is the fact that each regression problem will be easier to solve, because it involves only the examples with nonzero weight—the examples whose kernels overlap the query point. When kernel widths are small, this may be just a few points.

Most nonparametric models have the advantage that it is easy to do leave-one-out cross-validation without having to recompute everything. With a k-nearest-neighbors model, for instance, when given a test example $(\mathbf{x}, y)$ we retrieve the k nearest neighbors once, compute the per-example loss $L(y, h(\mathbf{x}))$ from them, and record that as the leave-one-out result for every example that is not one of the neighbors. Then we retrieve the $k + 1$ nearest neighbors and record distinct results for leaving out each of the k neighbors. With N examples the whole process is $O(k)$, not $O(kN)$.

18.9 SUPPORT VECTOR MACHINES

SUPPORT VECTOR MACHINE

The **support vector machine** or SVM framework is currently the most popular approach for "off-the-shelf" supervised learning: if you don't have any specialized prior knowledge about a domain, then the SVM is an excellent method to try first. There are three properties that make SVMs attractive:

1. SVMs construct a **maximum margin separator**—a decision boundary with the largest possible distance to example points. This helps them generalize well.
2. SVMs create a linear separating hyperplane, but they have the ability to embed the data into a higher-dimensional space, using the so-called **kernel trick**. Often, data that are not linearly separable in the original input space are easily separable in the higher-dimensional space. The high-dimensional linear separator is actually nonlinear in the original space. This means the hypothesis space is greatly expanded over methods that use strictly linear representations.
3. SVMs are a nonparametric method—they retain training examples and potentially need to store them all. On the other hand, in practice they often end up retaining only a small fraction of the number of examples—sometimes as few as a small constant times the number of dimensions. Thus SVMs combine the advantages of nonparametric and parametric models: they have the flexibility to represent complex functions, but they are resistant to overfitting.

You could say that SVMs are successful because of one key insight and one neat trick. We will cover each in turn. In Figure 18.30(a), we have a binary classification problem with three candidate decision boundaries, each a linear separator. Each of them is consistent with all the examples, so from the point of view of 0/1 loss, each would be equally good. Logistic regression would find some separating line; the exact location of the line depends on *all* the example points. The key insight of SVMs is that some examples are more important than others, and that paying attention to them can lead to better generalization.

Consider the lowest of the three separating lines in (a). It comes very close to 5 of the black examples. Although it classifies all the examples correctly, and thus minimizes loss, it

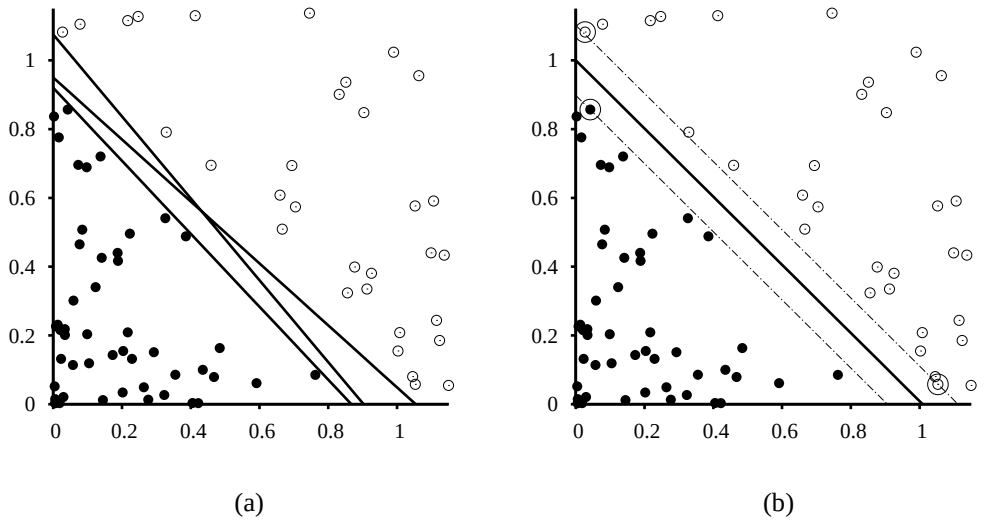

Figure 18.30 Support vector machine classification: (a) Two classes of points (black and white circles) and three candidate linear separators. (b) The maximum margin separator (heavy line), is at the midpoint of the **margin** (area between dashed lines). The **support vectors** (points with large circles) are the examples closest to the separator.

should make you nervous that so many examples are close to the line; it may be that other black examples will turn out to fall on the other side of the line.

SVMs address this issue: Instead of minimizing expected *empirical loss* on the training data, SVMs attempt to minimize expected *generalization* loss. We don't know where the as-yet-unseen points may fall, but under the probabilistic assumption that they are drawn from the same distribution as the previously seen examples, there are some arguments from computational learning theory (Section 18.5) suggesting that we minimize generalization loss by choosing the separator that is farthest away from the examples we have seen so far. We call this separator, shown in Figure 18.30(b) the **maximum margin separator**. The **margin** is the width of the area bounded by dashed lines in the figure—twice the distance from the separator to the nearest example point.

MAXIMUM MARGIN SEPARATOR

MARGIN

Now, how do we find this separator? Before showing the equations, some notation: Traditionally SVMs use the convention that class labels are +1 and -1, instead of the +1 and 0 we have been using so far. Also, where we put the intercept into the weight vector $\mathbf{w}$ (and a corresponding dummy 1 value into $x_{j,0}$), SVMs do not do that; they keep the intercept as a separate parameter, b. With that in mind, the separator is defined as the set of points $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$. We could search the space of $\mathbf{w}$ and b with gradient descent to find the parameters that maximize the margin while correctly classifying all the examples.

However, it turns out there is another approach to solving this problem. We won't show the details, but will just say that there is an alternative representation called the dual

representation, in which the optimal solution is found by solving

$$\underset{\alpha}{\text{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k) \tag{18.13}$$

QUADRATIC PROGRAMMING

subject to the constraints $\alpha_j \geq 0$ and $\sum_j \alpha_j y_j = 0$. This is a **quadratic programming** optimization problem, for which there are good software packages. Once we have found the vector α we can get back to $\mathbf{w}$ with the equation $\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$, or we can stay in the dual representation. There are three important properties of Equation (18.13). First, the expression is convex; it has a single global maximum that can be found efficiently. Second, *the data enter the expression only in the form of dot products of pairs of points.* This second property is also true of the equation for the separator itself; once the optimal α_j have been calculated, it is

$$h(\mathbf{x}) = \text{sign}\left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x}_j) - b\right) . \tag{18.14}$$

SUPPORT VECTOR

A final important property is that the weights α_j associated with each data point are *zero* except for the **support vectors**—the points closest to the separator. (They are called "support" vectors because they "hold up" the separating plane.) Because there are usually many fewer support vectors than examples, SVMs gain some of the advantages of parametric models.

What if the examples are not linearly separable? Figure 18.31(a) shows an input space defined by attributes $\mathbf{x} = (x_1, x_2)$, with positive examples ($y = +1$) inside a circular region and negative examples ($y = -1$) outside. Clearly, there is no linear separator for this problem. Now, suppose we re-express the input data—i.e., we map each input vector $\mathbf{x}$ to a new vector of feature values, $F(\mathbf{x})$. In particular, let us use the three features

$$f_1 = x_1^2 , \qquad f_2 = x_2^2 , \qquad f_3 = \sqrt{2} x_1 x_2 . \tag{18.15}$$

We will see shortly where these came from, but for now, just look at what happens. Figure 18.31(b) shows the data in the new, three-dimensional space defined by the three features; the data are *linearly separable* in this space! This phenomenon is actually fairly general: if data are mapped into a space of sufficiently high dimension, then they will almost always be linearly separable—if you look at a set of points from enough directions, you'll find a way to make them line up. Here, we used only three dimensions;[11] Exercise 18.16 asks you to show that four dimensions suffice for linearly separating a circle anywhere in the plane (not just at the origin), and five dimensions suffice to linearly separate any ellipse. In general (with some special cases excepted) if we have N data points then they will always be separable in spaces of $N - 1$ dimensions or more (Exercise 18.25).

Now, we would not usually expect to find a linear separator in the input space $\mathbf{x}$, but we can find linear separators in the high-dimensional feature space $F(\mathbf{x})$ simply by replacing $\mathbf{x}_j \cdot \mathbf{x}_k$ in Equation (18.13) with $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$. This by itself is not remarkable—replacing $\mathbf{x}$ by $F(\mathbf{x})$ in *any* learning algorithm has the required effect—but the dot product has some special properties. It turns out that $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$ can often be computed without first computing F

[11] The reader may notice that we could have used just f_1 and f_2, but the 3D mapping illustrates the idea better.

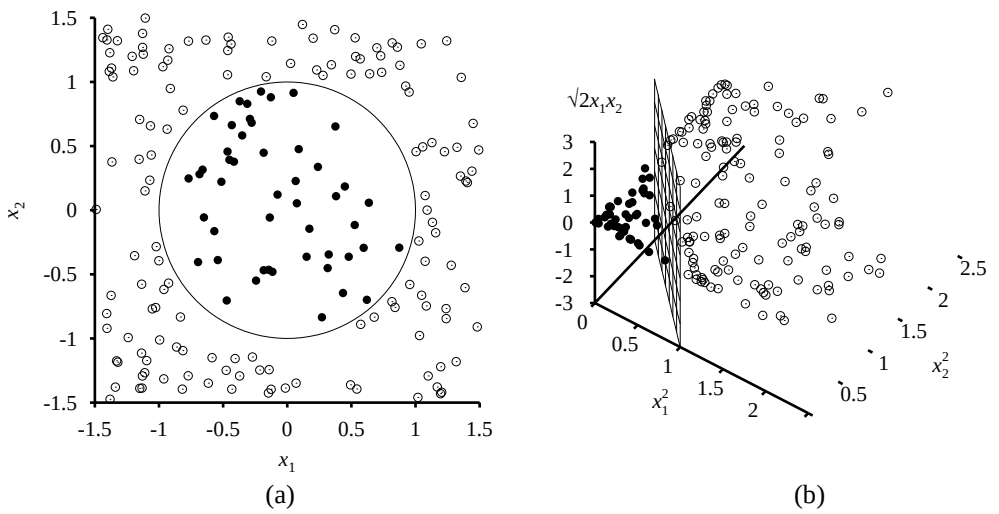

Figure 18.31 (a) A two-dimensional training set with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown. (b) The same data after mapping into a three-dimensional input space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in (a) becomes a linear decision boundary in three dimensions. Figure 18.30(b) gives a closeup of the separator in (b).

for each point. In our three-dimensional feature space defined by Equation (18.15), a little bit of algebra shows that

$$F(\mathbf{x}_j) \cdot F(\mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2 \ .$$

KERNEL FUNCTION

(That's why the $\sqrt{2}$ is in f_3.) The expression $(\mathbf{x}_j \cdot \mathbf{x}_k)^2$ is called a **kernel function**,[12] and is usually written as $K(\mathbf{x}_j, \mathbf{x}_k)$. The kernel function can be applied to pairs of input data to evaluate dot products in some corresponding feature space. So, we can find linear separators in the higher-dimensional feature space $F(\mathbf{x})$ simply by replacing $\mathbf{x}_j \cdot \mathbf{x}_k$ in Equation (18.13) with a kernel function $K(\mathbf{x}_j, \mathbf{x}_k)$. Thus, we can learn in the higher-dimensional space, but we compute only kernel functions rather than the full list of features for each data point.

The next step is to see that there's nothing special about the kernel $K(\mathbf{x}_j, \mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2$. It corresponds to a particular higher-dimensional feature space, but other kernel functions correspond to other feature spaces. A venerable result in mathematics, **Mercer's theorem** (1909), tells us that any "reasonable"[13] kernel function corresponds to *some* feature space. These feature spaces can be very large, even for innocuous-looking kernels. For example, the **polynomial kernel**, $K(\mathbf{x}_j, \mathbf{x}_k) = (1 + \mathbf{x}_j \cdot \mathbf{x}_k)^d$, corresponds to a feature space whose dimension is exponential in d.

MERCER'S THEOREM

POLYNOMIAL KERNEL

[12] This usage of "kernel function" is slightly different from the kernels in locally weighted regression. Some SVM kernels are distance metrics, but not all are.

[13] Here, "reasonable" means that the matrix $\mathbf{K}_{jk} = K(\mathbf{x}_j, \mathbf{x}_k)$ is positive definite.

KERNEL TRICK

This then is the clever **kernel trick**: Plugging these kernels into Equation (18.13), *optimal linear separators can be found efficiently in feature spaces with billions of (or, in some cases, infinitely many) dimensions.* The resulting linear separators, when mapped back to the original input space, can correspond to arbitrarily wiggly, nonlinear decision boundaries between the positive and negative examples.

In the case of inherently noisy data, we may not want a linear separator in some high-dimensional space. Rather, we'd like a decision surface in a lower-dimensional space that does not cleanly separate the classes, but reflects the reality of the noisy data. That is possible with the **soft margin** classifier, which allows examples to fall on the wrong side of the decision boundary, but assigns them a penalty proportional to the distance required to move them back on the correct side.

SOFT MARGIN

The kernel method can be applied not only with learning algorithms that find optimal linear separators, but also with any other algorithm that can be reformulated to work only with dot products of pairs of data points, as in Equations 18.13 and 18.14. Once this is done, the dot product is replaced by a kernel function and we have a **kernelized** version of the algorithm. This can be done easily for k-nearest-neighbors and perceptron learning (Section 18.7.2), among others.

KERNELIZATION

18.10 ENSEMBLE LEARNING

ENSEMBLE LEARNING

So far we have looked at learning methods in which a single hypothesis, chosen from a hypothesis space, is used to make predictions. The idea of **ensemble learning** methods is to select a collection, or **ensemble**, of hypotheses from the hypothesis space and combine their predictions. For example, during cross-validation we might generate twenty different decision trees, and have them vote on the best classification for a new example.

The motivation for ensemble learning is simple. Consider an ensemble of $K = 5$ hypotheses and suppose that we combine their predictions using simple majority voting. For the ensemble to misclassify a new example, *at least three of the five hypotheses have to misclassify it.* The hope is that this is much less likely than a misclassification by a single hypothesis. Suppose we assume that each hypothesis h_k in the ensemble has an error of p—that is, the probability that a randomly chosen example is misclassified by h_k is p. Furthermore, suppose we assume that the errors made by each hypothesis are *independent*. In that case, if p is small, then the probability of a large number of misclassifications occurring is minuscule. For example, a simple calculation (Exercise 18.18) shows that using an ensemble of five hypotheses reduces an error rate of 1 in 10 down to an error rate of less than 1 in 100. Now, obviously the assumption of independence is unreasonable, because hypotheses are likely to be misled in the same way by any misleading aspects of the training data. But if the hypotheses are at least a little bit different, thereby reducing the correlation between their errors, then ensemble learning can be very useful.

Another way to think about the ensemble idea is as a generic way of enlarging the hypothesis space. That is, think of the ensemble itself as a hypothesis and the new hypothesis