathbf{w}\|}\right\}$$

The SVM task is to find the hyperplane that maximizes the margin $\frac{1}{\|\mathbf{w}\|}$, subject to the n constraints given in Eq. (21.8), namely, $y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$, for all points $\mathbf{x}_i \in \mathbf{D}$. Notice that instead of maximizing the margin $\frac{1}{\|\mathbf{w}\|}$, we can minimize $\|\mathbf{w}\|$. In fact, we can obtain an equivalent minimization formulation given as follows:

$$\textbf{Objective Function: } \min_{\mathbf{w},b} \left\{\frac{\|\mathbf{w}\|^2}{2}\right\}$$

$$\textbf{Linear Constraints: } y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1,\ \ \forall \mathbf{x}_i \in \mathbf{D}$$

We can directly solve the above *primal* convex minimization problem with the n linear constraints using standard optimization algorithms, as outlined later in Section 21.5. However, it is more common to solve the *dual* problem, which is obtained via the use of *Lagrange multipliers*. The main idea is to introduce a Lagrange multiplier

α_i for each constraint, which satisfies the Karush–Kuhn–Tucker (KKT) conditions at the optimal solution:

$$\alpha_i \big(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\big) = 0$$
$$\text{and } \alpha_i \geq 0$$

Incorporating all the n constraints, the new objective function, called the *Lagrangian*, then becomes

$$\min L = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \big(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\big) \tag{21.9}$$

L should be minimized with respect to $\mathbf{w}$ and b, and it should be maximized with respect to α_i.

Taking the derivative of L with respect to $\mathbf{w}$ and b, and setting those to zero, we obtain

$$\frac{\partial}{\partial \mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \quad \text{or} \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{21.10}$$

$$\frac{\partial}{\partial b} L = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{21.11}$$

The above equations give important intuition about the optimal weight vector $\mathbf{w}$. In particular, Eq. (21.10) implies that $\mathbf{w}$ can be expressed as a linear combination of the data points $\mathbf{x}_i$, with the signed Lagrange multipliers, $\alpha_i y_i$, serving as the coefficients. Further, Eq. (21.11) implies that the sum of the signed Lagrange multipliers, $\alpha_i y_i$, must be zero.

Plugging these into Eq. (21.9), we obtain the *dual Lagrangian* objective function, which is specified purely in terms of the Lagrange multipliers:

$$\begin{aligned} L_{dual} &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T \underbrace{\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right)}_{\mathbf{w}} - b \underbrace{\sum_{i=1}^{n} \alpha_i y_i}_{0} + \sum_{i=1}^{n} \alpha_i \\ &= -\frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \end{aligned}$$

The dual objective is thus given as

$$\begin{aligned} &\textbf{Objective Function: } \max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ &\textbf{Linear Constraints: } \alpha_i \geq 0,\ \forall i \in \mathbf{D}, \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{21.12}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$ is the vector comprising the Lagrange multipliers. L_{dual} is a convex quadratic programming problem (note the $\alpha_i\alpha_j$ terms), which can be solved using standard optimization techniques. See Section 21.5 for a gradient-based method for solving the dual formulation.

Weight Vector and Bias

Once we have obtained the α_i values for $i = 1, \ldots, n$, we can solve for the weight vector $\mathbf{w}$ and the bias b. Note that according to the KKT conditions, we have

$$\alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right) = 0$$

which gives rise to two cases:

(1) $\alpha_i = 0$, or

(2) $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 = 0$, which implies $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$

This is a very important result because if $\alpha_i > 0$, then $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$, and thus the point $\mathbf{x}_i$ must be a support vector. On the other hand if $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 1$, then $\alpha_i = 0$, that is, if a point is not a support vector, then $\alpha_i = 0$.

Once we know α_i for all points, we can compute the weight vector $\mathbf{w}$ using Eq. (21.10), but by taking the summation only for the support vectors:

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \tag{21.13}$$

In other words, $\mathbf{w}$ is obtained as a linear combination of the support vectors, with the $\alpha_i y_i$'s representing the weights. The rest of the points (with $\alpha_i = 0$) are not support vectors and thus do not play a role in determining $\mathbf{w}$.

To compute the bias b, we first compute one solution b_i, per support vector, as follows:

$$\begin{aligned}
&\alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right) = 0 \\
&y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1 \\
&b_i = \frac{1}{y_i} - \mathbf{w}^T\mathbf{x}_i = y_i - \mathbf{w}^T\mathbf{x}_i
\end{aligned} \tag{21.14}$$

We can take b as the average bias value over all the support vectors:

$$b = \text{avg}_{\alpha_i > 0}\{b_i\} \tag{21.15}$$

SVM Classifier

Given the optimal hyperplane function $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, for any new point $\mathbf{z}$, we predict its class as

$$\hat{y} = \text{sign}(h(\mathbf{z})) = \text{sign}(\mathbf{w}^T\mathbf{z} + b) \tag{21.16}$$

where the sign$(\cdot)$ function returns $+1$ if its argument is positive, and -1 if its argument is negative.

Table 21.1. Dataset corresponding to Figure 21.2

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i
$\mathbf{x}_1$	3.5	4.25	+1
$\mathbf{x}_2$	4	3	+1
$\mathbf{x}_3$	4	4	+1
$\mathbf{x}_4$	4.5	1.75	+1
$\mathbf{x}_5$	4.9	4.5	+1
$\mathbf{x}_6$	5	4	+1
$\mathbf{x}_7$	5.5	2.5	+1
$\mathbf{x}_8$	5.5	3.5	+1
$\mathbf{x}_9$	0.5	1.5	−1
$\mathbf{x}_{10}$	1	2.5	−1
$\mathbf{x}_{11}$	1.25	0.5	−1
$\mathbf{x}_{12}$	1.5	1.5	−1
$\mathbf{x}_{13}$	2	2	−1
$\mathbf{x}_{14}$	2.5	0.75	−1

Example 21.3. Let us continue with the example dataset shown in Figure 21.2. The dataset has 14 points as shown in Table 21.1.

Solving the L_{dual} quadratic program yields the following nonzero values for the Lagrangian multipliers, which determine the support vectors

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	3.5	4.25	+1	0.0437
$\mathbf{x}_2$	4	3	+1	0.2162
$\mathbf{x}_4$	4.5	1.75	+1	0.1427
$\mathbf{x}_{13}$	2	2	−1	0.3589
$\mathbf{x}_{14}$	2.5	0.75	−1	0.0437

All other points have $\alpha_i = 0$ and therefore they are not support vectors. Using Eq. (21.13), we can compute the weight vector for the hyperplane:

$$\begin{aligned}\mathbf{w} &= \sum_{i,\alpha_i>0} \alpha_i y_i \mathbf{x}_i \\ &= 0.0437\begin{pmatrix}3.5\\4.25\end{pmatrix} + 0.2162\begin{pmatrix}4\\3\end{pmatrix} + 0.1427\begin{pmatrix}4.5\\1.75\end{pmatrix} - 0.3589\begin{pmatrix}2\\2\end{pmatrix} - 0.0437\begin{pmatrix}2.5\\0.75\end{pmatrix} \\ &= \begin{pmatrix}0.833\\0.334\end{pmatrix}\end{aligned}$$

The final bias is the average of the bias obtained from each support vector using Eq. (21.14):

$\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i$	$b_i = y_i - \mathbf{w}^T\mathbf{x}_i$
$\mathbf{x}_1$	4.332	−3.332
$\mathbf{x}_2$	4.331	−3.331
$\mathbf{x}_4$	4.331	−3.331
$\mathbf{x}_{13}$	2.333	−3.333
$\mathbf{x}_{14}$	2.332	−3.332
$b = \text{avg}\{b_i\}$		−3.332

Thus, the optimal hyperplane is given as follows:

$$h(\mathbf{x}) = \begin{pmatrix} 0.833 \\ 0.334 \end{pmatrix}^T \mathbf{x} - 3.332 = 0$$

which matches the canonical hyperplane in Example 21.2.

21.3 SOFT MARGIN SVM: LINEAR AND NONSEPARABLE CASE

So far we have assumed that the dataset is perfectly linearly separable. Here we consider the case where the classes overlap to some extent so that a perfect separation is not possible, as depicted in Figure 21.3.

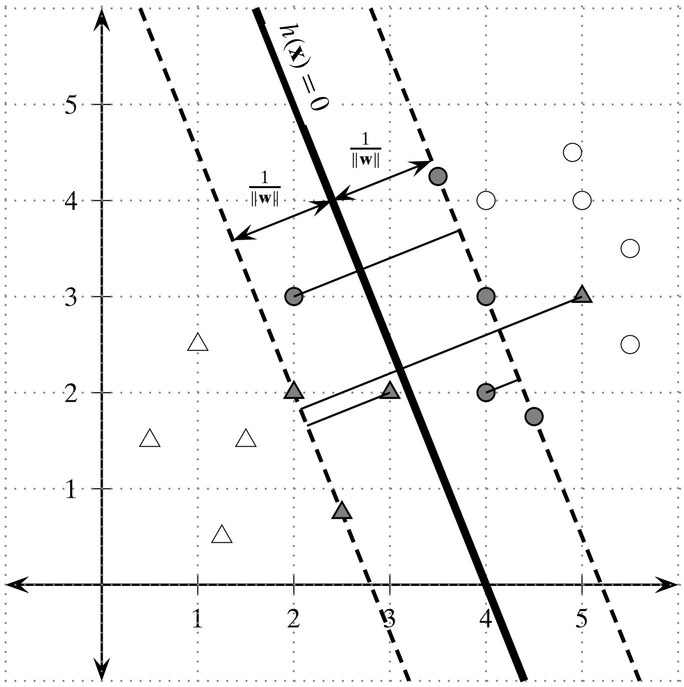


Figure 21.3. Soft margin hyperplane: the shaded points are the support vectors. The margin is $1/\|\mathbf{w}\|$ as illustrated, and points with positive slack values are also shown (thin black line).

Recall that when points are linearly separable we can find a separating hyperplane so that all points satisfy the condition $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$. SVMs can handle non-separable points by introducing *slack variables* ξ_i in Eq. (21.8), as follows:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

where $\xi_i \geq 0$ is the slack variable for point $\mathbf{x}_i$, which indicates how much the point violates the separability condition, that is, the point may no longer be at least $1/\|\mathbf{w}\|$ away from the hyperplane. The slack values indicate three types of points. If $\xi_i = 0$, then the corresponding point $\mathbf{x}_i$ is at least $\frac{1}{\|\mathbf{w}\|}$ away from the hyperplane. If $0 < \xi_i < 1$, then the point is within the margin and still correctly classified, that is, it is on the correct side of the hyperplane. However, if $\xi_i \geq 1$ then the point is misclassified and appears on the wrong side of the hyperplane.

In the nonseparable case, also called the *soft margin* case, the goal of SVM classification is to find the hyperplane with maximum margin that also minimizes the slack terms. The new objective function is given as

$$\begin{aligned} &\textbf{Objective Function: } \min_{\mathbf{w},b,\xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{n}(\xi_i)^k \right\} \\ &\textbf{Linear Constraints: } y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \forall \mathbf{x}_i \in \mathbf{D} \\ &\qquad \xi_i \geq 0\ \forall \mathbf{x}_i \in \mathbf{D} \end{aligned} \tag{21.17}$$

where C and k are constants that incorporate the cost of misclassification. The term $\sum_{i=1}^{n}(\xi_i)^k$ gives the *loss*, that is, an estimate of the deviation from the separable case. The scalar C, which is chosen empirically, is a *regularization constant* that controls the trade-off between maximizing the margin (corresponding to minimizing $\|\mathbf{w}\|^2/2$) or minimizing the loss (corresponding to minimizing the sum of the slack terms $\sum_{i=1}^{n}(\xi_i)^k$). For example, if $C \to 0$, then the loss component essentially disappears, and the objective defaults to maximizing the margin. On the other hand, if $C \to \infty$, then the margin ceases to have much effect, and the objective function tries to minimize the loss. The constant k governs the form of the loss. Typically k is set to 1 or 2. When $k = 1$, called *hinge loss*, the goal is to minimize the sum of the slack variables, whereas when $k = 2$, called *quadratic loss*, the goal is to minimize the sum of the squared slack variables.

21.3.1 Hinge Loss

Assuming $k = 1$, we can compute the Lagrangian for the optimization problem in Eq. (21.17) by introducing Lagrange multipliers α_i and β_i that satisfy the following KKT conditions at the optimal solution:

$$\begin{aligned} \alpha_i\left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right) &= 0 \text{ with } \alpha_i \geq 0 \\ \beta_i(\xi_i - 0) &= 0 \text{ with } \beta_i \geq 0 \end{aligned} \tag{21.18}$$

The Lagrangian is then given as

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right) - \sum_{i=1}^{n}\beta_i\xi_i \tag{21.19}$$

We turn this into a dual Lagrangian by taking its partial derivative with respect to $\mathbf{w}$, b and ξ_i, and setting those to zero:

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \quad \text{or} \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\
&\frac{\partial}{\partial b} L = \sum_{i=1}^{n} \alpha_i y_i = 0 \\
&\frac{\partial}{\partial \xi_i} L = C - \alpha_i - \beta_i = 0 \quad \text{or} \quad \beta_i = C - \alpha_i
\end{aligned} \tag{21.20}$$

Plugging these values into Eq. (21.19), we get

$$\begin{aligned}
L_{dual} &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T \underbrace{\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right)}_{\mathbf{w}} - b \underbrace{\sum_{i=1}^{n} \alpha_i y_i}_{0} + \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \underbrace{(C - \alpha_i + \beta_i)}_{0} \xi_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j
\end{aligned}$$

The dual objective is thus given as

$$\begin{aligned}
&\textbf{Objective Function: } \max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\
&\textbf{Linear Constraints: } 0 \le \alpha_i \le C, \; \forall i \in \mathbf{D} \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0
\end{aligned} \tag{21.21}$$

Notice that the objective is the same as the dual Lagrangian in the linearly separable case [Eq. (21.12)]. However, the constraints on α_i's are different because we now require that $\alpha_i + \beta_i = C$ with $\alpha_i \ge 0$ and $\beta_i \ge 0$, which implies that $0 \le \alpha_i \le C$. Section 21.5 describes a gradient ascent approach for solving this dual objective function.

Weight Vector and Bias

Once we solve for α_i, we have the same situation as before, namely, $\alpha_i = 0$ for points that are not support vectors, and $\alpha_i > 0$ only for the support vectors, which comprise all points $\mathbf{x}_i$ for which we have

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1 - \xi_i \tag{21.22}$$

Notice that the support vectors now include all points that are on the margin, which have zero slack ($\xi_i = 0$), as well as all points with positive slack ($\xi_i > 0$).

We can obtain the weight vector from the support vectors as before:

$$\mathbf{w} = \sum_{i,\alpha_i>0} \alpha_i y_i \mathbf{x}_i \tag{21.23}$$

We can also solve for the β_i using Eq. (21.20):

$$\beta_i = C - \alpha_i$$

Replacing β_i in the KKT conditions [Eq. (21.18)] with the expression from above we obtain

$$(C - \alpha_i)\xi_i = 0 \tag{21.24}$$

Thus, for the support vectors with $\alpha_i > 0$, we have two cases to consider:

(1) $\xi_i > 0$, which implies that $C - \alpha_i = 0$, that is, $\alpha_i = C$, or

(2) $C - \alpha_i > 0$, that is $\alpha_i < C$. In this case, from Eq. (21.24) we must have $\xi_i = 0$. In other words, these are precisely those support vectors that are on the margin.

Using those support vectors that are on the margin, that is, have $0 < \alpha_i < C$ and $\xi_i = 0$, we can solve for b_i:

$$\begin{aligned}
\alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b_i) - 1\right) &= 0\\
y_i(\mathbf{w}^T\mathbf{x}_i + b_i) &= 1\\
b_i &= \frac{1}{y_i} - \mathbf{w}^T\mathbf{x}_i = y_i - \mathbf{w}^T\mathbf{x}_i
\end{aligned} \tag{21.25}$$

To obtain the final bias b, we can take the average over all the b_i values. From Eqs. (21.23) and (21.25), both the weight vector $\mathbf{w}$ and the bias term b can be computed without explicitly computing the slack terms ξ_i for each point.

Once the optimal hyperplane plane has been determined, the SVM model predicts the class for a new point $\mathbf{z}$ as follows:

$$\hat{y} = \text{sign}(h(\mathbf{z})) = \text{sign}(\mathbf{w}^T\mathbf{z} + b)$$

Example 21.4. Let us consider the data points shown in Figure 21.3. There are four new points in addition to the 14 points from Table 21.1 that we considered in Example 21.3; these points are

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i
$\mathbf{x}_{15}$	4	2	+1
$\mathbf{x}_{16}$	2	3	+1
$\mathbf{x}_{17}$	3	2	−1
$\mathbf{x}_{18}$	5	3	−1

Let $k = 1$ and $C = 1$, then solving the L_{dual} yields the following support vectors and Lagrangian values α_i:

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	3.5	4.25	+1	0.0271
$\mathbf{x}_2$	4	3	+1	0.2162
$\mathbf{x}_4$	4.5	1.75	+1	0.9928
$\mathbf{x}_{13}$	2	2	−1	0.9928
$\mathbf{x}_{14}$	2.5	0.75	−1	0.2434
$\mathbf{x}_{15}$	4	2	+1	1
$\mathbf{x}_{16}$	2	3	+1	1
$\mathbf{x}_{17}$	3	2	−1	1
$\mathbf{x}_{18}$	5	3	−1	1

All other points are not support vectors, having $\alpha_i = 0$. Using Eq. (21.23) we compute the weight vector for the hyperplane:

$$\begin{aligned}\mathbf{w} &= \sum_{i,\alpha_i>0} \alpha_i y_i \mathbf{x}_i \\ &= 0.0271\begin{pmatrix}3.5\\4.25\end{pmatrix} + 0.2162\begin{pmatrix}4\\3\end{pmatrix} + 0.9928\begin{pmatrix}4.5\\1.75\end{pmatrix} - 0.9928\begin{pmatrix}2\\2\end{pmatrix} \\ &\quad - 0.2434\begin{pmatrix}2.5\\0.75\end{pmatrix} + \begin{pmatrix}4\\2\end{pmatrix} + \begin{pmatrix}2\\3\end{pmatrix} - \begin{pmatrix}3\\2\end{pmatrix} - \begin{pmatrix}5\\3\end{pmatrix} \\ &= \begin{pmatrix}0.834\\0.333\end{pmatrix}\end{aligned}$$

The final bias is the average of the biases obtained from each support vector using Eq. (21.25). Note that we compute the per-point bias only for the support vectors that lie precisely on the margin. These support vectors have $\xi_i = 0$ and have $0 < \alpha_i < C$. Put another way, we do not compute the bias for support vectors with $\alpha_i = C = 1$, which include the points $\mathbf{x}_{15}$, $\mathbf{x}_{16}$, $\mathbf{x}_{17}$, and $\mathbf{x}_{18}$. From the remaining support vectors, we get

$\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i$	$b_i = y_i - \mathbf{w}^T\mathbf{x}_i$
$\mathbf{x}_1$	4.334	−3.334
$\mathbf{x}_2$	4.334	−3.334
$\mathbf{x}_4$	4.334	−3.334
$\mathbf{x}_{13}$	2.334	−3.334
$\mathbf{x}_{14}$	2.334	−3.334
$b = \text{avg}\{b_i\}$		−3.334

Thus, the optimal hyperplane is given as follows:

$$h(\mathbf{x}) = \begin{pmatrix}0.834\\0.333\end{pmatrix}^T \mathbf{x} - 3.334 = 0$$

One can see that this is essentially the same as the canonical hyperplane we found in Example 21.3.

It is instructive to see what the slack variables are in this case. Note that $\xi_i = 0$ for all points that are not support vectors, and also for those support vectors that are on the margin. So the slack is positive only for the remaining support vectors, for whom the slack can be computed directly from Eq. (21.22), as follows:

$$\xi_i = 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

Thus, for all support vectors not on the margin, we have

$\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i + b$	$\xi_i = 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$
$\mathbf{x}_{15}$	4.001	0.667	0.333
$\mathbf{x}_{16}$	2.667	−0.667	1.667
$\mathbf{x}_{17}$	3.167	−0.167	0.833
$\mathbf{x}_{18}$	5.168	1.834	2.834

As expected, the slack variable $\xi_i > 1$ for those points that are misclassified (i.e., are on the wrong side of the hyperplane), namely $\mathbf{x}_{16} = (3,3)^T$ and $\mathbf{x}_{18} = (5,3)^T$. The other two points are correctly classified, but lie within the margin, and thus satisfy $0 < \xi_i < 1$. The total slack is given as

$$\sum_i \xi_i = \xi_{15} + \xi_{16} + \xi_{17} + \xi_{18} = 0.333 + 1.667 + 0.833 + 2.834 = 5.667$$

21.3.2 Quadratic Loss

For quadratic loss, we have $k = 2$ in the objective function [Eq. (21.17)]. In this case we can drop the positivity constraint $\xi_i \geq 0$ due to the fact that (1) the sum of the slack terms $\sum_{i=1}^n \xi_i^2$ is always positive, and (2) a potential negative value of slack will be ruled out during optimization because a choice of $\xi_i = 0$ leads to a smaller value of the primary objective, and it still satisfies the constraint $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ whenever $\xi_i < 0$. In other words, the optimization process will replace any negative slack variables by zero values. Thus, the SVM objective for quadratic loss is given as

$$\textbf{Objective Function: } \min_{\mathbf{w},b,\xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^n \xi_i^2 \right\}$$

$$\textbf{Linear Constraints: } y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \ \forall \mathbf{x}_i \in \mathbf{D}$$

The Lagrangian is then given as:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^n \xi_i^2 - \sum_{i=1}^n \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i \right) \tag{21.26}$$

Differentiating with respect to $\mathbf{w}$, b, and ξ_i and setting them to zero results in the following conditions, respectively:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\xi_i = \frac{1}{2C}\alpha_i$$

Substituting these back into Eq. (21.26) yields the dual objective

$$\begin{aligned} L_{dual} &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \frac{1}{4C}\sum_{i=1}^{n}\alpha_i^2 \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C}\delta_{ij}\right) \end{aligned}$$

where δ is the *Kronecker delta* function, defined as $\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ otherwise. Thus, the dual objective is given as

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C}\delta_{ij}\right) \\ &\text{subject to the constraints } \alpha_i \geq 0, \forall i \in \mathbf{D}, \text{and } \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{21.27}$$

Once we solve for α_i using the methods from Section 21.5, we can recover the weight vector and bias as follows:

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i$$

$$b = \text{avg}_{i, C > \alpha_i > 0} \left\{ y_i - \mathbf{w}^T \mathbf{x}_i \right\}$$

21.4 KERNEL SVM: NONLINEAR CASE

The linear SVM approach can be used for datasets with a nonlinear decision boundary via the kernel trick from Chapter 5. Conceptually, the idea is to map the original d-dimensional points $\mathbf{x}_i$ in the input space to points $\phi(\mathbf{x}_i)$ in a high-dimensional feature space via some nonlinear transformation ϕ. Given the extra flexibility, it is more likely that the points $\phi(\mathbf{x}_i)$ might be linearly separable in the feature space. Note, however, that a linear decision surface in feature space actually corresponds to a nonlinear decision surface in the input space. Further, the kernel trick allows us to carry out all operations via the kernel function computed in input space, rather than having to map the points into feature space.

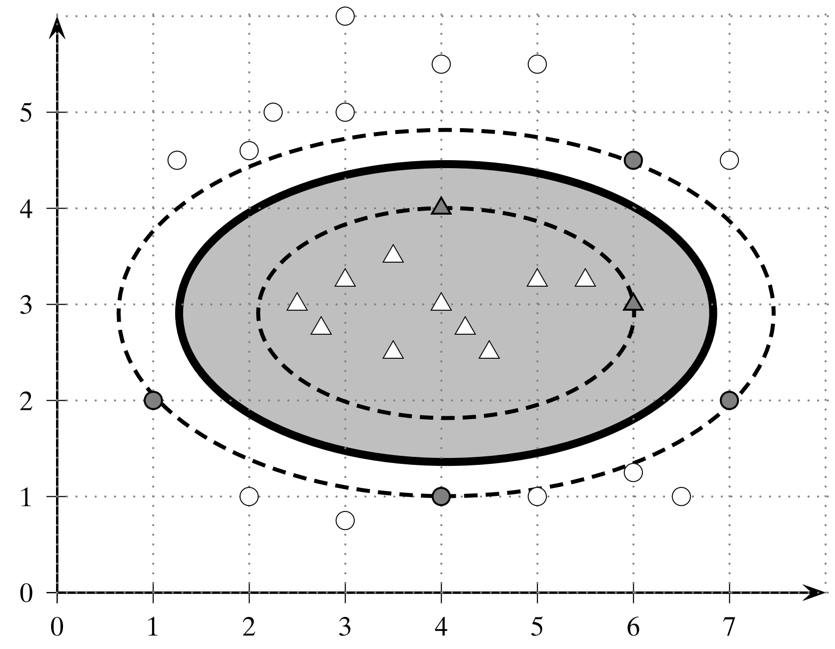


Figure 21.4. Nonlinear SVM: shaded points are the support vectors.

Example 21.5. Consider the set of points shown in Figure 21.4. There is no linear classifier that can discriminate between the points. However, there exists a perfect quadratic classifier that can separate the two classes. Given the input space over the two dimensions X_1 and X_2, if we transform each point $\mathbf{x} = (x_1, x_2)^T$ into a point in the feature space consisting of the dimensions $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$, via the transformation $\phi(\mathbf{x}) = (\sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)^T$, then it is possible to find a separating hyperplane in feature space. For this dataset, it is possible to map the hyperplane back to the input space, where it is seen as an ellipse (thick black line) that separates the two classes (circles and triangles). The support vectors are those points (shown in gray) that lie on the margin (dashed ellipses).

To apply the kernel trick for nonlinear SVM classification, we have to show that all operations require only the kernel function:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Let the original database be given as $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$. Applying ϕ to each point, we can obtain the new dataset in the feature space $\mathbf{D}_\phi = \{\phi(\mathbf{x}_i), y_i\}_{i=1}^n$.

The SVM objective function [Eq. (21.17)] in feature space is given as

$$\textbf{Objective Function: } \min_{\mathbf{w}, b, \xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n (\xi_i)^k \right\} \tag{21.28}$$

$$\textbf{Linear Constraints: } y_i \, (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \text{ and } \xi_i \geq 0, \;\; \forall \mathbf{x}_i \in \mathbf{D}$$

where $\mathbf{w}$ is the weight vector, b is the bias, and ξ_i are the slack variables, all in feature space.

Hinge Loss

For hinge loss, the dual Lagrangian [Eq. (21.21)] in feature space is given as

$$\begin{aligned}\max_{\boldsymbol{\alpha}} L_{dual} &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)\end{aligned} \tag{21.29}$$

Subject to the constraints that $0 \le \alpha_i \le C$, and $\sum_{i=1}^{n} \alpha_i y_i = 0$. Notice that the dual Lagrangian depends only on the dot product between two vectors in feature space $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = K(\mathbf{x}_i,\mathbf{x}_j)$, and thus we can solve the optimization problem using the kernel matrix $\mathbf{K} = \{K(\mathbf{x}_i,\mathbf{x}_j)\}_{i,j=1,\ldots,n}$. Section 21.5 describes a stochastic gradient-based approach for solving the dual objective function.

Quadratic Loss

For quadratic loss, the dual Lagrangian [Eq. (21.27)] corresponds to a change of kernel. Define a new kernel function K_q, as follows:

$$K_q(\mathbf{x}_i,\mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j + \frac{1}{2C}\delta_{ij} = K(\mathbf{x}_i,\mathbf{x}_j) + \frac{1}{2C}\delta_{ij}$$

which affects only the diagonal entries of the kernel matrix $\mathbf{K}$, as $\delta_{ij} = 1$ iff $i = j$, and zero otherwise. Thus, the dual Lagrangian is given as

$$\max_{\boldsymbol{\alpha}} L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j K_q(\mathbf{x}_i,\mathbf{x}_j) \tag{21.30}$$

subject to the constraints that $\alpha_i \ge 0$, and $\sum_{i=1}^{n} \alpha_i y_i = 0$. The above optimization can be solved using the same approach as for hinge loss, with a simple change of kernel.

Weight Vector and Bias

We can solve for $\mathbf{w}$ in feature space as follows:

$$\mathbf{w} = \sum_{\alpha_i > 0} \alpha_i y_i \phi(\mathbf{x}_i) \tag{21.31}$$

Because $\mathbf{w}$ uses $\phi(\mathbf{x}_i)$ directly, in general, we may not be able or willing to compute $\mathbf{w}$ explicitly. However, as we shall see next, it is not necessary to explicitly compute $\mathbf{w}$ for classifying the points.

Let us now see how to compute the bias via kernel operations. Using Eq. (21.25), we compute b as the average over the support vectors that are on the margin, that is, those with $0 < \alpha_i < C$, and $\xi_i = 0$:

$$b = \text{avg}_{i,\,0<\alpha_i<C}\big\{b_i\big\} = \text{avg}_{i,\,0<\alpha_i<C}\Big\{y_i - \mathbf{w}^T\phi(\mathbf{x}_i)\Big\} \tag{21.32}$$

Substituting $\mathbf{w}$ from Eq. (21.31), we obtain a new expression for b_i as

$$\begin{aligned} b_i &= y_i - \sum_{\alpha_j>0} \alpha_j y_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) \\ &= y_i - \sum_{\alpha_j>0} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) \end{aligned} \tag{21.33}$$

Notice that b_i is a function of the dot product between two vectors in feature space and therefore it can be computed via the kernel function in the input space.

Kernel SVM Classifier

We can predict the class for a new point $\mathbf{z}$ as follows:

$$\begin{aligned} \hat{y} &= \text{sign}(\mathbf{w}^T \phi(\mathbf{z}) + b) \\ &= \text{sign}\left(\sum_{\alpha_i>0} \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z}) + b \right) \\ &= \text{sign}\left(\sum_{\alpha_i>0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{z}) + b \right) \end{aligned}$$

Once again we see that $\hat{y}$ uses only dot products in feature space.

Based on the above derivations, we can see that, to train and test the SVM classifier, the mapped points $\phi(\mathbf{x}_i)$ are never needed in isolation. Instead, all operations can be carried out in terms of the kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. Thus, any nonlinear kernel function can be used to do nonlinear classification in the input space. Examples of such nonlinear kernels include the polynomial kernel [Eq. (5.9)], and the Gaussian kernel [Eq. (5.10)], among others.

Example 21.6. Let us consider the example dataset shown in Figure 21.4; it has 29 points in total. Although it is generally too expensive or infeasible (depending on the choice of the kernel) to compute an explicit representation of the hyperplane in feature space, and to map it back into input space, we will illustrate the application of SVMs in both input and feature space to aid understanding.

We use an inhomogeneous polynomial kernel [Eq. (5.9)] of degree $q = 2$, that is, we use the kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$$

With $C=4$, solving the L_{dual} quadratic program [Eq. (21.30)] in input space yields the following six support vectors, shown as the shaded (gray) points in Figure 21.4.

$\mathbf{x}_i$	$(x_{i1}, x_{i2})^T$	$\phi(\mathbf{x}_i)$	y_i	α_i
$\mathbf{x}_1$	$(1,2)^T$	$(1, 1.41, 2.83, 1, 4, 2.83)^T$	$+1$	0.6198
$\mathbf{x}_2$	$(4,1)^T$	$(1, 5.66, 1.41, 16, 1, 5.66)^T$	$+1$	2.069
$\mathbf{x}_3$	$(6,4.5)^T$	$(1, 8.49, 6.36, 36, 20.25, 38.18)^T$	$+1$	3.803
$\mathbf{x}_4$	$(7,2)^T$	$(1, 9.90, 2.83, 49, 4, 19.80)^T$	$+1$	0.3182
$\mathbf{x}_5$	$(4,4)^T$	$(1, 5.66, 5.66, 16, 16, 15.91)^T$	-1	2.9598
$\mathbf{x}_6$	$(6,3)^T$	$(1, 8.49, 4.24, 36, 9, 25.46)^T$	-1	3.8502

For the inhomogeneous quadratic kernel, the mapping ϕ maps an input point $\mathbf{x}_i$ into feature space as follows:

$$\phi\left(\mathbf{x} = (x_1, x_2)^T\right) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right)^T$$

The table above shows all the mapped points, which reside in feature space. For example, $\mathbf{x}_1 = (1,2)^T$ is transformed into

$$\phi(\mathbf{x}_i) = \left(1, \sqrt{2}\cdot 1, \sqrt{2}\cdot 2, 1^2, 2^2, \sqrt{2}\cdot 1\cdot 2\right)^T = (1, 1.41, 2.83, 1, 2, 2.83)^T$$

We compute the weight vector for the hyperplane using Eq. (21.31):

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \phi(\mathbf{x}_i) = (0, -1.413, -3.298, 0.256, 0.82, -0.018)^T$$

and the bias is computed using Eq. (21.32), which yields

$$b = -8.841$$

For the quadratic polynomial kernel, the decision boundary in input space corresponds to an ellipse. For our example, the center of the ellipse is given as $(4.046, 2.907)$, and the semimajor axis length is 2.78 and the semiminor axis length is 1.55. The resulting decision boundary is the ellipse shown in Figure 21.4. We emphasize that in this example we explicitly transformed all the points into the feature space just for illustration purposes. The kernel trick allows us to achieve the same goal using only the kernel function.

21.5 SVM TRAINING ALGORITHMS

We now turn our attention to algorithms for solving the SVM optimization problems. We will consider simple optimization approaches for solving the dual as well as the primal formulations. It is important to note that these methods are not the most efficient. However, since they are relatively simple, they can serve as a starting point for more sophisticated methods.

For the SVM algorithms in this section, instead of dealing explicitly with the bias b, we map each point $\mathbf{x}_i \in \mathbb{R}^d$ to the point $\mathbf{x}_i' \in \mathbb{R}^{d+1}$ as follows:

$$\mathbf{x}_i' = (x_{i1}, \ldots, x_{id}, 1)^T \tag{21.34}$$

Furthermore, we also map the weight vector to $\mathbb{R}^{d+1}$, with $w_{d+1} = b$, so that

$$\mathbf{w} = (w_1, \dots, w_d, b)^T \tag{21.35}$$

The equation of the hyperplane [Eq. (21.1)] is then given as follows:

$$h(\mathbf{x}') : \mathbf{w}^T\mathbf{x}' = 0$$

$$h(\mathbf{x}') : \begin{pmatrix} w_1 & \cdots & w_d & b \end{pmatrix} \begin{pmatrix} x_{i1} \\ \vdots \\ x_{id} \\ 1 \end{pmatrix} = 0$$

$$h(\mathbf{x}') : w_1 x_{i1} + \cdots + w_d x_{id} + b = 0$$

In the discussion below we assume that the bias term has been included in $\mathbf{w}$, and that each point has been mapped to $\mathbb{R}^{d+1}$ as per Eqs. (21.34) and (21.35). Thus, the last component of $\mathbf{w}$ yields the bias b. Another consequence of mapping the points to $\mathbb{R}^{d+1}$ is that the constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$ does not apply in the SVM dual formulations given in Eqs. (21.21), (21.27), (21.29), and (21.30), as there is no explicit bias term b for the linear constraints in the SVM objective given in Eq. (21.17). The new set of constraints is given as

$$y_i \mathbf{w}^T \mathbf{x} \geq 1 - \xi_i$$

21.5.1 Dual Solution: Stochastic Gradient Ascent

We consider only the hinge loss case because quadratic loss can be handled by a change of kernel, as shown in Eq. (21.30). The dual optimization objective for hinge loss [Eq. (21.29)] is given as

$$\max_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the constraints $0 \leq \alpha_i \leq C$ for all $i = 1, \dots, n$. Here $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)^T \in \mathbb{R}^n$.

Let us consider the terms in $J(\boldsymbol{\alpha})$ that involve the Lagrange multiplier α_k:

$$J(\alpha_k) = \alpha_k - \frac{1}{2} \alpha_k^2 y_k^2 K(\mathbf{x}_k, \mathbf{x}_k) - \alpha_k y_k \sum_{\substack{i=1 \\ i \neq k}}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)$$

The gradient or the rate of change in the objective function at $\boldsymbol{\alpha}$ is given as the partial derivative of $J(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$, that is, with respect to each α_k:

$$\nabla J(\boldsymbol{\alpha}) = \left(\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_1}, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_2}, \dots, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_n} \right)^T$$

where the kth component of the gradient is obtained by differentiating $J(\alpha_k)$ with respect to α_k:

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \frac{\partial J(\alpha_k)}{\partial \alpha_k} = 1 - y_k \left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k) \right) \tag{21.36}$$

Because we want to maximize the objective function $J(\boldsymbol{\alpha})$, we should move in the direction of the gradient $\nabla J(\boldsymbol{\alpha})$. Starting from an initial $\boldsymbol{\alpha}$, the gradient ascent approach successively updates it as follows:

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \eta_t \nabla J(\boldsymbol{\alpha}_t)$$

where $\boldsymbol{\alpha}_t$ is the estimate at the tth step, and η_t is the step size.

Instead of updating the entire $\boldsymbol{\alpha}$ vector in each step, in the stochastic gradient ascent approach, we update each component α_k independently and immediately use the new value to update other components. This can result in faster convergence. The update rule for the k-th component is given as

$$\alpha_k = \alpha_k + \eta_k \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) \tag{21.37}$$

where η_k is the step size. We also have to ensure that the constraints $\alpha_k \in [0, C]$ are satisfied. Thus, in the update step above, if $\alpha_k < 0$ we reset it to $\alpha_k = 0$, and if $\alpha_k > C$ we reset it to $\alpha_k = C$. The pseudo-code for stochastic gradient ascent is given in Algorithm 21.1.

ALGORITHM 21.1. Dual SVM Algorithm: Stochastic Gradient Ascent

```
SVM-DUAL ($\mathbf{D}, K, C, \epsilon$):
1  foreach $\mathbf{x}_i \in \mathbf{D}$ do $\mathbf{x}_i \leftarrow \begin{pmatrix}\mathbf{x}_i \\ 1\end{pmatrix}$ // map to $\mathbb{R}^{d+1}$
2  if $loss$ = hinge then
3      $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\dots,n}$ // kernel matrix, hinge loss
4  else if $loss$ = quadratic then
5      $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{2C}\delta_{ij}\}_{i,j=1,\dots,n}$ // kernel matrix, quadratic loss
6  for $k = 1, \dots, n$ do $\eta_k \leftarrow \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)}$ // set step size
7  $t \leftarrow 0$
8  $\boldsymbol{\alpha}_0 \leftarrow (0, \dots, 0)^T$
9  repeat
10     $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}_t$
11     for $k = 1$ to $n$ do
           // update $k$th component of $\boldsymbol{\alpha}$
12         $\alpha_k \leftarrow \alpha_k + \eta_k\left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right)$
13         if $\alpha_k < 0$ then $\alpha_k \leftarrow 0$
14         if $\alpha_k > C$ then $\alpha_k \leftarrow C$
15     $\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}$
16     $t \leftarrow t + 1$
17 until $\|\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{t-1}\| \le \epsilon$
```

To determine the step size η_k, ideally, we would like to choose it so that the gradient at α_k goes to zero, which happens when

$$\eta_k = \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)} \tag{21.38}$$

To see why, note that when only α_k is updated, the other α_i do not change. Thus, the new $\boldsymbol{\alpha}$ has a change only in α_k, and from Eq. (21.36) we get

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \left(1 - y_k \sum_{i \neq k} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) - y_k \alpha_k y_k K(\mathbf{x}_k, \mathbf{x}_k)$$

Plugging in the value of α_k from Eq. (21.37), we have

$$\begin{aligned}\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} &= \left(1 - y_k \sum_{i \neq k} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) - \left(\alpha_k + \eta_k\left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right)\right) K(\mathbf{x}_k, \mathbf{x}_k) \\ &= \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) - \eta_k K(\mathbf{x}_k, \mathbf{x}_k)\left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) \\ &= \left(1 - \eta_k K(\mathbf{x}_k, \mathbf{x}_k)\right)\left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right)\end{aligned}$$

Substituting η_k from Eq. (21.38), we have

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial a_k} = \left(1 - \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)} K(\mathbf{x}_k, \mathbf{x}_k)\right)\left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) = 0$$

In Algorithm 21.1, for better convergence, we thus choose η_k according to Eq. (21.38). The method successively updates $\boldsymbol{\alpha}$ and stops when the change falls below a given threshold ϵ. Since the above description assumes a general kernel function between any two points, we can recover the linear, nonseparable case by simply setting $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$. The computational complexity of the method is $O(n^2)$ per iteration.

Note that once we obtain the final $\boldsymbol{\alpha}$, we classify a new point $\mathbf{z} \in \mathbb{R}^{d+1}$ as follows:

$$\hat{y} = \text{sign}\Big(h(\phi(\mathbf{z}))\Big) = \text{sign}\Big(\mathbf{w}^T \phi(\mathbf{z})\Big) = \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{z})\right)$$

Example 21.7 (Dual SVM: Linear Kernel). Figure 21.5 shows the $n = 150$ points from the Iris dataset, using `sepal length` and `sepal width` as the two attributes. The goal is to discriminate between `Iris-setosa` (shown as circles) and other types of Iris flowers (shown as triangles). Algorithm 21.1 was used to train the SVM classifier with a linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ and convergence threshold $\epsilon = 0.0001$, with hinge loss. Two different values of C were used; hyperplane h_{10} is obtained by using $C = 10$, whereas h_{1000} uses $C = 1000$; the hyperplanes are given as follows:

$$\begin{aligned} h_{10}(\mathbf{x}): &\quad 2.74x_1 - 3.74x_2 - 3.09 = 0 \\ h_{1000}(\mathbf{x}): &\quad 8.56x_1 - 7.14x_2 - 23.12 = 0 \end{aligned}$$

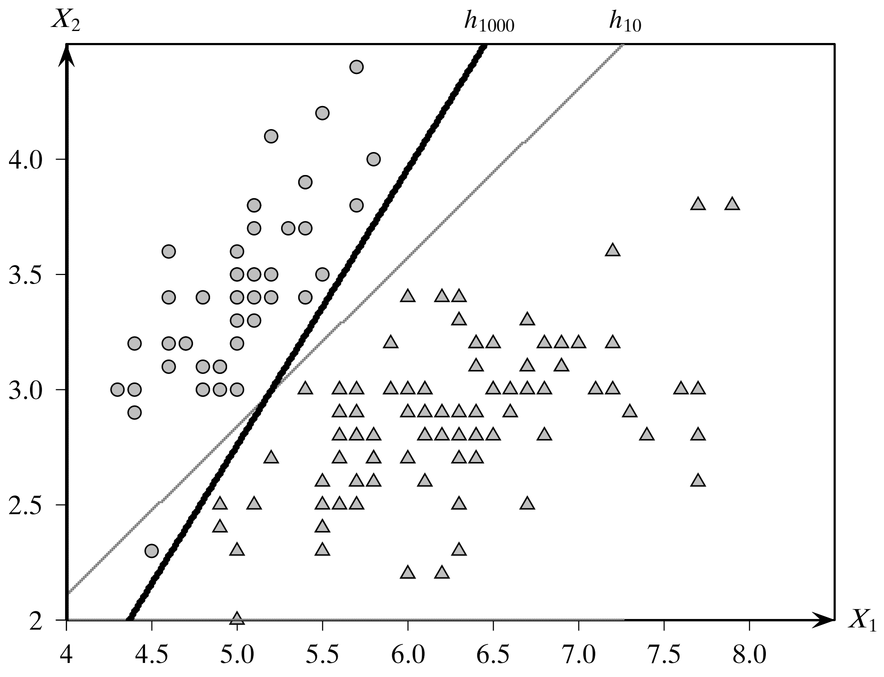


Figure 21.5. SVM dual algorithm with linear kernel.

The hyperplane h_{10} has a larger margin, but it has a larger slack; it misclassifies one of the circles. On the other hand, the hyperplane h_{1000} has a smaller margin, but it minimizes the slack; it is a separating hyperplane. This example illustrates the fact that the higher the value of C the more the emphasis on minimizing the slack.

Example 21.8 (Dual SVM: Quadratic Kernel). Figure 21.6 shows the $n = 150$ points from the Iris dataset projected on the first two principal components. The task is to separate `Iris-versicolor` (in circles) from the other two types of Irises (in triangles). The figure plots the decision boundaries obtained when using the linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$, and the homogeneous quadratic kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j)^2$, where $\mathbf{x}_i \in \mathbb{R}^{d+1}$, as per Eq. (21.34). The optimal hyperplane in both cases was found via the gradient ascent approach in Algorithm 21.1, with $C = 10$, $\epsilon = 0.0001$ and using hinge loss.

The optimal hyperplane h_l (shown in gray) for the linear kernel is given as

$$h_l(\mathbf{x}) : 0.16x_1 + 1.9x_2 + 0.8 = 0$$

As expected, h_l is unable to separate the classes. On the other hand, the optimal hyperplane h_q (shown as clipped black ellipse) for the quadratic kernel is given as

$$h_q(\mathbf{x}) : \mathbf{w}^T\phi(\mathbf{x}) = 1.86x_1^2 + 1.87x_1x_2 + 0.14x_1 + 0.85x_2^2 - 1.22x_2 - 3.25 = 0$$

where $\mathbf{x} = (x_1, x_2)^T$, $\mathbf{w} = \left(1.86, 1.32, 0.099, 0.85, -0.87, -3.25\right)^T$ and $\phi(\mathbf{x}) = \left(x_1^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, x_2^2, \sqrt{2}x_2, 1\right)^T$.

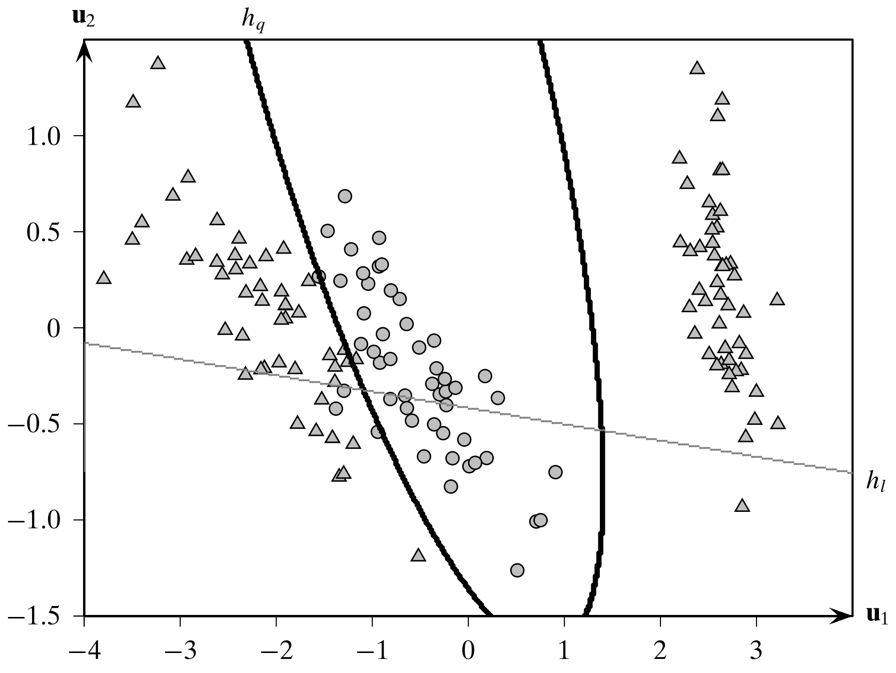


Figure 21.6. SVM dual algorithm with quadratic kernel.

The hyperplane h_q is able to separate the two classes quite well. Here we explicitly reconstructed $\mathbf{w}$ for illustration purposes; note that the last element of $\mathbf{w}$ gives the bias term $b = -3.25$.

21.5.2 Primal Solution: Newton Optimization

The dual approach is the one most commonly used to train SVMs, but it is also possible to train using the primal formulation.

Consider the primal optimization function for the linear, but nonseparable case [Eq. (21.17)]. With $\mathbf{w}, \mathbf{x}_i \in \mathbb{R}^{d+1}$ as discussed earlier, we have to minimize the objective function:

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i)^k \tag{21.39}$$

subject to the linear constraints:

$$y_i\ (\mathbf{w}^T\mathbf{x}_i) \ge 1 - \xi_i \text{ and } \xi_i \ge 0 \text{ for all } i = 1, \dots, n$$

Rearranging the above, we obtain an expression for ξ_i

$$\xi_i \ge 1 - y_i\ (\mathbf{w}^T\mathbf{x}_i) \text{ and } \xi_i \ge 0, \text{ which implies that}$$

$$\xi_i = \max\left\{0, 1 - y_i\ (\mathbf{w}^T\mathbf{x}_i)\right\} \tag{21.40}$$

Plugging Eq. (21.40) into the objective function [Eq. (21.39)], we obtain

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left\{0, 1 - y_i\,(\mathbf{w}^T\mathbf{x}_i)\right\}^k \\ &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)^k \end{aligned} \tag{21.41}$$

The last step follows from Eq. (21.40) because $\xi_i > 0$ if and only if $1 - y_i(\mathbf{w}^T\mathbf{x}_i) > 0$, that is, $y_i(\mathbf{w}^T\mathbf{x}_i) < 1$. Unfortunately, the hinge loss formulation, with $k = 1$, is not differentiable. One could use a differentiable approximation to the hinge loss, but here we describe the quadratic loss formulation.

Quadratic Loss

For quadratic loss, we have $k = 2$, and the primal objective [Eq. (21.41)] can be written as

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)^2$$

The gradient or the rate of change of the objective function at $\mathbf{w}$ is given as the partial derivative of $J(\mathbf{w})$ with respect to $\mathbf{w}$:

$$\begin{aligned} \nabla_{\mathbf{w}} = \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{w} - 2C\Big(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i\left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)\Big) \\ &= \mathbf{w} - 2C\Big(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i\Big) + 2C\Big(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \mathbf{x}_i\mathbf{x}_i^T\Big)\mathbf{w} \\ &= \mathbf{w} - 2C\mathbf{v} + 2C\mathbf{S}\mathbf{w} \end{aligned}$$

where the vector $\mathbf{v}$ and the matrix $\mathbf{S}$ are given as

$$\mathbf{v} = \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i \qquad\qquad \mathbf{S} = \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \mathbf{x}_i\mathbf{x}_i^T$$

Note that the matrix $\mathbf{S}$ is the scatter matrix, and the vector $\mathbf{v}$ is m times the mean of the, say m, signed points $y_i\mathbf{x}_i$ that satisfy the condition $y_i h(\mathbf{x}_i) < 1$.

The *Hessian matrix* is defined as the matrix of second-order partial derivatives of $J(\mathbf{w})$ with respect to $\mathbf{w}$, which is given as

$$\mathbf{H}_{\mathbf{w}} = \frac{\partial \nabla_{\mathbf{w}}}{\partial \mathbf{w}} = \mathbf{I} + 2C\mathbf{S}$$

Because we want to minimize the objective function $J(\mathbf{w})$, we should move in the direction opposite to the gradient. The Newton optimization update rule for $\mathbf{w}$ is given as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{H}_{\mathbf{w}_t}^{-1} \nabla_{\mathbf{w}_t} \tag{21.42}$$

where $\eta_t > 0$ is a scalar value denoting the step size at iteration t. Normally one needs to use a line search method to find the optimal step size η_t, but the default value of $\eta_t = 1$ usually works for quadratic loss.

ALGORITHM 21.2. Primal SVM Algorithm: Newton Optimization, Quadratic Loss

```
SVM-PRIMAL ($\mathbf{D}, C, \epsilon$):
1  foreach $\mathbf{x}_i \in \mathbf{D}$ do
2      $\mathbf{x}_i \leftarrow \begin{pmatrix}\mathbf{x}_i \\ 1\end{pmatrix}$ // map to $\mathbb{R}^{d+1}$
3  $t \leftarrow 0$
4  $\mathbf{w}_0 \leftarrow (0,\dots,0)^T$ // initialize $\mathbf{w}_t \in \mathbb{R}^{d+1}$
5  repeat
6      $\mathbf{v} \leftarrow \sum_{y_i(\mathbf{w}_t^T\mathbf{x}_i)<1} y_i\mathbf{x}_i$
7      $\mathbf{S} \leftarrow \sum_{y_i(\mathbf{w}_t^T\mathbf{x}_i)<1} \mathbf{x}_i\mathbf{x}_i^T$
8      $\nabla \leftarrow (\mathbf{I}+2C\mathbf{S})\mathbf{w}_t - 2C\mathbf{v}$ // gradient
9      $\mathbf{H} \leftarrow \mathbf{I}+2C\mathbf{S}$ // Hessian
10     $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t\mathbf{H}^{-1}\nabla$ // Newton update rule [Eq. (21.42)]
11     $t \leftarrow t+1$
12 until $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| \le \epsilon$
```

The Newton optimization algorithm for training linear, nonseparable SVMs in the primal is given in Algorithm 21.2. The step size η_t is set to 1 by default. After computing the gradient and Hessian at $\mathbf{w}_t$ (lines 6–9), the Newton update rule is used to obtain the new weight vector $\mathbf{w}_{t+1}$ (line 10). The iterations continue until there is very little change in the weight vector. Computing $\mathbf{S}$ requires $O(nd^2)$ steps; computing the gradient ∇, the Hessian matrix $\mathbf{H}$ and updating the weight vector $\mathbf{w}_{t+1}$ takes time $O(d^2)$; and inverting the Hessian takes $O(d^3)$ operations, for a total computational complexity of $O(nd^2 + d^3)$ per iteration in the worst case.

Example 21.9 (Primal SVM). Figure 21.7 plots the hyperplanes obtained using the dual and primal approaches for the 2-dimensional Iris dataset comprising the `sepal length` versus `sepal width` attributes. We used $C = 1000$ and $\epsilon = 0.0001$ with the quadratic loss function. The dual solution h_d (gray line) and the primal solution h_p (thick black line) are essentially identical; they are as follows:

$$h_d(\mathbf{x}): \quad 7.47x_1 - 6.34x_2 - 19.89 = 0$$
$$h_p(\mathbf{x}): \quad 7.47x_1 - 6.34x_2 - 19.91 = 0$$

Primal Kernel SVMs

In the preceding discussion we considered the linear, nonseparable case for primal SVM learning. We now generalize the primal approach to learn kernel-based SVMs, again for quadratic loss.

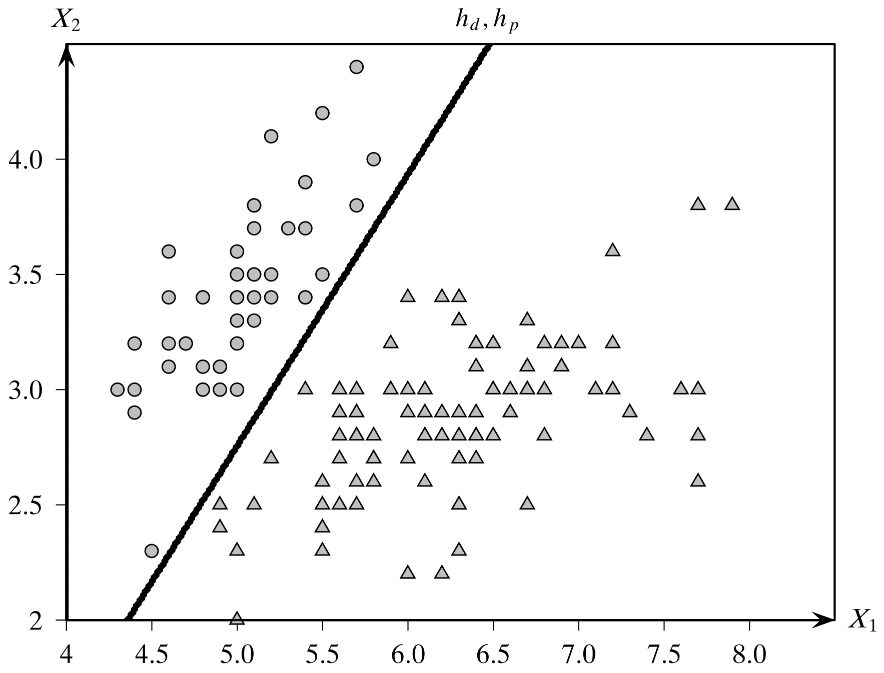


Figure 21.7. SVM primal algorithm with linear kernel.

Let ϕ denote a mapping from the input space to the feature space; each input point $\mathbf{x}_i$ is mapped to the feature point $\phi(\mathbf{x}_i)$. Let $K(\mathbf{x}_i, \mathbf{x}_j)$ denote the kernel function, and let $\mathbf{w}$ denote the weight vector in feature space. The hyperplane in feature space is then given as

$$h(\mathbf{x}): \mathbf{w}^T\phi(\mathbf{x}) = 0$$

Using Eqs. (21.28) and (21.40), the primal objective function in feature space can be written as

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} L(y_i, h(\mathbf{x}_i)) \tag{21.43}$$

where $L(y_i, h(\mathbf{x}_i)) = \max\{0, 1 - y_i h(\mathbf{x}_i)\}^k$ is the loss function.

The gradient at $\mathbf{w}$ is given as

$$\nabla_{\mathbf{w}} = \mathbf{w} + C\sum_{i=1}^{n} \frac{\partial L(y_i, h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)} \cdot \frac{\partial h(\mathbf{x}_i)}{\partial \mathbf{w}}$$

where

$$\frac{\partial h(\mathbf{x}_i)}{\partial \mathbf{w}} = \frac{\partial \mathbf{w}^T\phi(\mathbf{x}_i)}{\partial \mathbf{w}} = \phi(\mathbf{x}_i)$$

At the optimal solution, the gradient vanishes, that is, $\nabla_{\mathbf{w}} = \mathbf{0}$, which yields

$$\begin{aligned}\mathbf{w} &= -C\sum_{i=1}^{n} \frac{\partial L(y_i, h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)} \cdot \phi(\mathbf{x}_i)\\ &= \sum_{i=1}^{n} \beta_i \phi(\mathbf{x}_i) \qquad (21.44)\end{aligned}$$

where β_i is the coefficient of the point $\phi(\mathbf{x}_i)$ in feature space. In other words, the optimal weight vector in feature space is expressed as a linear combination of the points $\phi(\mathbf{x}_i)$ in feature space.

Using Eq. (21.44), the distance to the hyperplane in feature space can be expressed as

$$y_i h(\mathbf{x}_i) = y_i \mathbf{w}^T \phi(\mathbf{x}_i) = y_i \sum_{j=1}^{n} \beta_j K(\mathbf{x}_j, \mathbf{x}_i) = y_i \mathbf{K}_i^T \boldsymbol{\beta} \qquad (21.45)$$

where $\mathbf{K} = \left\{K(\mathbf{x}_i, \mathbf{x}_j)\right\}_{i,j=1}^{n}$ is the $n \times n$ kernel matrix, $\mathbf{K}_i$ is the ith column of $\mathbf{K}$, and $\boldsymbol{\beta} = \left(\beta_1, \dots, \beta_n\right)^T$ is the coefficient vector.

Plugging Eqs. (21.44) and (21.45) into Eq. (21.43), with quadratic loss ($k = 2$), yields the primal kernel SVM formulation purely in terms of the kernel matrix:

$$\begin{aligned}\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j K(\mathbf{x}_i, \mathbf{x}_j) + C\sum_{i=1}^{n} \max\left\{0, 1 - y_i \mathbf{K}_i^T \boldsymbol{\beta}\right\}^2\\ &= \frac{1}{2}\boldsymbol{\beta}^T \mathbf{K} \boldsymbol{\beta} + C \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} (1 - y_i \mathbf{K}_i^T \boldsymbol{\beta})^2\end{aligned}$$

The gradient of $J(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ is given as

$$\begin{aligned}\nabla_{\boldsymbol{\beta}} = \frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \mathbf{K}\boldsymbol{\beta} - 2C \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} y_i \mathbf{K}_i (1 - y_i \mathbf{K}_i^T \boldsymbol{\beta})\\ &= \mathbf{K}\boldsymbol{\beta} + 2C \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} (\mathbf{K}_i \mathbf{K}_i^T)\, \boldsymbol{\beta} - 2C \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} y_i \mathbf{K}_i\\ &= (\mathbf{K} + 2C\mathbf{S})\,\boldsymbol{\beta} - 2C\mathbf{v}\end{aligned}$$

where the vector $\mathbf{v} \in \mathbb{R}^n$ and the matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$ are given as

$$\mathbf{v} = \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} y_i \mathbf{K}_i \qquad\qquad \mathbf{S} = \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} \mathbf{K}_i \mathbf{K}_i^T$$

Furthermore, the *Hessian matrix* is given as

$$\mathbf{H}_{\boldsymbol{\beta}} = \frac{\partial \nabla_{\boldsymbol{\beta}}}{\partial \boldsymbol{\beta}} = \mathbf{K} + 2C\mathbf{S}$$

We can now minimize $J(\boldsymbol{\beta})$ by Newton optimization using the following update rule:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta_t \mathbf{H}_{\boldsymbol{\beta}}^{-1} \nabla_{\boldsymbol{\beta}}$$

ALGORITHM 21.3. Primal Kernel SVM Algorithm: Newton Optimization, Quadratic Loss

```
SVM-PRIMAL-KERNEL ($\mathbf{D}, K, C, \epsilon$):
1  foreach $\mathbf{x}_i \in \mathbf{D}$ do
2      $\mathbf{x}_i \leftarrow \begin{pmatrix}\mathbf{x}_i \\ 1\end{pmatrix}$ // map to $\mathbb{R}^{d+1}$
3  $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\dots,n}$ // compute kernel matrix
4  $t \leftarrow 0$
5  $\boldsymbol{\beta}_0 \leftarrow (0, \dots, 0)^T$ // initialize $\boldsymbol{\beta}_t \in \mathbb{R}^n$
6  repeat
7      $\mathbf{v} \leftarrow \sum_{y_i(\mathbf{K}_i^T \boldsymbol{\beta}_t)<1} y_i \mathbf{K}_i$
8      $\mathbf{S} \leftarrow \sum_{y_i(\mathbf{K}_i^T \boldsymbol{\beta}_t)<1} \mathbf{K}_i \mathbf{K}_i^T$
9      $\nabla \leftarrow (\mathbf{K} + 2C\mathbf{S})\boldsymbol{\beta}_t - 2C\mathbf{v}$ // gradient
10     $\mathbf{H} \leftarrow \mathbf{K} + 2C\mathbf{S}$ // Hessian
11     $\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_t \mathbf{H}^{-1} \nabla$ // Newton update rule
12     $t \leftarrow t + 1$
13 until $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}_{t-1}\| \le \epsilon$
```

Note that if $\mathbf{H}_\beta$ is singular, that is, if it does not have an inverse, then we add a small *ridge* to the diagonal to regularize it. That is, we make $\mathbf{H}$ invertible as follows:

$$\mathbf{H}_\beta = \mathbf{H}_\beta + \lambda \mathbf{I}$$

where $\lambda > 0$ is some small positive ridge value.

Once $\boldsymbol{\beta}$ has been found, it is easy to classify any test point $\mathbf{z}$ as follows:

$$\hat{y} = \text{sign}\left(\mathbf{w}^T \phi(\mathbf{z})\right) = \text{sign}\left(\sum_{i=1}^n \beta_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z})\right) = \text{sign}\left(\sum_{i=1}^n \beta_i K(\mathbf{x}_i, \mathbf{z})\right)$$

The Newton optimization algorithm for kernel SVM in the primal is given in Algorithm 21.3. The step size η_t is set to 1 by default, as in the linear case. In each iteration, the method first computes the gradient and Hessian (lines 7–10). Next, the Newton update rule is used to obtain the updated coefficient vector $\boldsymbol{\beta}_{t+1}$ (line 11). The iterations continue until there is very little change in $\boldsymbol{\beta}$. The computational complexity of the method is $O(n^3)$ per iteration in the worst case.

Example 21.10 (Primal SVM: Quadratic Kernel). Figure 21.8 plots the hyperplanes obtained using the dual and primal approaches on the Iris dataset projected onto the first two principal components. The task is to separate `iris versicolor` from the others, the same as in Example 21.8. Because a linear kernel is not suitable for this task, we employ the quadratic kernel. We further set $C = 10$ and $\epsilon = 0.0001$, with

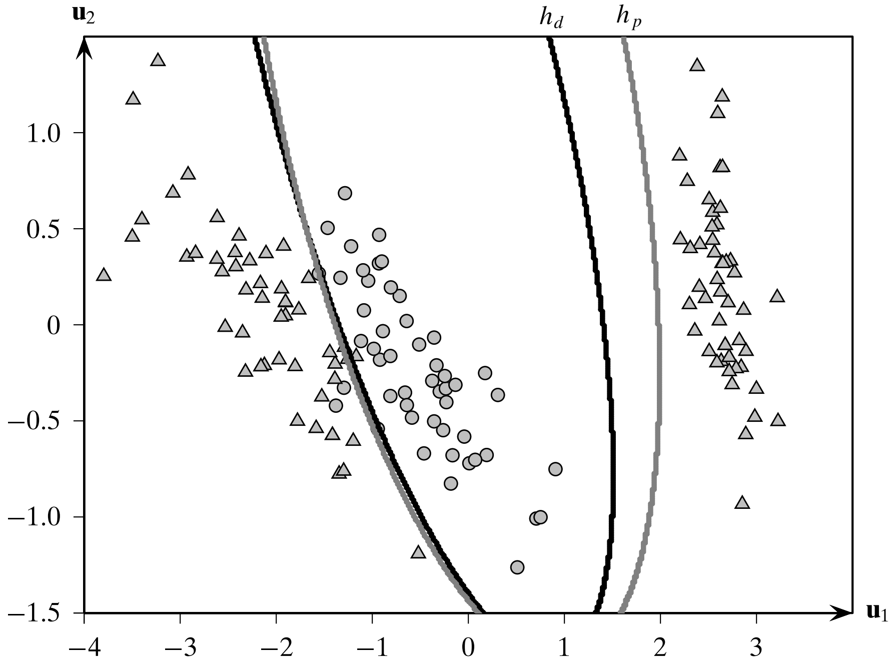


Figure 21.8. SVM quadratic kernel: dual and primal.

the quadratic loss function. The dual solution h_d (black contours) and the primal solution h_p (gray contours) are given as follows:

$$\begin{aligned} h_d(\mathbf{x}):&\quad 1.4x_1^2 + 1.34x_1x_2 - 0.05x_1 + 0.66x_2^2 - 0.96x_2 - 2.66 = 0 \\ h_p(\mathbf{x}):&\quad 0.87x_1^2 + 0.64x_1x_2 - 0.5x_1 + 0.43x_2^2 - 1.04x_2 - 2.398 = 0 \end{aligned}$$

Although the solutions are not identical, they are close, especially on the left decision boundary.

21.6 FURTHER READING

The origins of support vector machines can be found in Vapnik (1982). In particular, it introduced the generalized portrait approach for constructing an optimal separating hyperplane. The use of the kernel trick for SVMs was introduced in Boser, Guyon, and Vapnik (1992), and the soft margin SVM approach for nonseparable data was proposed in Cortes and Vapnik (1995). For a good introduction to support vector machines, including implementation techniques, see Cristianini and Shawe-Taylor (2000) and Schölkopf and Smola (2002). The primal training approach described in this chapter is from Chapelle (2007).

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). "A training algorithm for optimal margin classifiers." *In Proceedings of the 5th Annual Workshop on Computational Learning Theory,* ACM, pp. 144–152.

Chapelle, O. (2007). “Training a support vector machine in the primal.” *Neural Computation*, 19 (5): 1155–1178.

Cortes, C. and Vapnik, V. (1995). “Support-vector networks.” *Machine Learning*, 20 (3): 273–297.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. Cambridge, MA: MIT Press.

Vapnik, V. N. (1982). *Estimation of Dependences Based on Empirical Data,* vol. 41. New York: Springer-Verlag.

21.7 EXERCISES

Q1. Consider the dataset in Figure 21.9, which has points from two classes c_1 (triangles) and c_2 (circles). Answer the questions below.

(a) Find the equations for the two hyperplanes h_1 and h_2.

(b) Show all the support vectors for h_1 and h_2.

(c) Which of the two hyperplanes is better at separating the two classes based on the margin computation?

(d) Find the equation of the best separating hyperplane for this dataset, and show the corresponding support vectors. You can do this witout having to solve the Lagrangian by considering the convex hull of each class and the possible hyperplanes at the boundary of the two classes.

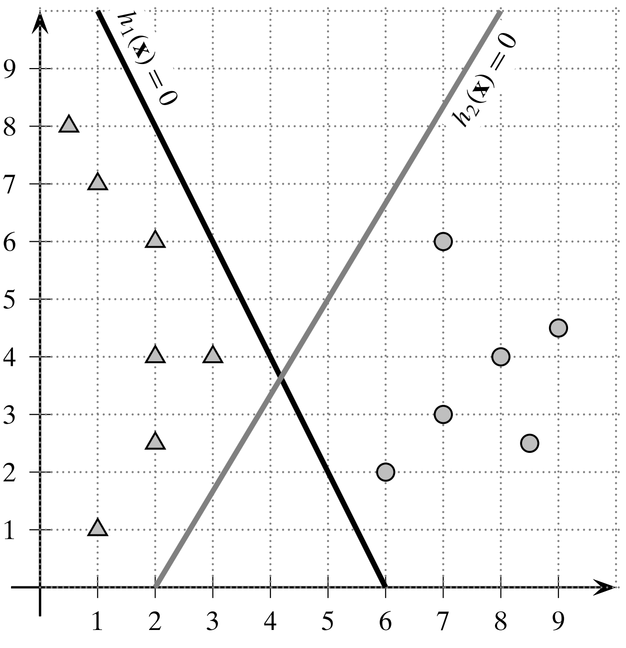


Figure 21.9. Dataset for Q1.

Table 21.2. Dataset for Q2

i	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	4	2.9	1	0.414
$\mathbf{x}_2$	4	4	1	0
$\mathbf{x}_3$	1	2.5	−1	0
$\mathbf{x}_4$	2.5	1	−1	0.018
$\mathbf{x}_5$	4.9	4.5	1	0
$\mathbf{x}_6$	1.9	1.9	−1	0
$\mathbf{x}_7$	3.5	4	1	0.018
$\mathbf{x}_8$	0.5	1.5	−1	0
$\mathbf{x}_9$	2	2.1	−1	0.414
$\mathbf{x}_{10}$	4.5	2.5	1	0

Q2. Given the 10 points in Table 21.2, along with their classes and their Lagranian multipliers (α_i), answer the following questions:

(a) What is the equation of the SVM hyperplane $h(\mathbf{x})$?

(b) What is the distance of $\mathbf{x}_6$ from the hyperplane? Is it within the margin of the classifier?

(c) Classify the point $\mathbf{z} = (3, 3)^T$ using $h(\mathbf{x})$ from above.