9
Support Vector Machines

In this chapter, we discuss the *support vector machine* (SVM), an approach for classification that was developed in the computer science community in the 1990s and that has grown in popularity since then. SVMs have been shown to perform well in a variety of settings, and are often considered one of the best "out of the box" classifiers.

The support vector machine is a generalization of a simple and intuitive classifier called the *maximal margin classifier*, which we introduce in Section 9.1. Though it is elegant and simple, we will see that this classifier unfortunately cannot be applied to most data sets, since it requires that the classes be separable by a linear boundary. In Section 9.2, we introduce the *support vector classifier*, an extension of the maximal margin classifier that can be applied in a broader range of cases. Section 9.3 introduces the *support vector machine*, which is a further extension of the support vector classifier in order to accommodate non-linear class boundaries. Support vector machines are intended for the binary classification setting in which there are two classes; in Section 9.4 we discuss extensions of support vector machines to the case of more than two classes. In Section 9.5 we discuss the close connections between support vector machines and other statistical methods such as logistic regression.

People often loosely refer to the maximal margin classifier, the support vector classifier, and the support vector machine as "support vector machines". To avoid confusion, we will carefully distinguish between these three notions in this chapter.

9.1 Maximal Margin Classifier

In this section, we define a hyperplane and introduce the concept of an optimal separating hyperplane.

9.1.1 What Is a Hyperplane?

In a p-dimensional space, a *hyperplane* is a flat affine subspace of dimension $p - 1$.[1] For instance, in two dimensions, a hyperplane is a flat one-dimensional subspace—in other words, a line. In three dimensions, a hyperplane is a flat two-dimensional subspace—that is, a plane. In $p > 3$ dimensions, it can be hard to visualize a hyperplane, but the notion of a $(p - 1)$-dimensional flat subspace still applies.

hyperplane

The mathematical definition of a hyperplane is quite simple. In two dimensions, a hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \qquad (9.1)$$

> The equation of hyperplane.
> in 2 dimensions

for parameters β_0, β_1, and β_2. When we say that (9.1) "defines" the hyperplane, we mean that any $X = (X_1, X_2)^T$ for which (9.1) holds is a point on the hyperplane. Note that (9.1) is simply the equation of a line, since indeed in two dimensions a hyperplane is a line.

Equation 9.1 can be easily extended to the p-dimensional setting:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0 \qquad (9.2)$$

defines a p-dimensional hyperplane, again in the sense that if a point $X = (X_1, X_2, \ldots, X_p)^T$ in p-dimensional space (i.e. a vector of length p) satisfies (9.2), then X lies on the hyperplane.

Now, suppose that X does not satisfy (9.2); rather,

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0. \qquad (9.3)$$

Then this tells us that X lies to one side of the hyperplane. On the other hand, if

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0, \qquad (9.4)$$

then X lies on the other side of the hyperplane. So we can think of the hyperplane as dividing p-dimensional space into two halves. One can easily determine on which side of the hyperplane a point lies by simply calculating the sign of the left hand side of (9.2). A hyperplane in two-dimensional space is shown in Figure 9.1.

---

[1] The word *affine* indicates that the subspace need not pass through the origin.

The hyperplane $\tilde{\beta_0} + \tilde{\beta_1}X_1 + \tilde{\beta_2}X_2 > 0$

$\tilde{\beta_0} + \tilde{\beta_1}X_1 + \tilde{\beta_2}X_2 < 0$
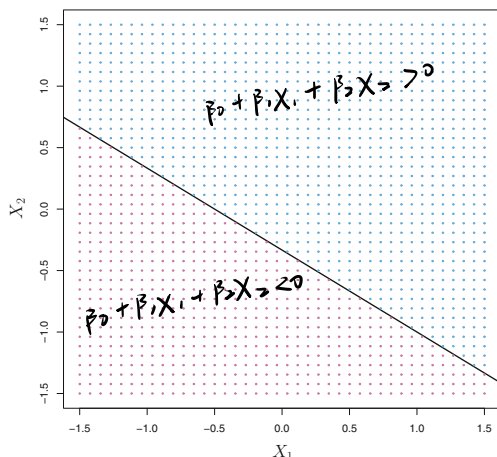
FIGURE 9.1. *The hyperplane* $1 + 2X_1 + 3X_2 = 0$ *is shown. The blue region is the set of points for which* $1 + 2X_1 + 3X_2 > 0$, *and the purple region is the set of points for which* $1 + 2X_1 + 3X_2 < 0$.

9.1.2 *Classification Using a Separating Hyperplane*

Now suppose that we have a $n \times p$ data matrix $\mathbf{X}$ that consists of n training observations in p-dimensional space,

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}, \tag{9.5}$$

and that these observations fall into two classes—that is, $y_1, \ldots, y_n \in \{-1, 1\}$ where -1 represents one class and 1 the other class. We also have a test observation, a p-vector of observed features $x^* = \begin{pmatrix} x_1^* & \cdots & x_p^* \end{pmatrix}^T$. Our goal is to develop a classifier based on the training data that will correctly classify the test observation using its feature measurements. We have seen a number of approaches for this task, such as linear discriminant analysis and logistic regression in Chapter 4, and classification trees, bagging, and boosting in Chapter 8. We will now see a new approach that is based upon the concept of a *separating hyperplane.*

Suppose that it is possible to construct a hyperplane that separates the training observations perfectly according to their class labels. Examples of three such *separating hyperplanes* are shown in the left-hand panel of Figure 9.2. We can label the observations from the blue class as $y_i = 1$ and

separating
hyperplane
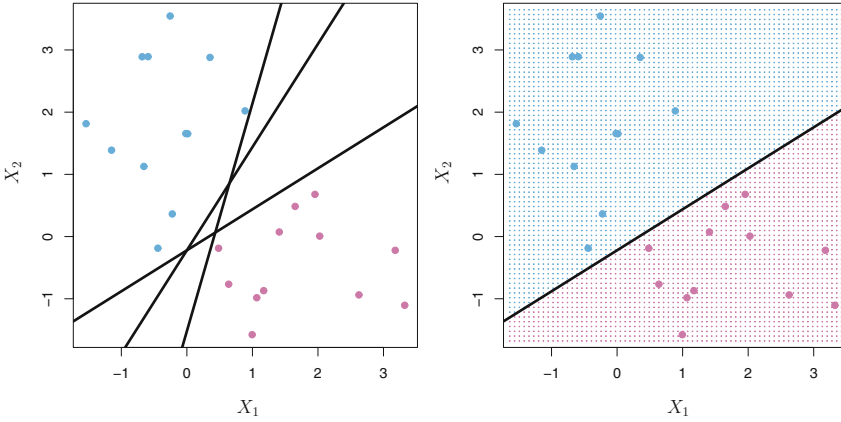
FIGURE 9.2. Left: *There are two classes of observations, shown in blue and in purple, each of which has measurements on two variables. Three separating hyperplanes, out of many possible, are shown in black.* Right: *A separating hyperplane is shown in black. The blue and purple grid indicates the decision rule made by a classifier based on this separating hyperplane: a test observation that falls in the blue portion of the grid will be assigned to the blue class, and a test observation that falls into the purple portion of the grid will be assigned to the purple class.*

those from the purple class as $y_i = -1$. Then a separating hyperplane has the property that

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0 \text{ if } y_i = 1, \tag{9.6}$$

and

$$\longrightarrow \text{put points of } -1 \text{ in to downside and put 1 into up side.}$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0 \text{ if } y_i = -1. \tag{9.7}$$

Equivalently, a separating hyperplane has the property that

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0 \tag{9.8}$$

for all $i = 1, \ldots, n$.

If a separating hyperplane exists, we can use it to construct a very natural classifier: a test observation is assigned a class depending on which side of the hyperplane it is located. The right-hand panel of Figure 9.2 shows an example of such a classifier. That is, we classify the test observation x^* based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^*$. If $f(x^*)$ is positive, then we assign the test observation to class 1, and if $f(x^*)$ is negative, then we assign it to class -1. We can also make use of the *magnitude* of $f(x^*)$. If $f(x^*)$ is far from zero, then this means that x^* lies far from the hyperplane, and so we can be confident about our class assignment for x^*. On the other

hand, if $f(x^*)$ is close to zero, then x^* is located near the hyperplane, and so we are less certain about the class assignment for x^*. Not surprisingly, and as we see in Figure 9.2, a classifier that is based on a separating hyperplane leads to a linear decision boundary.

9.1.3 The Maximal Margin Classifier

In general, if our data can be perfectly separated using a hyperplane, then there will in fact exist an infinite number of such hyperplanes. This is because a given separating hyperplane can usually be shifted a tiny bit up or down, or rotated, without coming into contact with any of the observations. Three possible separating hyperplanes are shown in the left-hand panel of Figure 9.2. In order to construct a classifier based upon a separating hyperplane, we must have a reasonable way to decide which of the infinite possible separating hyperplanes to use.

A natural choice is the *maximal margin hyperplane* (also known as the *optimal separating hyperplane*), which is the separating hyperplane that is farthest from the training observations. That is, we can compute the (perpendicular) distance from each training observation to a given separating hyperplane; the smallest such distance is the minimal distance from the observations to the hyperplane, and is known as the *margin*. The maximal margin hyperplane is the separating hyperplane for which the margin is largest—that is, it is the hyperplane that has the farthest minimum distance to the training observations. We can then classify a test observation based on which side of the maximal margin hyperplane it lies. This is known as the *maximal margin classifier.* We hope that a classifier that has a large margin on the training data will also have a large margin on the test data, and hence will classify the test observations correctly. Although the maximal margin classifier is often successful, it can also lead to overfitting when p is large.

If $\beta_0, \beta_1, \ldots, \beta_p$ are the coefficients of the maximal margin hyperplane, then the maximal margin classifier classifies the test observation x^* based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^*$.

Figure 9.3 shows the maximal margin hyperplane on the data set of Figure 9.2. Comparing the right-hand panel of Figure 9.2 to Figure 9.3, we see that the maximal margin hyperplane shown in Figure 9.3 does indeed result in a greater minimal distance between the observations and the separating hyperplane—that is, a larger margin. In a sense, the maximal margin hyperplane represents the mid-line of the widest "slab" that we can insert between the two classes.

Examining Figure 9.3, we see that three training observations are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin. These three observations are known as

maximal
margin
hyperplane
optimal
separating
hyperplane

margin

maximal
margin
classifier
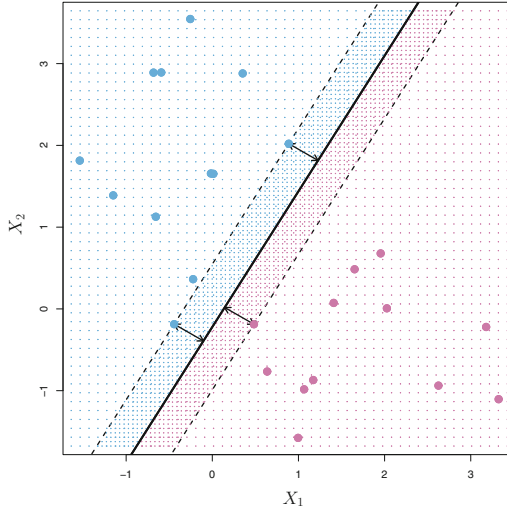
FIGURE 9.3. *There are two classes of observations, shown in blue and in purple. The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the support vectors, and the distance from those points to the hyperplane is indicated by arrows. The purple and blue grid indicates the decision rule made by a classifier based on this separating hyperplane.*

support vectors, since they are vectors in p-dimensional space (in Figure 9.3, $p = 2$) and they "support" the maximal margin hyperplane in the sense that if these points were moved slightly then the maximal margin hyperplane would move as well. Interestingly, the maximal margin hyperplane depends directly on the support vectors, but not on the other observations: a movement to any of the other observations would not affect the separating hyperplane, provided that the observation's movement does not cause it to cross the boundary set by the margin. The fact that the maximal margin hyperplane depends directly on only a small subset of the observations is an important property that will arise later in this chapter when we discuss the support vector classifier and support vector machines.

support vector

9.1.4 *Construction of the Maximal Margin Classifier*

We now consider the task of constructing the maximal margin hyperplane based on a set of n training observations $x_1, \ldots, x_n \in \mathbb{R}^p$ and associated class labels $y_1, \ldots, y_n \in \{-1, 1\}$. Briefly, the maximal margin hyperplane is the solution to the optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{\text{maximize}} \ M \tag{9.9}$$

$$\text{subject to} \ \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.10}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \ \ \forall \ i = 1,\ldots,n. \tag{9.11}$$

This optimization problem (9.9)–(9.11) is actually simpler than it looks. First of all, the constraint in (9.11) that

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \ \ \forall \ i = 1,\ldots,n$$

guarantees that each observation will be on the correct side of the hyperplane, provided that M is positive. (Actually, for each observation to be on the correct side of the hyperplane we would simply need $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0$, so the constraint in (9.11) in fact requires that each observation be on the correct side of the hyperplane, with some cushion, provided that M is positive.)

Second, note that (9.10) is not really a constraint on the hyperplane, since if $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} = 0$ defines a hyperplane, then so does $k(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) = 0$ for any $k \neq 0$. However, (9.10) adds meaning to (9.11); one can show that with this constraint the perpendicular distance from the ith observation to the hyperplane is given by

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}).$$

Therefore, the constraints (9.10) and (9.11) ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane. Hence, M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \beta_1, \ldots, \beta_p$ to maximize M. This is exactly the definition of the maximal margin hyperplane! The problem (9.9)–(9.11) can be solved efficiently, but details of this optimization are outside of the scope of this book.

9.1.5 The Non-separable Case

The maximal margin classifier is a very natural way to perform classification, *if a separating hyperplane exists*. However, as we have hinted, in many cases no separating hyperplane exists, and so there is no maximal margin classifier. In this case, the optimization problem (9.9)–(9.11) has no solution with $M > 0$. An example is shown in Figure 9.4. In this case, we cannot *exactly* separate the two classes. However, as we will see in the next section, we can extend the concept of a separating hyperplane in order to develop a hyperplane that *almost* separates the classes, using a so-called *soft margin*. The generalization of the maximal margin classifier to the non-separable case is known as the *support vector classifier*.
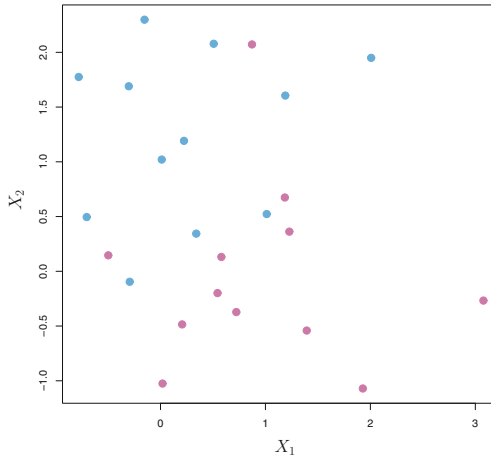
FIGURE 9.4. *There are two classes of observations, shown in blue and in purple. In this case, the two classes are not separable by a hyperplane, and so the maximal margin classifier cannot be used.*

9.2 Support Vector Classifiers

9.2.1 Overview of the Support Vector Classifier

In Figure 9.4, we see that observations that belong to two classes are not necessarily separable by a hyperplane. In fact, even if a separating hyperplane does exist, then there are instances in which a classifier based on a separating hyperplane might not be desirable. A classifier based on a separating hyperplane will necessarily perfectly classify all of the training observations; this can lead to sensitivity to individual observations. An example is shown in Figure 9.5. The addition of a single observation in the right-hand panel of Figure 9.5 leads to a dramatic change in the maximal margin hyperplane. The resulting maximal margin hyperplane is not satisfactory—for one thing, it has only a tiny margin. This is problematic because as discussed previously, the distance of an observation from the hyperplane can be seen as a measure of our confidence that the observation was correctly classified. Moreover, the fact that the maximal margin hyperplane is extremely sensitive to a change in a single observation suggests that it may have overfit the training data.

In this case, we might be willing to consider a classifier based on a hyperplane that does *not* perfectly separate the two classes, in the interest of
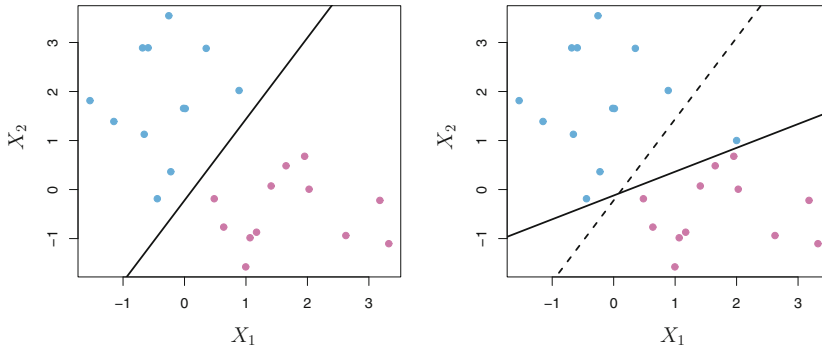
FIGURE 9.5. Left: *Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane.* Right: *An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line. The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.*

- Greater robustness to individual observations, and

- Better classification of *most* of the training observations.

That is, it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.

The *support vector classifier*, sometimes called a *soft margin classifier*, does exactly this. Rather than seeking the largest possible margin so that every observation is not only on the correct side of the hyperplane but also on the correct side of the margin, we instead allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane. (The margin is *soft* because it can be violated by some of the training observations.) An example is shown in the left-hand panel of Figure 9.6. Most of the observations are on the correct side of the margin. However, a small subset of the observations are on the wrong side of the margin.

support vector classifier

soft margin classifier

An observation can be not only on the wrong side of the margin, but also on the wrong side of the hyperplane. In fact, when there is no separating hyperplane, such a situation is inevitable. Observations on the wrong side of the hyperplane correspond to training observations that are misclassified by the support vector classifier. The right-hand panel of Figure 9.6 illustrates such a scenario.

9.2.2   *Details of the Support Vector Classifier*

The support vector classifier classifies a test observation depending on which side of a hyperplane it lies. The hyperplane is chosen to correctly
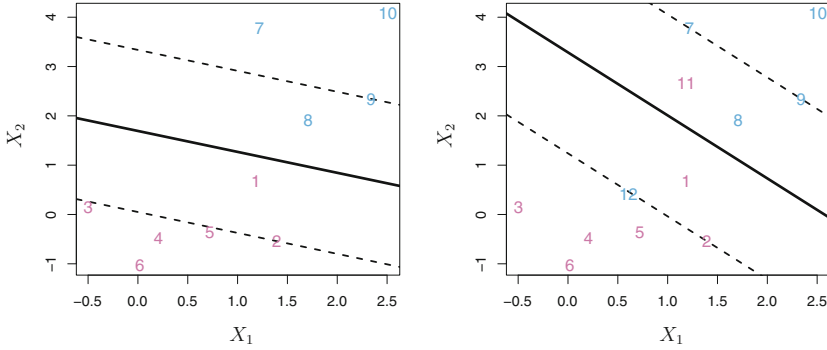
FIGURE 9.6. Left: *A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines. Purple observations: Observations* 3, 4, 5, *and* 6 *are on the correct side of the margin, observation* 2 *is on the margin, and observation* 1 *is on the wrong side of the margin.* Blue observations: *Observations* 7 *and* 10 *are on the correct side of the margin, observation* 9 *is on the margin, and observation* 8 *is on the wrong side of the margin. No observations are on the wrong side of the hyperplane.* Right: *Same as left panel with two additional points,* 11 *and* 12. *These two observations are on the wrong side of the hyperplane and the wrong side of the margin.*

separate most of the training observations into the two classes, but may misclassify a few observations. It is the solution to the optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} M \tag{9.12}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.13}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \tag{9.14}$$

$$\epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq C, \tag{9.15}$$

where C is a nonnegative tuning parameter. As in (9.11), M is the width of the margin; we seek to make this quantity as large as possible. In (9.14), $\epsilon_1, \ldots, \epsilon_n$ are *slack variables* that allow individual observations to be on the wrong side of the margin or the hyperplane; we will explain them in greater detail momentarily. Once we have solved (9.12)–(9.15), we classify a test observation x^* as before, by simply determining on which side of the hyperplane it lies. That is, we classify the test observation based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \ldots + \beta_p x_p^*$.

slack variable

The problem (9.12)–(9.15) seems complex, but insight into its behavior can be made through a series of simple observations presented below. First of all, the slack variable ϵ_i tells us where the ith observation is located, relative to the hyperplane and relative to the margin. If $\epsilon_i = 0$ then the ith

observation is on the correct side of the margin, as we saw in Section 9.1.4. If $\epsilon_i > 0$ then the ith observation is on the wrong side of the margin, and we say that the ith observation has *violated* the margin. If $\epsilon_i > 1$ then it is on the wrong side of the hyperplane.

We now consider the role of the tuning parameter C. In (9.15), C bounds the sum of the ϵ_i's, and so it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate. We can think of C as a *budget* for the amount that the margin can be violated by the n observations. If $C = 0$ then there is no budget for violations to the margin, and it must be the case that $\epsilon_1 = \ldots = \epsilon_n = 0$, in which case (9.12)–(9.15) simply amounts to the maximal margin hyperplane optimization problem (9.9)–(9.11). (Of course, a maximal margin hyperplane exists only if the two classes are separable.) For $C > 0$ no more than C observations can be on the wrong side of the hyperplane, because if an observation is on the wrong side of the hyperplane then $\epsilon_i > 1$, and (9.15) requires that $\sum_{i=1}^{n} \epsilon_i \leq C$. As the budget C increases, we become more tolerant of violations to the margin, and so the margin will widen. Conversely, as C decreases, we become less tolerant of violations to the margin and so the margin narrows. An example in shown in Figure 9.7.

In practice, C is treated as a tuning parameter that is generally chosen via cross-validation. As with the tuning parameters that we have seen throughout this book, C controls the bias-variance trade-off of the statistical learning technique. When C is small, we seek narrow margins that are rarely violated; this amounts to a classifier that is highly fit to the data, which may have low bias but high variance. On the other hand, when C is larger, the margin is wider and we allow more violations to it; this amounts to fitting the data less hard and obtaining a classifier that is potentially more biased but may have lower variance.

The optimization problem (9.12)–(9.15) has a very interesting property: it turns out that only observations that either lie on the margin or that violate the margin will affect the hyperplane, and hence the classifier obtained. In other words, an observation that lies strictly on the correct side of the margin does not affect the support vector classifier! Changing the position of that observation would not change the classifier at all, provided that its position remains on the correct side of the margin. Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as *support vectors*. These observations do affect the support vector classifier.

The fact that only support vectors affect the classifier is in line with our previous assertion that C controls the bias-variance trade-off of the support vector classifier. When the tuning parameter C is large, then the margin is wide, many observations violate the margin, and so there are many support vectors. In this case, many observations are involved in determining the hyperplane. The top left panel in Figure 9.7 illustrates this setting: this classifier has low variance (since many observations are support vectors)
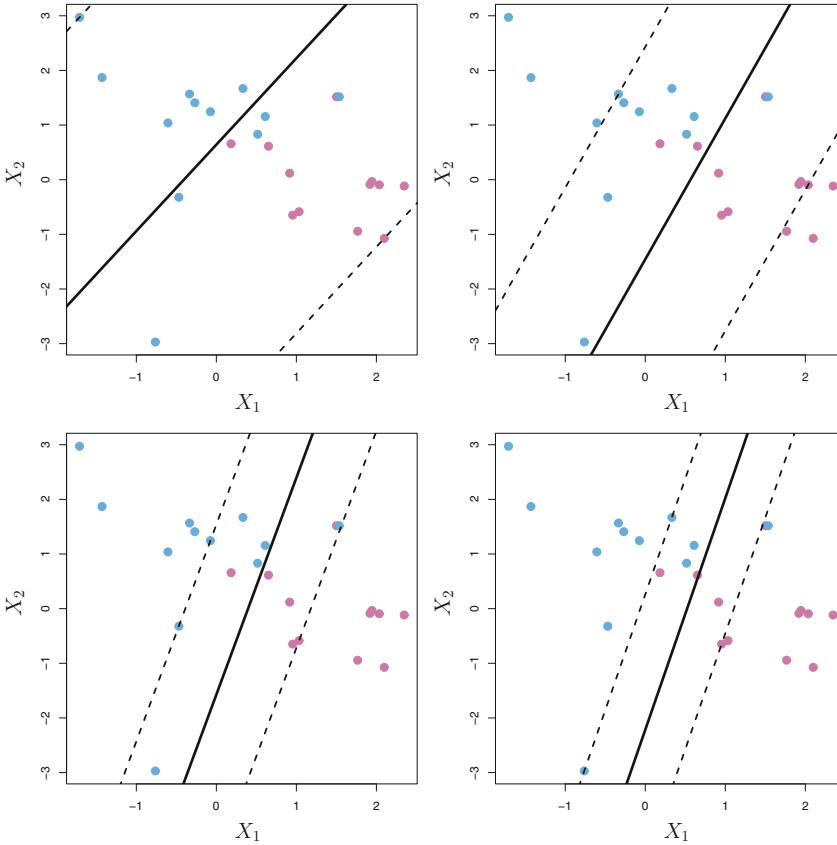
FIGURE 9.7. *A support vector classifier was fit using four different values of the tuning parameter C in (9.12)–(9.15). The largest value of C was used in the top left panel, and smaller values were used in the top right, bottom left, and bottom right panels. When C is large, then there is a high tolerance for observations being on the wrong side of the margin, and so the margin will be large. As C decreases, the tolerance for observations being on the wrong side of the margin decreases, and the margin narrows.*

but potentially high bias. In contrast, if C is small, then there will be fewer support vectors and hence the resulting classifier will have low bias but high variance. The bottom right panel in Figure 9.7 illustrates this setting, with only eight support vectors.

The fact that the support vector classifier's decision rule is based only on a potentially small subset of the training observations (the support vectors) means that it is quite robust to the behavior of observations that are far away from the hyperplane. This property is distinct from some of the other classification methods that we have seen in preceding chapters, such as linear discriminant analysis. Recall that the LDA classification rule
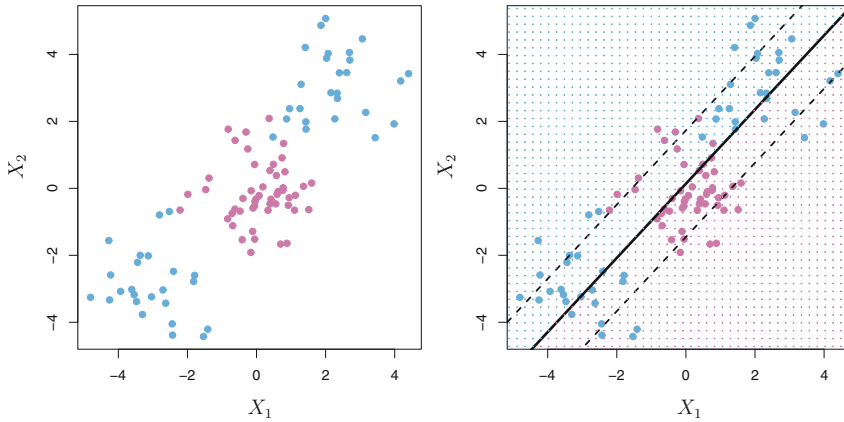
FIGURE 9.8. Left: *The observations fall into two classes, with a non-linear boundary between them.* Right: *The support vector classifier seeks a linear boundary, and consequently performs very poorly.*

depends on the mean of *all* of the observations within each class, as well as the within-class covariance matrix computed using *all* of the observations. In contrast, logistic regression, unlike LDA, has very low sensitivity to observations far from the decision boundary. In fact we will see in Section 9.5 that the support vector classifier and logistic regression are closely related.

9.3   Support Vector Machines

We first discuss a general mechanism for converting a linear classifier into one that produces non-linear decision boundaries. We then introduce the support vector machine, which does this in an automatic way.

9.3.1   Classification with Non-linear Decision Boundaries

The support vector classifier is a natural approach for classification in the two-class setting, if the boundary between the two classes is linear. However, in practice we are sometimes faced with non-linear class boundaries. For instance, consider the data in the left-hand panel of Figure 9.8. It is clear that a support vector classifier or any linear classifier will perform poorly here. Indeed, the support vector classifier shown in the right-hand panel of Figure 9.8 is useless here.

In Chapter 7, we are faced with an analogous situation. We see there that the performance of linear regression can suffer when there is a non-linear relationship between the predictors and the outcome. In that case, we consider enlarging the feature space using functions of the predictors,

such as quadratic and cubic terms, in order to address this non-linearity. In the case of the support vector classifier, we could address the problem of possibly non-linear boundaries between classes in a similar way, by enlarging the feature space using quadratic, cubic, and even higher-order polynomial functions of the predictors. For instance, rather than fitting a support vector classifier using p features

$$X_1, X_2, \ldots, X_p,$$

we could instead fit a support vector classifier using $2p$ features

$$X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2.$$

Then (9.12)–(9.15) would become

$$\underset{\beta_0, \beta_{11}, \beta_{12}, \ldots, \beta_{p1}, \beta_{p2}, \epsilon_1, \ldots, \epsilon_n, M}{\text{maximize}} M \tag{9.16}$$

$$\text{subject to } y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i),$$

$$\sum_{i=1}^{n} \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1.$$

Why does this lead to a non-linear decision boundary? In the enlarged feature space, the decision boundary that results from (9.16) is in fact linear. But in the original feature space, the decision boundary is of the form $q(x) = 0$, where q is a quadratic polynomial, and its solutions are generally non-linear. One might additionally want to enlarge the feature space with higher-order polynomial terms, or with interaction terms of the form $X_j X_{j'}$ for $j \neq j'$. Alternatively, other functions of the predictors could be considered rather than polynomials. It is not hard to see that there are many possible ways to enlarge the feature space, and that unless we are careful, we could end up with a huge number of features. Then computations would become unmanageable. The support vector machine, which we present next, allows us to enlarge the feature space used by the support vector classifier in a way that leads to efficient computations.

9.3.2 The Support Vector Machine

The *support vector machine* (SVM) is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using *kernels*. We will now discuss this extension, the details of which are somewhat complex and beyond the scope of this book. However, the main idea is described in Section 9.3.1: we may want to enlarge our feature space

support
vector
machine
kernel

in order to accommodate a non-linear boundary between the classes. The kernel approach that we describe here is simply an efficient computational approach for enacting this idea.

We have not discussed exactly how the support vector classifier is computed because the details become somewhat technical. However, it turns out that the solution to the support vector classifier problem (9.12)–(9.15) involves only the *inner products* of the observations (as opposed to the observations themselves). The inner product of two r-vectors a and b is defined as $\langle a, b \rangle = \sum_{i=1}^{r} a_i b_i$. Thus the inner product of two observations x_i, $x_{i'}$ is given by

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}. \tag{9.17}$$

It can be shown that

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle, \tag{9.18}$$

  where there are n parameters α_i, $i = 1, \ldots, n$, one per training observation.

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations. (The notation $\binom{n}{2}$ means $n(n-1)/2$, and gives the number of pairs among a set of n items.)

Notice that in (9.18), in order to evaluate the function $f(x)$, we need to compute the inner product between the new point x and each of the training points x_i. However, it turns out that α_i is nonzero only for the support vectors in the solution—that is, if a training observation is not a support vector, then its α_i equals zero. So if $\mathcal{S}$ is the collection of indices of these support points, we can rewrite any solution function of the form (9.18) as

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle, \tag{9.19}$$

which typically involves far fewer terms than in (9.18).[2]

To summarize, in representing the linear classifier $f(x)$, and in computing its coefficients, all we need are inner products.

Now suppose that every time the inner product (9.17) appears in the representation (9.18), or in a calculation of the solution for the support

---

[2] By expanding each of the inner products in (9.19), it is easy to see that $f(x)$ is a linear function of the coordinates of x. Doing so also establishes the correspondence between the α_i and the original parameters β_j.

vector classifier, we replace it with a *generalization* of the inner product of the form

$$K(x_i, x_{i'}),\qquad(9.20)$$

where K is some function that we will refer to as a *kernel*. A kernel is a function that quantifies the similarity of two observations. For instance, we could simply take

kernel

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j},\qquad(9.21)$$

which would just give us back the support vector classifier. Equation 9.21 is known as a *linear* kernel because the support vector classifier is linear in the features; the linear kernel essentially quantifies the similarity of a pair of observations using Pearson (standard) correlation. But one could instead choose another form for (9.20). For instance, one could replace every instance of $\sum_{j=1}^{p} x_{ij} x_{i'j}$ with the quantity

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d.\qquad(9.22)$$

→ polynomial K func.

This is known as a *polynomial kernel* of degree d, where d is a positive integer. Using such a kernel with $d > 1$, instead of the standard linear kernel (9.21), in the support vector classifier algorithm leads to a much more flexible decision boundary. It essentially amounts to fitting a support vector classifier in a higher-dimensional space involving polynomials of degree d, rather than in the original feature space. When the support vector classifier is combined with a non-linear kernel such as (9.22), the resulting classifier is known as a support vector machine. Note that in this case the (non-linear) function has the form

polynomial kernel

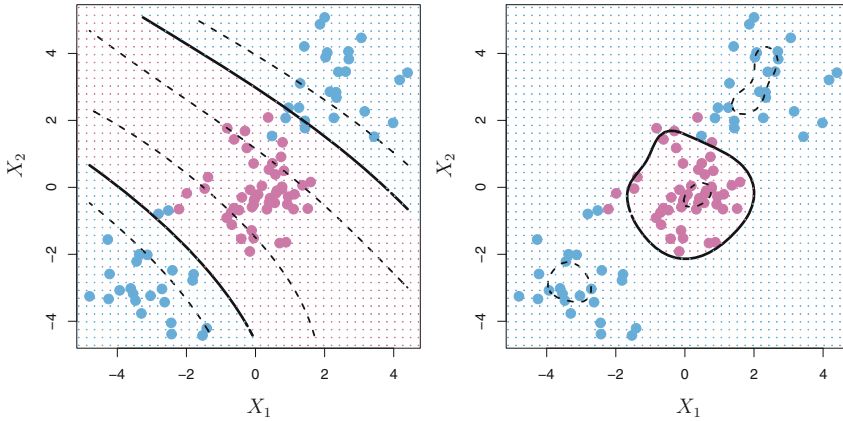$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i).\qquad(9.23)$$

The left-hand panel of Figure 9.9 shows an example of an SVM with a polynomial kernel applied to the non-linear data from Figure 9.8. The fit is a substantial improvement over the linear support vector classifier. When $d = 1$, then the SVM reduces to the support vector classifier seen earlier in this chapter.

The polynomial kernel shown in (9.22) is one example of a possible non-linear kernel, but alternatives abound. Another popular choice is the *radial kernel*, which takes the form

radial kernel

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).\qquad(9.24)$$

FIGURE 9.9. Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

In (9.24), γ is a positive constant. The right-hand panel of Figure 9.9 shows an example of an SVM with a radial kernel on this non-linear data; it also does a good job in separating the two classes.

How does the radial kernel (9.24) actually work? If a given test observation $x^* = (x_1^* \ldots x_p^*)^T$ is far from a training observation x_i in terms of Euclidean distance, then $\sum_{j=1}^{p}(x_j^* - x_{ij})^2$ will be large, and so $K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p}(x_j^* - x_{ij})^2)$ will be very tiny. This means that in (9.23), x_i will play virtually no role in $f(x^*)$. Recall that the predicted class label for the test observation x^* is based on the sign of $f(x^*)$. In other words, training observations that are far from x^* will play essentially no role in the predicted class label for x^*. This means that the radial kernel has very *local* behavior, in the sense that only nearby training observations have an effect on the class label of a test observation.

What is the advantage of using a kernel rather than simply enlarging the feature space using functions of the original features, as in (9.16)? One advantage is computational, and it amounts to the fact that using kernels, one need only compute $K(x_i, x_{i'})$ for all $\binom{n}{2}$ distinct pairs i, i'. This can be done without explicitly working in the enlarged feature space. This is important because in many applications of SVMs, the enlarged feature space is so large that computations are intractable. For some kernels, such as the radial kernel (9.24), the feature space is *implicit* and infinite-dimensional, so we could never do the computations there anyway!
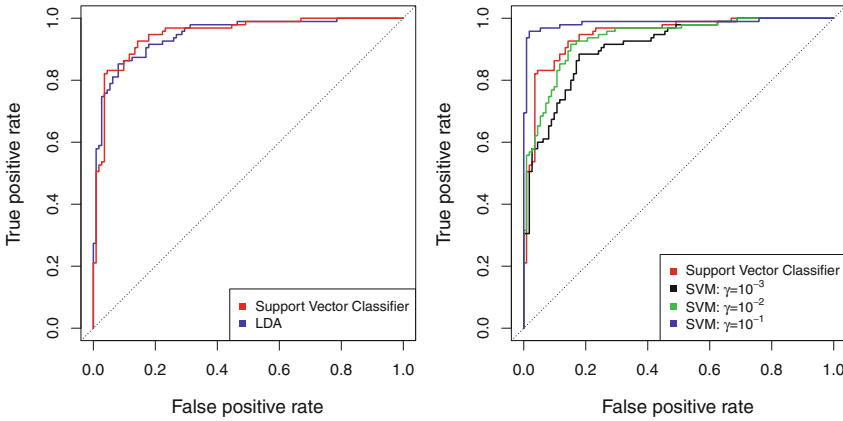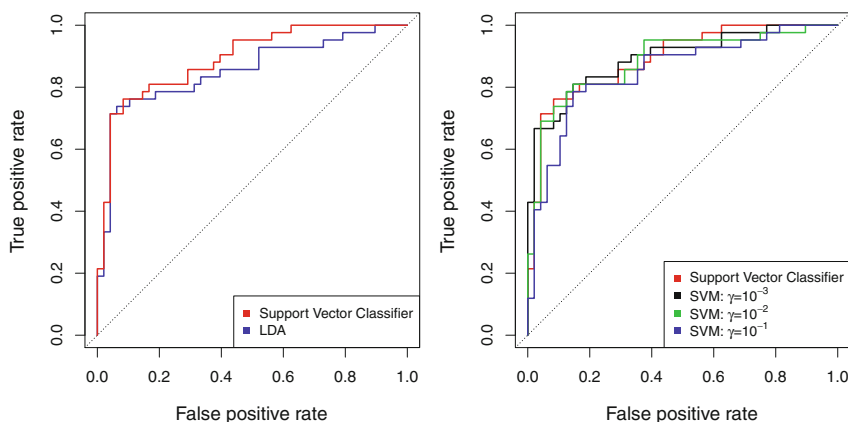
FIGURE 9.10. *ROC curves for the* `Heart` *data training set.* Left: *The support vector classifier and LDA are compared.* Right: *The support vector classifier is compared to an SVM using a radial basis kernel with* $\gamma = 10^{-3}$, 10^{-2}, *and* 10^{-1}.

9.3.3   An Application to the Heart Disease Data

In Chapter 8 we apply decision trees and related methods to the `Heart` data. The aim is to use 13 predictors such as `Age`, `Sex`, and `Chol` in order to predict whether an individual has heart disease. We now investigate how an SVM compares to LDA on this data. After removing 6 missing observations, the data consist of 297 subjects, which we randomly split into 207 training and 90 test observations.

We first fit LDA and the support vector classifier to the training data. Note that the support vector classifier is equivalent to a SVM using a polynomial kernel of degree $d = 1$. The left-hand panel of Figure 9.10 displays ROC curves (described in Section 4.4.3) for the training set predictions for both LDA and the support vector classifier. Both classifiers compute scores of the form $\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \ldots + \hat{\beta}_p X_p$ for each observation. For any given cutoff t, we classify observations into the *heart disease* or *no heart disease* categories depending on whether $\hat{f}(X) < t$ or $\hat{f}(X) \geq t$. The ROC curve is obtained by forming these predictions and computing the false positive and true positive rates for a range of values of t. An optimal classifier will hug the top left corner of the ROC plot. In this instance LDA and the support vector classifier both perform well, though there is a suggestion that the support vector classifier may be slightly superior.

The right-hand panel of Figure 9.10 displays ROC curves for SVMs using a radial kernel, with various values of γ. As γ increases and the fit becomes more non-linear, the ROC curves improve. Using $\gamma = 10^{-1}$ appears to give an almost perfect ROC curve. However, these curves represent training error rates, which can be misleading in terms of performance on new test data. Figure 9.11 displays ROC curves computed on the 90 test observa-

FIGURE 9.11. *ROC curves for the test set of the* `Heart` *data. Left: The support vector classifier and LDA are compared. Right: The support vector classifier is compared to an SVM using a radial basis kernel with $\gamma = 10^{-3}$, 10^{-2}, and 10^{-1}.*

tions. We observe some differences from the training ROC curves. In the left-hand panel of Figure 9.11, the support vector classifier appears to have a small advantage over LDA (although these differences are not statistically significant). In the right-hand panel, the SVM using $\gamma = 10^{-1}$, which showed the best results on the training data, produces the worst estimates on the test data. This is once again evidence that while a more flexible method will often produce lower training error rates, this does not necessarily lead to improved performance on test data. The SVMs with $\gamma = 10^{-2}$ and $\gamma = 10^{-3}$ perform comparably to the support vector classifier, and all three outperform the SVM with $\gamma = 10^{-1}$.

9.4 SVMs with More than Two Classes

So far, our discussion has been limited to the case of binary classification: that is, classification in the two-class setting. How can we extend SVMs to the more general case where we have some arbitrary number of classes? It turns out that the concept of separating hyperplanes upon which SVMs are based does not lend itself naturally to more than two classes. Though a number of proposals for extending SVMs to the K-class case have been made, the two most popular are the *one-versus-one* and *one-versus-all* approaches. We briefly discuss those two approaches here.

9.4.1 One-Versus-One Classification

Suppose that we would like to perform classification using SVMs, and there are $K > 2$ classes. A *one-versus-one* or *all-pairs* approach constructs $\binom{K}{2}$

one-versus-one

SVMs, each of which compares a pair of classes. For example, one such SVM might compare the kth class, coded as $+1$, to the k'th class, coded as -1. We classify a test observation using each of the $\binom{K}{2}$ classifiers, and we tally the number of times that the test observation is assigned to each of the K classes. The final classification is performed by assigning the test observation to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.

9.4.2   One-Versus-All Classification

The *one-versus-all* approach is an alternative procedure for applying SVMs in the case of $K > 2$ classes. We fit K SVMs, each time comparing one of the K classes to the remaining $K - 1$ classes. Let $\beta_{0k}, \beta_{1k}, \ldots, \beta_{pk}$ denote the parameters that result from fitting an SVM comparing the kth class (coded as $+1$) to the others (coded as -1). Let x^* denote a test observation. We assign the observation to the class for which $\beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* + \ldots + \beta_{pk}x_p^*$ is largest, as this amounts to a high level of confidence that the test observation belongs to the kth class rather than to any of the other classes.

one-versus-all

9.5   Relationship to Logistic Regression

When SVMs were first introduced in the mid-1990s, they made quite a splash in the statistical and machine learning communities. This was due in part to their good performance, good marketing, and also to the fact that the underlying approach seemed both novel and mysterious. The idea of finding a hyperplane that separates the data as well as possible, while allowing some violations to this separation, seemed distinctly different from classical approaches for classification, such as logistic regression and linear discriminant analysis. Moreover, the idea of using a kernel to expand the feature space in order to accommodate non-linear class boundaries appeared to be a unique and valuable characteristic.

However, since that time, deep connections between SVMs and other more classical statistical methods have emerged. It turns out that one can rewrite the criterion (9.12)–(9.15) for fitting the support vector classifier $f(X) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$ as

$$\underset{\beta_0, \beta_1, \ldots, \beta_p}{\text{minimize}} \left\{ \sum_{i=1}^{n} \max\left[0, 1 - y_i f(x_i)\right] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}, \qquad (9.25)$$

where λ is a nonnegative tuning parameter. When λ is large then $\beta_1,\ldots,\beta_p$ are small, more violations to the margin are tolerated, and a low-variance but high-bias classifier will result. When λ is small then few violations to the margin will occur; this amounts to a high-variance but low-bias classifier. Thus, a small value of λ in (9.25) amounts to a small value of C in (9.15). Note that the $\lambda \sum_{j=1}^{p} \beta_j^2$ term in (9.25) is the ridge penalty term from Section 6.2.1, and plays a similar role in controlling the bias-variance trade-off for the support vector classifier.

Now (9.25) takes the "Loss + Penalty" form that we have seen repeatedly throughout this book:

$$\operatorname*{minimize}_{\beta_0,\beta_1,\ldots,\beta_p} \left\{ L(\mathbf{X},\mathbf{y},\beta) + \lambda P(\beta) \right\}. \tag{9.26}$$

In (9.26), $L(\mathbf{X},\mathbf{y},\beta)$ is some loss function quantifying the extent to which the model, parametrized by β, fits the data $(\mathbf{X},\mathbf{y})$, and $P(\beta)$ is a penalty function on the parameter vector β whose effect is controlled by a nonnegative tuning parameter λ. For instance, ridge regression and the lasso both take this form with

$$L(\mathbf{X},\mathbf{y},\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

and with $P(\beta) = \sum_{j=1}^{p} \beta_j^2$ for ridge regression and $P(\beta) = \sum_{j=1}^{p} |\beta_j|$ for the lasso. In the case of (9.25) the loss function instead takes the form

$$L(\mathbf{X},\mathbf{y},\beta) = \sum_{i=1}^{n} \max\left[0, 1 - y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})\right].$$

This is known as *hinge loss*, and is depicted in Figure 9.12. However, it turns out that the hinge loss function is closely related to the loss function used in logistic regression, also shown in Figure 9.12.

hinge loss

An interesting characteristic of the support vector classifier is that only support vectors play a role in the classifier obtained; observations on the correct side of the margin do not affect it. This is due to the fact that the loss function shown in Figure 9.12 is exactly zero for observations for which $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq 1$; these correspond to observations that are on the correct side of the margin.[3] In contrast, the loss function for logistic regression shown in Figure 9.12 is not exactly zero anywhere. But it is very small for observations that are far from the decision boundary. Due to the similarities between their loss functions, logistic regression and the support vector classifier often give very similar results. When the classes are well separated, SVMs tend to behave better than logistic regression; in more overlapping regimes, logistic regression is often preferred.

---

[3] With this hinge-loss + penalty representation, the margin corresponds to the value one, and the width of the margin is determined by $\sum \beta_j^2$.
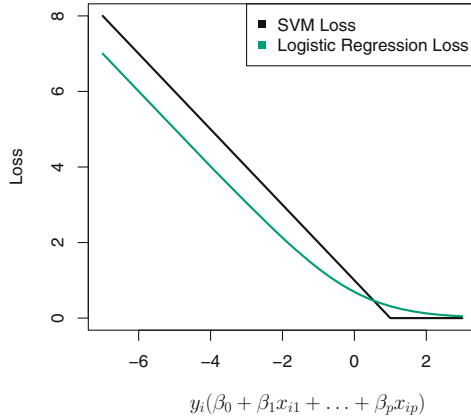
FIGURE 9.12. *The SVM and logistic regression loss functions are compared, as a function of $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$. When $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$ is greater than 1, then the SVM loss is zero, since this corresponds to an observation that is on the correct side of the margin. Overall, the two loss functions have quite similar behavior.*

When the support vector classifier and SVM were first introduced, it was thought that the tuning parameter C in (9.15) was an unimportant "nuisance" parameter that could be set to some default value, like 1. However, the "Loss + Penalty" formulation (9.25) for the support vector classifier indicates that this is not the case. The choice of tuning parameter is very important and determines the extent to which the model underfits or overfits the data, as illustrated, for example, in Figure 9.7.

We have established that the support vector classifier is closely related to logistic regression and other preexisting statistical methods. Is the SVM unique in its use of kernels to enlarge the feature space to accommodate non-linear class boundaries? The answer to this question is "no". We could just as well perform logistic regression or many of the other classification methods seen in this book using non-linear kernels; this is closely related to some of the non-linear approaches seen in Chapter 7. However, for historical reasons, the use of non-linear kernels is much more widespread in the context of SVMs than in the context of logistic regression or other methods.

Though we have not addressed it here, there is in fact an extension of the SVM for regression (i.e. for a quantitative rather than a qualitative response), called *support vector regression*. In Chapter 3, we saw that least squares regression seeks coefficients $\beta_0, \beta_1, \ldots, \beta_p$ such that the sum of squared residuals is as small as possible. (Recall from Chapter 3 that residuals are defined as $y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip}$.) Support vector regression instead seeks coefficients that minimize a different type of loss, where only residuals larger in absolute value than some positive constant

support
vector
regression

contribute to the loss function. This is an extension of the margin used in support vector classifiers to the regression setting.

9.6   Lab: Support Vector Machines

We use the `e1071` library in `R` to demonstrate the support vector classifier and the SVM. Another option is the `LiblineaR` library, which is useful for very large linear problems.

9.6.1   Support Vector Classifier

The `e1071` library contains implementations for a number of statistical learning methods. In particular, the `svm()` function can be used to fit a support vector classifier when the argument `kernel="linear"` is used. This `svm()` function uses a slightly different formulation from (9.14) and (9.25) for the support vector classifier. A `cost` argument allows us to specify the cost of a violation to the margin. When the `cost` argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. When the `cost` argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin.

We now use the `svm()` function to fit the support vector classifier for a given value of the `cost` parameter. Here we demonstrate the use of this function on a two-dimensional example so that we can plot the resulting decision boundary. We begin by generating the observations, which belong to two classes, and checking whether the classes are linearly separable.

```
> set.seed(1)
> x=matrix(rnorm(20*2), ncol=2)
> y=c(rep(-1,10), rep(1,10))
> x[y==1,]=x[y==1,] + 1
> plot(x, col=(3-y))
```

They are not. Next, we fit the support vector classifier. Note that in order for the `svm()` function to perform classification (as opposed to SVM-based regression), we must encode the response as a factor variable. We now create a data frame with the response coded as a factor.

```
> dat=data.frame(x=x, y=as.factor(y))
> library(e1071)
> svmfit=svm(y~., data=dat, kernel="linear", cost=10,
    scale=FALSE)
```

The argument `scale=FALSE` tells the `svm()` function not to scale each feature to have mean zero or standard deviation one; depending on the application, one might prefer to use `scale=TRUE`.

We can now plot the support vector classifier obtained:

```
> plot(svmfit, dat)
```

Note that the two arguments to the `plot.svm()` function are the output of the call to `svm()`, as well as the data used in the call to `svm()`. The region of feature space that will be assigned to the −1 class is shown in light blue, and the region that will be assigned to the +1 class is shown in purple. The decision boundary between the two classes is linear (because we used the argument `kernel="linear"`), though due to the way in which the plotting function is implemented in this library the decision boundary looks somewhat jagged in the plot. (Note that here the second feature is plotted on the x-axis and the first feature is plotted on the y-axis, in contrast to the behavior of the usual `plot()` function in R.) The support vectors are plotted as crosses and the remaining observations are plotted as circles; we see here that there are seven support vectors. We can determine their identities as follows:

```
> svmfit$index
[1]   1   2   5   7 14 16 17
```

We can obtain some basic information about the support vector classifier fit using the `summary()` command:

```
> summary(svmfit)
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
    scale = FALSE)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5
Number of Support Vectors:  7
 ( 4 3 )
Number of Classes:  2
Levels:
 -1 1
```

This tells us, for instance, that a linear kernel was used with `cost=10`, and that there were seven support vectors, four in one class and three in the other.

What if we instead used a smaller value of the cost parameter?

```
> svmfit=svm(y~., data=dat, kernel="linear", cost=0.1,
    scale=FALSE)
> plot(svmfit, dat)
> svmfit$index
[1]   1   2   3   4   5   7   9 10 12 13 14 15 16 17 18 20
```

Now that a smaller value of the cost parameter is being used, we obtain a larger number of support vectors, because the margin is now wider. Unfortunately, the `svm()` function does not explicitly output the coefficients of the linear decision boundary obtained when the support vector classifier is fit, nor does it output the width of the margin.

The `e1071` library includes a built-in function, `tune()`, to perform cross-validation. By default, `tune()` performs ten-fold cross-validation on a set of models of interest. In order to use this function, we pass in relevant information about the set of models that are under consideration. The following command indicates that we want to compare SVMs with a linear kernel, using a range of values of the `cost` parameter.

`tune()`

```
> set.seed(1)
> tune.out=tune(svm,y~.,data=dat,kernel="linear",
    ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
```

We can easily access the cross-validation errors for each of these models using the `summary()` command:

```
> summary(tune.out)
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
  0.1
- best performance: 0.1
- Detailed performance results:
   cost error  dispersion
1 1e-03  0.70      0.422
2 1e-02  0.70      0.422
3 1e-01  0.10      0.211
4 1e+00  0.15      0.242
5 5e+00  0.15      0.242
6 1e+01  0.15      0.242
7 1e+02  0.15      0.242
```

We see that `cost=0.1` results in the lowest cross-validation error rate. The `tune()` function stores the best model obtained, which can be accessed as follows:

```
> bestmod=tune.out$best.model
> summary(bestmod)
```

The `predict()` function can be used to predict the class label on a set of test observations, at any given value of the cost parameter. We begin by generating a test data set.

```
> xtest=matrix(rnorm(20*2), ncol=2)
> ytest=sample(c(-1,1), 20, rep=TRUE)
> xtest[ytest==1,]=xtest[ytest==1,] + 1
> testdat=data.frame(x=xtest, y=as.factor(ytest))
```

Now we predict the class labels of these test observations. Here we use the best model obtained through cross-validation in order to make predictions.

```
> ypred=predict(bestmod,testdat)
> table(predict=ypred, truth=testdat$y)
        truth
predict -1   1
     -1 11   1
      1   0   8
```

Thus, with this value of `cost`, 19 of the test observations are correctly classified. What if we had instead used `cost=0.01`?

```
> svmfit=svm(y~., data=dat, kernel="linear", cost=.01,
    scale=FALSE)
> ypred=predict(svmfit,testdat)
> table(predict=ypred, truth=testdat$y)
        truth
predict -1   1
     -1 11   2
      1   0   7
```

In this case one additional observation is misclassified.

Now consider a situation in which the two classes are linearly separable. Then we can find a separating hyperplane using the `svm()` function. We first further separate the two classes in our simulated data so that they are linearly separable:

```
> x[y==1,]=x[y==1,]+0.5
> plot(x, col=(y+5)/2, pch=19)
```

Now the observations are just barely linearly separable. We fit the support vector classifier and plot the resulting hyperplane, using a very large value of `cost` so that no observations are misclassified.

```
> dat=data.frame(x=x,y=as.factor(y))
> svmfit=svm(y~., data=dat, kernel="linear", cost=1e5)
> summary(svmfit)
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e
    +05)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e+05
      gamma:  0.5
Number of Support Vectors:  3
 ( 1 2 )
Number of Classes:  2
Levels:
 -1 1
> plot(svmfit, dat)
```

No training errors were made and only three support vectors were used. However, we can see from the figure that the margin is very narrow (because the observations that are not support vectors, indicated as circles, are very

close to the decision boundary). It seems likely that this model will perform poorly on test data. We now try a smaller value of `cost`:

```
> svmfit = svm (y~., data = dat , kernel = " linear ", cost =1)
> summary ( svmfit )
> plot ( svmfit , dat )
```

Using `cost=1`, we misclassify a training observation, but we also obtain a much wider margin and make use of seven support vectors. It seems likely that this model will perform better on test data than the model with `cost=1e5`.

9.6.2 *Support Vector Machine*

In order to fit an SVM using a non-linear kernel, we once again use the `svm()` function. However, now we use a different value of the parameter `kernel`. To fit an SVM with a polynomial kernel we use `kernel="polynomial"`, and to fit an SVM with a radial kernel we use `kernel="radial"`. In the former case we also use the `degree` argument to specify a degree for the polynomial kernel (this is d in (9.22)), and in the latter case we use `gamma` to specify a value of γ for the radial basis kernel (9.24).

We first generate some data with a non-linear class boundary, as follows:

```
> set . seed (1)
> x = matrix ( rnorm (200*2) , ncol =2)
> x [1:100 ,]= x [1:100 ,]+2
> x [101:150 ,]= x [101:150 ,] -2
> y=c ( rep (1 ,150) , rep (2 ,50))
> dat = data . frame (x=x ,y= as . factor (y))
```

Plotting the data makes it clear that the class boundary is indeed non-linear:

```
> plot (x , col =y)
```

The data is randomly split into training and testing groups. We then fit the training data using the `svm()` function with a radial kernel and $\gamma = 1$:

```
> train = sample (200 ,100)
> svmfit = svm (y~., data = dat [ train ,], kernel = " radial ",   gamma =1 ,
    cost =1)
> plot ( svmfit , dat [ train ,])
```

The plot shows that the resulting SVM has a decidedly non-linear boundary. The `summary()` function can be used to obtain some information about the SVM fit:

```
> summary ( svmfit )
Call :
svm ( formula = y ~ ., data = dat , kernel = " radial ",
    gamma = 1, cost = 1)
Parameters :
   SVM - Type :   C - classification
```

```
 SVM - Kernel :    radial
         cost :   1
        gamma :   1
 Number  of  Support  Vectors :   37
  ( 17 20 )
 Number  of  Classes :   2
 Levels :
  1 2
```

We can see from the figure that there are a fair number of training errors
in this SVM fit. If we increase the value of `cost`, we can reduce the number
of training errors. However, this comes at the price of a more irregular
decision boundary that seems to be at risk of overfitting the data.

```
> svmfit = svm ( y ~ . , data = dat [ train ,] , kernel = " radial " , gamma =1 ,
    cost =1 e5 )

> plot ( svmfit , dat [ train ,])
```

We can perform cross-validation using `tune()` to select the best choice of
γ and `cost` for an SVM with a radial kernel:

```
> set . seed (1)
> tune . out = tune ( svm , y ~ . , data = dat [ train ,] , kernel = " radial " ,
    ranges = list ( cost = c (0.1 ,1 ,10 ,100 ,1000) ,
    gamma = c (0.5 ,1 ,2 ,3 ,4) ) )
> summary ( tune . out )
Parameter  tuning  of  ' svm ':
- sampling  method :  10 - fold  cross  validation
- best  parameters :
 cost gamma
    1     2
- best  performance :  0.12
- Detailed  performance  results :
     cost gamma error dispersion
1   1e -01    0.5  0.27      0.1160
2   1e +00    0.5  0.13      0.0823
3   1e +01    0.5  0.15      0.0707
4   1e +02    0.5  0.17      0.0823
5   1e +03    0.5  0.21      0.0994
6   1e -01    1.0  0.25      0.1354
7   1e +00    1.0  0.13      0.0823
. . .
```

Therefore, the best choice of parameters involves `cost=1` and `gamma=2`. We
can view the test set predictions for this model by applying the `predict()`
function to the data. Notice that to do this we subset the dataframe `dat`
using `-train` as an index set.

```
> table ( true = dat [ - train ," y "] ,  pred = predict ( tune . out $best . model ,
    newdata = dat [ - train ,]) )
```

10 % of test observations are misclassified by this SVM.

9.6.3   ROC Curves

The `ROCR` package can be used to produce ROC curves such as those in
Figures 9.10 and 9.11. We first write a short function to plot an ROC curve
given a vector containing a numerical score for each observation, `pred`, and
a vector containing the class label for each observation, `truth`.

```
> library(ROCR)
> rocplot=function(pred, truth, ...){
+    predob = prediction(pred, truth)
+    perf = performance(predob, "tpr", "fpr")
+    plot(perf,...)}
```

   SVMs and support vector classifiers output class labels for each observa-
tion. However, it is also possible to obtain *fitted values* for each observation,
which are the numerical scores used to obtain the class labels. For instance,
in the case of a support vector classifier, the fitted value for an observation
$X = (X_1, X_2, \ldots, X_p)^T$ takes the form $\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \ldots + \hat{\beta}_p X_p$.
For an SVM with a non-linear kernel, the equation that yields the fitted
value is given in (9.23). In essence, the sign of the fitted value determines
on which side of the decision boundary the observation lies. Therefore, the
relationship between the fitted value and the class prediction for a given
observation is simple: if the fitted value exceeds zero then the observation
is assigned to one class, and if it is less than zero then it is assigned to the
other. In order to obtain the fitted values for a given SVM model fit, we
use `decision.values=TRUE` when fitting `svm()`. Then the `predict()` function
will output the fitted values.

```
> svmfit.opt=svm(y~., data=dat[train,], kernel="radial",
          gamma=2, cost=1,decision.values=T)
> fitted=attributes(predict(svmfit.opt,dat[train,],decision.
    values=TRUE))$decision.values
```

Now we can produce the ROC plot.

```
> par(mfrow=c(1,2))
> rocplot(fitted,dat[train,"y"],main="Training Data")
```

SVM appears to be producing accurate predictions. By increasing γ we can
produce a more flexible fit and generate further improvements in accuracy.

```
> svmfit.flex=svm(y~., data=dat[train,], kernel="radial",
     gamma=50, cost=1, decision.values=T)
> fitted=attributes(predict(svmfit.flex,dat[train,],decision.
    values=T))$decision.values
> rocplot(fitted,dat[train,"y"],add=T,col="red")
```

   However, these ROC curves are all on the training data. We are really
more interested in the level of prediction accuracy on the test data. When
we compute the ROC curves on the test data, the model with $\gamma = 2$ appears
to provide the most accurate results.

```
> fitted=attributes(predict(svmfit.opt,dat[-train,],decision.
    values=T))$decision.values
> rocplot(fitted,dat[-train,"y"],main="Test Data")
> fitted=attributes(predict(svmfit.flex,dat[-train,],decision.
    values=T))$decision.values
> rocplot(fitted,dat[-train,"y"],add=T,col="red")
```

9.6.4 SVM with Multiple Classes

If the response is a factor containing more than two levels, then the `svm()` function will perform multi-class classification using the one-versus-one approach. We explore that setting here by generating a third class of observations.

```
> set.seed(1)
> x=rbind(x, matrix(rnorm(50*2), ncol=2))
> y=c(y, rep(0,50))
> x[y==0,2]=x[y==0,2]+2
> dat=data.frame(x=x, y=as.factor(y))
> par(mfrow=c(1,1))
> plot(x,col=(y+1))
```

We now fit an SVM to the data:

```
> svmfit=svm(y~., data=dat, kernel="radial", cost=10, gamma=1)
> plot(svmfit, dat)
```

The `e1071` library can also be used to perform support vector regression, if the response vector that is passed in to `svm()` is numerical rather than a factor.

9.6.5 Application to Gene Expression Data

We now examine the `Khan` data set, which consists of a number of tissue samples corresponding to four distinct types of small round blue cell tumors. For each tissue sample, gene expression measurements are available. The data set consists of training data, `xtrain` and `ytrain`, and testing data, `xtest` and `ytest`.

We examine the dimension of the data:

```
> library(ISLR)
> names(Khan)
[1]   "xtrain"  "xtest"  "ytrain"  "ytest"
> dim(Khan$xtrain)
[1]   63 2308
> dim(Khan$xtest)
[1]   20 2308
> length(Khan$ytrain)
[1] 63
> length(Khan$ytest)
[1] 20
```

This data set consists of expression measurements for 2,308 genes. The training and test sets consist of 63 and 20 observations respectively.

```
> table(Khan$ytrain)
 1   2   3   4
 8  23  12  20
> table(Khan$ytest)
1 2 3 4
3 6 6 5
```

We will use a support vector approach to predict cancer subtype using gene expression measurements. In this data set, there are a very large number of features relative to the number of observations. This suggests that we should use a linear kernel, because the additional flexibility that will result from using a polynomial or radial kernel is unnecessary.

```
> dat=data.frame(x=Khan$xtrain, y=as.factor(Khan$ytrain))
> out=svm(y~., data=dat, kernel="linear",cost=10)
> summary(out)
Call:
svm(formula = y ~ ., data = dat, kernel = "linear",
    cost = 10)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.000433
Number of Support Vectors:  58
 ( 20 20 11 7 )
Number of Classes:  4
Levels:
 1 2 3 4
> table(out$fitted, dat$y)

      1   2   3   4
  1   8   0   0   0
  2   0  23   0   0
  3   0   0  12   0
  4   0   0   0  20
```

We see that there are *no* training errors. In fact, this is not surprising, because the large number of variables relative to the number of observations implies that it is easy to find hyperplanes that fully separate the classes. We are most interested not in the support vector classifier's performance on the training observations, but rather its performance on the test observations.

```
> dat.te=data.frame(x=Khan$xtest, y=as.factor(Khan$ytest))
> pred.te=predict(out, newdata=dat.te)
> table(pred.te, dat.te$y)

pred.te 1 2 3 4
      1 3 0 0 0
      2 0 6 2 0
      3 0 0 4 0
      4 0 0 0 5
```

We see that using `cost=10` yields two test set errors on this data.

9.7 Exercises

Conceptual

1. This problem involves hyperplanes in two dimensions.

    (a) Sketch the hyperplane $1 + 3X_1 - X_2 = 0$. Indicate the set of points for which $1 + 3X_1 - X_2 > 0$, as well as the set of points for which $1 + 3X_1 - X_2 < 0$.

    (b) On the same plot, sketch the hyperplane $-2 + X_1 + 2X_2 = 0$. Indicate the set of points for which $-2 + X_1 + 2X_2 > 0$, as well as the set of points for which $-2 + X_1 + 2X_2 < 0$.

2. We have seen that in $p = 2$ dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

    (a) Sketch the curve
    $$(1 + X_1)^2 + (2 - X_2)^2 = 4.$$

    (b) On your sketch, indicate the set of points for which
    $$(1 + X_1)^2 + (2 - X_2)^2 > 4,$$
    as well as the set of points for which
    $$(1 + X_1)^2 + (2 - X_2)^2 \leq 4.$$

    (c) Suppose that a classifier assigns an observation to the blue class if
    $$(1 + X_1)^2 + (2 - X_2)^2 > 4,$$
    and to the red class otherwise. To what class is the observation $(0, 0)$ classified? $(-1, 1)$? $(2, 2)$? $(3, 8)$?

    (d) Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2, X_2, and X_2^2.

3. Here we explore the maximal margin classifier on a toy data set.

    (a) We are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label.