

## Chapter 20

# Support Vector Machine

### 20.1 Linear Discriminants

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

In Chapter 19 we were interested in finding the line that best discriminates between the two classes. Note that a line is essentially a one dimensional object in the  $d$ -dimensional space. Here we are still interested in a linear discriminants, but instead of a vector, we are interested in the best hyperplane (which is essentially a  $d - 1$  dimensional object) that separates the two classes.

A linear discriminant function in  $d$  dimensions is given by a hyperplane, defined as follows:

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \quad (20.1)$$

$$= w_1 x^1 + w_2 x^2 + \dots + w_d x^d + b \quad (20.2)$$

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

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0 \quad (20.3)$$

In other words, the hyperplane is 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, since by (20.3), we have

$$w_1 x^1 = -b \quad \text{or} \quad x^1 = \frac{-b}{w_1} \quad (20.4)$$

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

A hyperplane  $h(\mathbf{x})$  splits the original  $d$ -dimensional space into two *half-spaces*. If the input dataset is linearly separable, then we can find a hyperplane  $h(\mathbf{x})$ , 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, for any given point  $\mathbf{x}$ ,  $h(\mathbf{x})$  serves as the linear classifier, which predicts the class  $y$  according to the decision rule:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases} \quad (20.5)$$

Note that if  $h(\mathbf{x}) = 0$  we may arbitrarily choose one of the two classes.

Consider any two points  $\mathbf{x}_i$  and  $\mathbf{x}_j$  that lie on the hyperplane. From (20.3) we have

$$h(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i + b = 0 \quad (20.6)$$

$$h(\mathbf{x}_j) = \mathbf{w}^T \mathbf{x}_j + b = 0 \quad (20.7)$$

Subtracting one from the other we obtain

$$\mathbf{w}^T (\mathbf{x}_i - \mathbf{x}_j) = 0 \quad (20.8)$$

This means that the weight vector  $\mathbf{w}$  is orthogonal to the hyperplane, since it is orthogonal to any arbitrary vector on the hyperplane ( $\mathbf{x}_i - \mathbf{x}_j$ ). In other words, the weight vector  $\mathbf{w}$ , gives the direction that is normal to the hyperplane. Thus  $\mathbf{w}$  gives the orientation of the hyperplane, while the bias  $b$  fixes the hyperplane in the  $d$ -dimensional space.

Now consider a point  $\mathbf{x} \in \mathbb{R}^d$ , such that  $\mathbf{x}$  does not lie on the hyperplane. Further let  $\mathbf{x}_p$  be the projection of  $\mathbf{x}$  on the hyperplane. Let  $r$  be the distance of  $\mathbf{x}$  to the hyperplane, then as shown in Figure 20.1, we can write  $\mathbf{x}$  as a combination of two vectors:

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|} \quad (20.9)$$

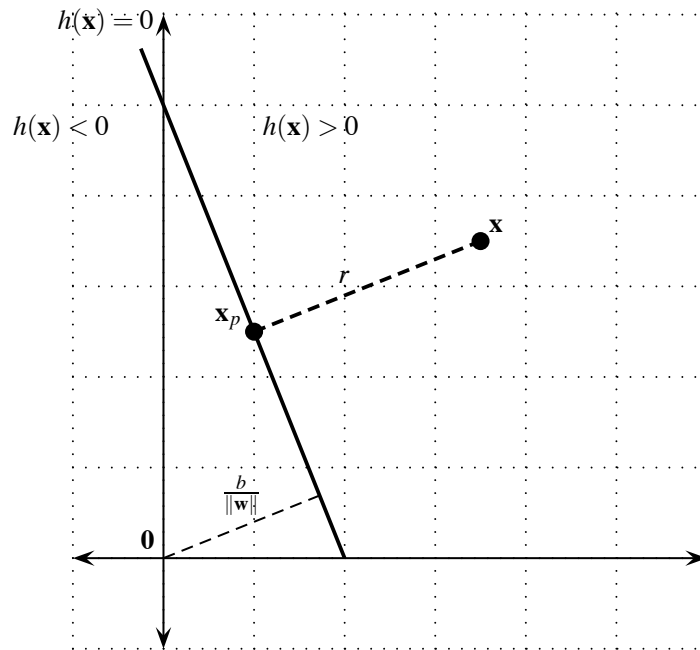


Figure 20.1: Geometry of the Hyperplane

Let us now evaluate the function  $h(\mathbf{x})$ :

$$h(\mathbf{x}) = h\left(\mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}\right) \quad (20.10)$$

$$= \mathbf{w}^T \left( \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) + b \quad (20.11)$$

$$= \mathbf{w}^T \mathbf{x}_p + b + r \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} \quad (20.12)$$

$$= h(\mathbf{x}_p) + r \|\mathbf{w}\| \quad (20.13)$$

$$= r \|\mathbf{w}\| \quad (20.14)$$

Note that  $h(\mathbf{x}_p) = 0$  since  $\mathbf{x}_p$  lies on the hyperplane. From 20.14, we obtain an expression for the distance of any point to the hyperplane, which is given as

$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|} \quad (20.15)$$

In particular, for the origin  $\mathbf{x} = \mathbf{0}$ , we have

$$r = \frac{h(\mathbf{0})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T \mathbf{0} + b}{\|\mathbf{w}\|} = \frac{b}{\|\mathbf{w}\|} \quad (20.16)$$

Thus the origin lies at a distance of  $\frac{b}{\|\mathbf{w}\|}$  from the hyperplane.

**Example:** Consider the example shown in Figure 20.1. In this two-dimensional example, the hyperplane is just a line. Recall that the equation of a line is given as:

$$y = mx + b \quad (20.17)$$

where  $m$  is the slope, and  $b$  is the  $y$ -axis offset. We can find out the equation of the line by first computing its slope:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad (20.18)$$

where  $(x_1, y_1)$  and  $(x_2, y_2)$  are any two points on the line. Since  $(2, 0)$  and  $(0, 5)$  are two such points in Figure 20.1, we obtain  $m = \frac{5-0}{0-2} = -\frac{5}{2}$ .

Next, using the slope from above, and one of the points on the line, say  $(2, 0)$ , we can compute the offset  $b$ .

$$\begin{aligned} y &= -\frac{5}{2}x + b \\ 0 &= -\frac{5}{2} \cdot 2 + b \\ b &= 5 \end{aligned}$$

Thus the equation of the hyperplane is given as:

$$y = -\frac{5}{2}x + 5 \quad (20.19)$$

$$5x + 2y - 10 = 0 \quad (20.20)$$

$$(5 \ 2) \begin{pmatrix} x \\ y \end{pmatrix} - 10 = 0 \quad (20.21)$$

where  $\mathbf{w} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$  is the weight vector, and  $b = -10$  is the bias.

## 20.2 SVM: Linearly Separable Points

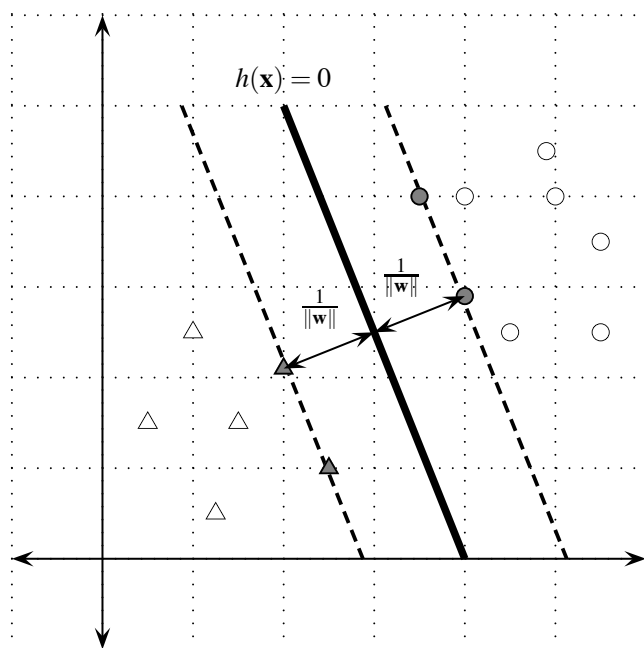


Figure 20.2: Maximum Margin Hyperplane: The shaded points are the support vectors.

Let's turn our attention to the training data  $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$  with  $y_i \in \{+1, -1\}$ . For each point  $\mathbf{x}_i$  we can find its distance to the hyperplane by (20.15):

$$r_i = \frac{y_i h(\mathbf{x}_i)}{\|\mathbf{w}\|} = \frac{y_i (\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|} \quad (20.22)$$

Note that since  $h(\mathbf{x}_i) < 0$  if  $y_i = -1$  and  $h(\mathbf{x}_i) > 0$  if  $y_i = 1$ , we combine these into one equation above:  $y_i h(\mathbf{x}_i) \geq 0$ .

**Example:** Consider the hyperplane shown in Figure 20.2, given by the equation:

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

Consider the point  $\mathbf{x}_i = (2, 2.1)$  with class  $y_i = -1$ . We have,

$$r_i = \frac{y_i h(\mathbf{x}_i)}{\|\mathbf{w}\|} = \frac{-1 \left( (5 \ 2)(2 \ 2.1)^T - 20 \right)}{\sqrt{5^2 + 2^2}} = \frac{5.8}{\sqrt{29}} = 1.08$$

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

$$r^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|} \right\} \quad (20.23)$$

Let  $\mathbf{x}^*$  be the corresponding point, so that we have:

$$r^* = \frac{y_i(\mathbf{w}^T \mathbf{x}^* + b)}{\|\mathbf{w}\|}, \text{ or} \quad (20.24)$$

$$r^* \|\mathbf{w}\| = y_i(\mathbf{w}^T \mathbf{x}^* + b) \quad (20.25)$$

Now we can rescale  $\mathbf{w}$ , so that the margin is given as

$$r^* \|\mathbf{w}\| = 1 \quad \text{or} \quad r^* = \frac{1}{\|\mathbf{w}\|} \quad (20.26)$$

In other words, we are rescaling the distance of the closest point  $\mathbf{x}^*$  so that  $y_i(\mathbf{w}^T \mathbf{x}^* + b) = y_i h(\mathbf{x}^*) = 1$ . In fact, the set of points  $\mathbf{x}_i$  such that  $y_i h(\mathbf{x}_i) = 1$  are those that set the margin for the classifier, and are called the *support vectors*. For all other points that are not support vectors, we have  $y_i h(\mathbf{x}_i) > 1$ , since by definition, they must be farther away than the support vectors.

**Example:** Continuing our example from above,  $\mathbf{x}^* = (2 \ 2.1)^T$ , with  $r^* = 1.08$ . We would like to scale  $\mathbf{w}$  by some factor  $s$ , so that we have:

$$\begin{aligned} -1 \left( (5s \ 2s)(2 \ 2.1)^T - 20 \right) &= 1 \\ -14.2s + 20 &= 1 \\ s &= \frac{19}{14.2} = 1.338 \end{aligned}$$

Thus the scaled weight vector is given as  $\mathbf{w} = \begin{pmatrix} 5 \times 1.338 \\ 2 \times 1.338 \end{pmatrix} = \begin{pmatrix} 6.69 \\ 2.676 \end{pmatrix}$ . The new margin is therefore given as:

$$r^* = \frac{1}{\|\mathbf{w}\|} = \frac{1}{\sqrt{6.69^2 + 2.676^2}} = \frac{1}{7.205} = 0.139$$

Figure 20.2 gives an illustration of the margin. Since every point in the dataset is at a distance of  $\frac{1}{\|\mathbf{w}\|}$  or more, we have the following set of inequalities:

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \text{ for all points } \mathbf{x}_i \in \mathcal{D} \quad (20.27)$$

**Maximum Margin SVM:** Our goal now is to find the hyperplane that maximizes the margin  $\frac{1}{\|\mathbf{w}\|}$ , subject to the  $n$  constraints given in (20.27). Instead of maximizing  $\frac{1}{\|\mathbf{w}\|}$ , we obtain an equivalent formulation if we minimize  $\|\mathbf{w}\|$ . In fact, we can obtain an equivalent minimization formulation given as follows:

$$\textbf{Objective Function:} \quad \min \frac{\|\mathbf{w}\|^2}{2} \quad (20.28)$$

$$\textbf{Linear Constraints:} \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall \mathbf{x}_i \in \mathcal{D} \quad (20.29)$$

We can solve the above minimization problem with linear constraints, using the method of *Lagrange multipliers*, which turn it into an unconstrained optimization problem. The main idea is to introduce a Lagrange multiplier  $\alpha_i$  for each constraint, based on the Karush-Kuhn-Tucker (KKT) conditions:

$$\alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0 \quad (20.30)$$

$$\text{and } \alpha_i \geq 0 \quad (20.31)$$

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

$$L_{\text{primal}} = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) \quad (20.32)$$

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

$$\frac{\partial}{\partial \mathbf{w}} L_{\text{primal}} = \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0 \quad \text{or} \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (20.33)$$

$$\frac{\partial}{\partial b} L_{\text{primal}} = \sum_{i=1}^n \alpha_i y_i = 0 \quad (20.34)$$

Plugging these into (20.32), we obtain the *dual Lagrangian* objective function, which is purely in terms of the Lagrange multipliers, as follows:

$$L_{\text{dual}} = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \mathbf{w}^T \left( \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right) - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i \quad (20.35)$$

$$= -\frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i \quad (20.36)$$

$$= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (20.37)$$

$L_{\text{dual}}$  is a convex quadratic programming problem (note the  $\alpha_i \alpha_j$  terms), which can be solved using standard optimization techniques.

**Weight Vector and Bias:** Once we have obtained the  $\alpha_i$  values for  $i = 1, \dots, n$ , we can solve for the weight vector  $\mathbf{w}$  and the bias  $b$ . Note first that according to the KKT condition (20.30), we have

$$\alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0 \quad (20.38)$$

which gives rise to two cases:

- i.)  $\alpha_i = 0$ , or
- ii.)  $y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$ , which gives  $y_i(\mathbf{w}^T \mathbf{x}_i + b) = 1$

This is a very important result, since it states that  $\alpha_i = 0$  for all points that are at a distance of more than  $\frac{1}{\|\mathbf{w}\|}$  from the hyperplane, and only for those points that are exactly at the margin, we have  $\alpha_i > 0$ . In fact, the vast majority of the points will have  $\alpha_i = 0$ , and we only have to deal with the few points (with  $\alpha_i > 0$ ) that actually define the margin; these points are also called the *support vectors*, as illustrated in Figure 20.2.

We can easily compute the weight vector  $\mathbf{w}$  using (20.33), by taking the summation only for the support vectors:

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \quad (20.39)$$

In other words,  $\mathbf{w}$  is obtained as a linear combination of the support vectors, with the  $\alpha_i$  representing the weightings.

To compute the bias  $b$ , we get one solution per support vector, as follows:

$$\alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0 \quad (20.40)$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1 \quad (20.41)$$

$$b = \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i \quad (20.42)$$

We can take  $b$  as the average value over all the support vectors.

**SVM Classifier:** Our final SVM model is given as follows. For any new point  $\mathbf{z}$ , we predict the class as:

$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{z} + b) \quad (20.43)$$

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

### 20.3 Linear SVM: Soft Margin Hyperplanes

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 20.3.

SVMs can handle such a set of points with overlapping classes by introducing *slack variables* in (20.27), as follows:

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad (20.44)$$

Here  $\xi_i \geq 0$  is the new slack variable for point  $\mathbf{x}_i$ . First note that if  $\xi_i = 0$ , then the point is treated the same way as before. In other words that point is at least  $\frac{1}{\|\mathbf{w}\|}$  away from the hyperplane. Now if  $0 < \xi_i < 1$ , then the point is still correctly classified, since it will remain on the right side of the hyperplane. Finally, if  $\xi_i \geq 1$  then the point is misclassified, since in this case it appears on the wrong side of the hyperplane.

The goal of classification now is to find the hyperplane (given by  $\mathbf{w}$  and  $b$ ) with the maximum margin, that also minimizes the sum of the slack variables. In other words, our new objective function becomes:

$$\textbf{Objective Function:} \quad \min \frac{\|\mathbf{w}\|^2}{2} + C \left( \sum_{i=1}^n \xi_i \right)^k \quad (20.45)$$

$$\textbf{Linear Constraints:} \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \in \mathcal{D} \quad (20.46)$$

$$\xi_i \geq 0 \quad \forall \mathbf{x}_i \in \mathcal{D} \quad (20.47)$$

where  $C$  and  $k$  are constants, that incorporate the cost of misclassification.  $C$  is usually set empirically, whereas  $k$  is typically 1 or 2.

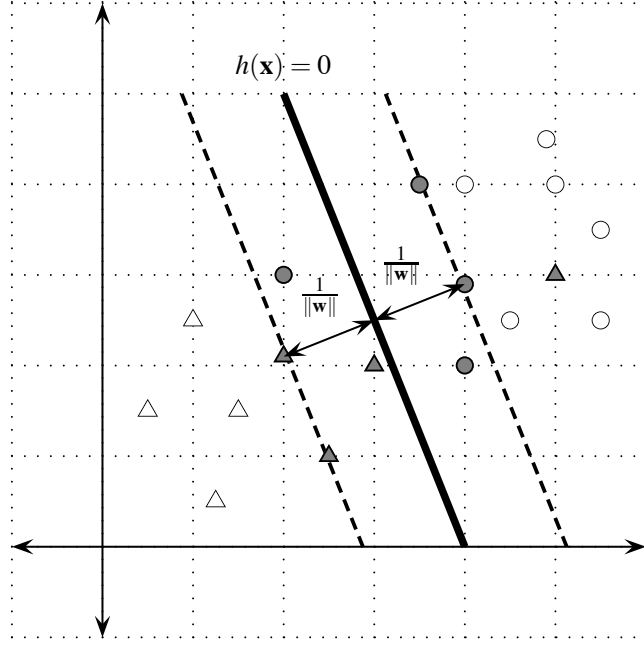


Figure 20.3: Soft Margin Hyperplane: The shaded points are the support vectors.

Assuming that  $k = 1$ , we can compute the Lagrangian for the above optimization problem by introducing Lagrange multipliers  $\alpha_i$  and  $\beta_i$  as follows:

$$\alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) = 0 \text{ with } \alpha_i \geq 0 \quad (20.48)$$

$$\beta_i (\xi_i - 0) = 0 \text{ with } \beta_i \geq 0 \quad (20.49)$$

The Lagrangian is then given as:

$$L_{primal} = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^n \beta_i \xi_i \quad (20.50)$$

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

$$\frac{\partial}{\partial \mathbf{w}} L_{primal} = \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0 \text{ or } \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (20.51)$$

$$\frac{\partial}{\partial b} L_{primal} = \sum_{i=1}^n \alpha_i y_i = 0 \quad (20.52)$$

$$\frac{\partial}{\partial \xi_i} L_{primal} = C - \alpha_i - \beta_i = 0 \text{ or } \beta_i = C - \alpha_i \quad (20.53)$$

Plugging these values into (20.50), we get:

$$L_{dual} = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \mathbf{w}^T \left( \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right) - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n (\alpha_i + \beta_i) \xi_i \quad (20.54)$$

$$= -\frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n (\alpha_i + C - \alpha_i) \xi_i \quad (20.55)$$

$$= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (20.56)$$

Notice that (20.56) is exactly the same as the dual Lagrangian in the linearly separable case (20.37). The only difference is the constraint on the  $\alpha_i$ , since we now require that  $\alpha_i + \beta_i = C$ , which means that  $0 \leq \alpha_i \leq C$ .

**Weight Vector and Bias:** Once we solve for  $\alpha_i$ , we have the same situation as before, namely that  $\alpha_i > 0$  only for the support vectors, which comprise of all points  $\mathbf{x}_i$  for which we have:

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1 - \xi_i \quad (20.57)$$

We can obtain the weight vector as before:

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \quad (20.58)$$

We can also solve for the  $\beta_i$  using (20.53)

$$\beta_i = C - \alpha_i \quad (20.59)$$

Replacing  $\beta_i$  in the KKT condition (20.49), with the expression from above, we obtain:

$$(C - \alpha_i) \xi_i = 0 \quad (20.60)$$

Thus for the support vectors ( $\alpha_i > 0$ ), we have two cases to consider: either  $C - \alpha_i = 0$ , or  $C - \alpha_i > 0$ , which means that  $\alpha_i < C$ . In this case, we must have  $\xi_i = 0$ , so we can solve for  $b$  as follows:

$$\alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0 \quad (20.61)$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1 \quad (20.62)$$

$$b = \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i \quad (20.63)$$

We can take the average over all such  $b$  values. The final SVM model predicts the class for a new point  $\mathbf{z}$  as follows

$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{z} + b) \quad (20.64)$$

## 20.4 Nonlinear SVM

If the dataset consists of points that are not linearly separable, then we can use the so called *kernel trick* to apply SVM in a non-linear setting.

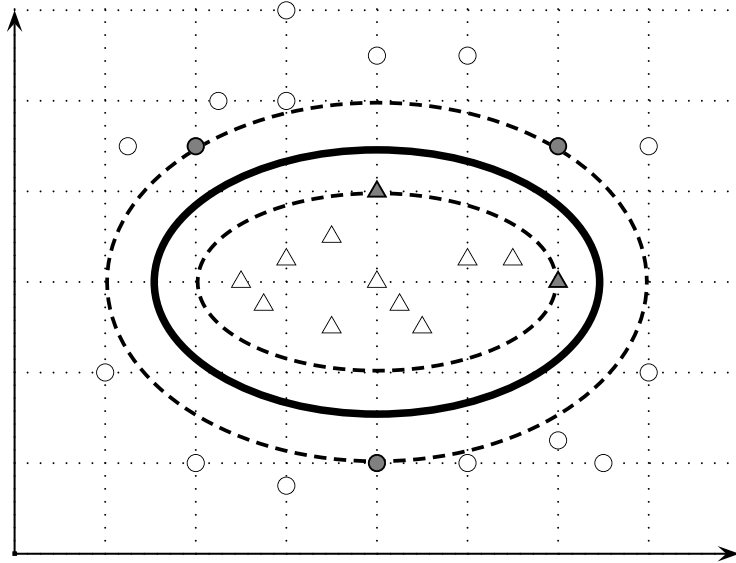


Figure 20.4: Nonlinear SVM: Shaded points are the support vectors.

Consider the set of points shown in Figure 20.4. In this case, there is no linear classifier that can discriminate between the points. However, there exists a perfect quadratic classifier that can separate the two classes. We will describe how we can apply the linear SVM (conceptually, in a transformed space) to solve problems with a non-linear decision boundary. The main idea is to map the original  $d$ -dimensional points into a higher dimensional space with  $d' > d$  dimensions, where the points can in fact be linearly separated. After finding the linear decision surface in  $d'$  dimensions, we can then map it back to the non-linear surface in the original  $d$ -dimensional space. For example, for the example shown in Figure 20.4, each point has two dimensions  $(X, Y)$ . If we map these points into the six dimensional space consisting of the new dimensions:  $(1, X, Y, X^2, Y^2, XY)$ , we can find a perfect linear hyperplane in the transformed space.

Given the original database  $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ , and the transformation function  $\phi$ , we obtain the new dataset in the transformed space  $\mathcal{D}_\phi = \{\phi(\mathbf{x}_i), y_i\}_{i=1}^n$ . Let us directly classify the points in the transformed space, where we assume they are perfectly linearly separable.

Based on Section 20.2, we have the following objective function for the linear SVM:

$$\textbf{Objective Function:} \quad \min \frac{\|\mathbf{w}\|^2}{2} \quad (20.65)$$

$$\textbf{Linear Constraints:} \quad y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1, \quad \forall \mathbf{x}_i \in \mathcal{D} \quad (20.66)$$

where  $\mathbf{w}$  is the weight vector in the transformed space.

As before we can set up the dual Lagrangian as follows:

$$L_{dual} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \quad (20.67)$$

Notice that the dual Lagrangian depends on being able to compute the dot product between two vectors in the transformed space:  $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ .

We can solve for  $\mathbf{w}$  and  $b$  in the transformed space as follows. Let  $m$  denote the number of support

vectors, i.e., the number of points with  $\alpha_i > 0$ , then we have

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i) \quad (20.68)$$

Furthermore, we compute  $b$  as the average over the support vectors:

$$b = \frac{1}{m} \sum_{i=1}^m \frac{1}{y_i} - \mathbf{w}^T \sum_{i=1}^m \frac{\phi(\mathbf{x}_i)}{m} \quad (20.69)$$

However, substituting  $\mathbf{w}$  from above, we obtain the new expression for  $b$  as:

$$b = \frac{1}{m} \sum_{i=1}^m \frac{1}{y_i} - \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^m \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \quad (20.70)$$

Notice that  $b$  is some function of the dot product between two vectors in the transformed space:  $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ .

Our final linear SVM model in the transformed space predicts the class for a new point  $\mathbf{z}$ , by first transforming it, and then applying the decision rule:

$$\hat{y} = \text{sign}(\mathbf{w}^T \phi(\mathbf{z}) + b) \quad (20.71)$$

Once again, plugging in  $\mathbf{w}$  from above, we have:

$$\hat{y} = \text{sign} \left( \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z}) + b \right) \quad (20.72)$$

Once again  $\hat{y}$  requires being able to compute the dot product in the transformed space (since  $b$  is also a function of the dot product). We can thus summarize the main observation so far. Namely that to solve the dual Lagrangian,  $b$ , and to classify points we only need to be able to compute the dot product between two points in the high dimensional space. We never need  $\phi(\mathbf{x})$  in isolation, especially since we never have to explicitly compute  $\mathbf{w}$ . Wherever  $\mathbf{w}$  appears, after substituting with  $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i)$  we always obtain a dot product.

### 20.4.1 Kernel Approach

As we saw above, we never need to compute  $\phi(\mathbf{x})$  in isolation. The only operation required in the transformed space is the ability to compute dot products  $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ . Define the kernel function between two points  $\mathbf{x}_i$  and  $\mathbf{x}_j$  in the original space as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \quad (20.73)$$

Wherever the dot product appears in (20.67, 20.70, 20.72) we replace it with the kernel  $K(\mathbf{x}_i, \mathbf{x}_j)$ .

What we would really like is to replace the dot product in the transformed space, with some function of the dot product in the original space. This way we never have to transform any point, and neither do we have to compute a dot product in the transformed space. There is a special class of kernel functions, the so called *Mercer Kernels*, that can be represented as some function of the dot product in the original space.

**Theorem 20.4.1** A function  $K : D \times D \rightarrow \mathbb{R}$  is called a **Positive Semi-Definite Kernel** iff it is symmetric

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

and positive semi-definite

$$\sum_{i=1}^n \sum_{j=1}^n a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

for any non-empty set of objects  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \subseteq D$  and for any choice of corresponding real numbers  $\{a_1, a_2, \dots, a_n\}$ .

Some commonly used kernel functions used in SVMs include:

- **Polynomial Kernel:**

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^q \quad (20.74)$$

where  $q$  is the degree of the polynomial.

For example, consider the following feature transformation from  $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^6$ ,

$$\phi(\mathbf{x} = (x, y)) = (1, \sqrt{2}x, \sqrt{2}y, x^2, y^2, \sqrt{2}xy) \quad (20.75)$$

The kernel in this case is given as:

$$K(\mathbf{x}, \mathbf{x}') = K((x, y), (x', y')) \quad (20.76)$$

$$= \phi(x, y)^T \phi(x', y') \quad (20.77)$$

$$= (1, \sqrt{2}x, \sqrt{2}y, x^2, y^2, \sqrt{2}xy)(1, \sqrt{2}x', \sqrt{2}y', x'^2, y'^2, \sqrt{2}x'y')^T \quad (20.78)$$

$$= 1 + 2xx' + 2yy' + x^2x'^2 + y^2y'^2 + 2xx'yy' \quad (20.79)$$

$$= 1^2 + 2(xx' + yy') + (xx' + yy')^2 \quad (20.80)$$

$$= (1 + \mathbf{x}^T \mathbf{x}')^2 \quad (20.81)$$

In other words, the dot product in the transformed space, can be expressed as a polynomial kernel in the original space.

- **Gaussian Kernel:**

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}} \quad (20.82)$$

where  $\sigma$  is the spread or standard deviation.

## 20.5 Exercises and Projects

1. Using Figure 20.5, and assuming the following kernel function:

$$K(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

Let the gradient vector at a point  $x$  using neighborhood  $N(x)$  be computed as:

$$\nabla f(x) = \sum_{x_i \in N(x)} (x_i - x) K\left(\frac{x - x_i}{h}\right)$$