additional hour of daylight, 129 ± 9 more people choose to ride; a temperature increase of one degree Celsius encourages 65 ± 4 people to grab their bicycle; a dry day means an average of 546 ± 33 more riders; and each inch of precipitation means 665 ± 62 more people leave their bike at home. Once all these effects are accounted for, we see a modest increase of 28 ± 18 new daily riders each year.

Our model is almost certainly missing some relevant information. For example, nonlinear effects (such as effects of precipitation *and* cold temperature) and nonlinear trends within each variable (such as disinclination to ride at very cold and very hot temperatures) cannot be accounted for in this model. Additionally, we have thrown away some of the finer-grained information (such as the difference between a rainy morning and a rainy afternoon), and we have ignored correlations between days (such as the possible effect of a rainy Tuesday on Wednesday's numbers, or the effect of an unexpected sunny day after a streak of rainy days). These are all potentially interesting effects, and you now have the tools to begin exploring them if you wish!

In-Depth: Support Vector Machines

Support vector machines (SVMs) are a particularly powerful and flexible class of supervised algorithms for both classification and regression. In this section, we will develop the intuition behind support vector machines and their use in classification problems. We begin with the standard imports:

```
In[1]: %matplotlib inline
       import numpy as np
       import matplotlib.pyplot as plt
       from scipy import stats

       # use Seaborn plotting defaults
       import seaborn as sns; sns.set()
```

Motivating Support Vector Machines

As part of our discussion of Bayesian classification (see "In Depth: Naive Bayes Classification" on page 382), we learned a simple model describing the distribution of each underlying class, and used these generative models to probabilistically determine labels for new points. That was an example of *generative classification*; here we will consider instead *discriminative classification*: rather than modeling each class, we simply find a line or curve (in two dimensions) or manifold (in multiple dimensions) that divides the classes from each other.

As an example of this, consider the simple case of a classification task, in which the two classes of points are well separated (Figure 5-53):

```
In[2]: from sklearn.datasets.samples_generator import make_blobs
       X, y = make_blobs(n_samples=50, centers=2,
                         random_state=0, cluster_std=0.60)
       plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```

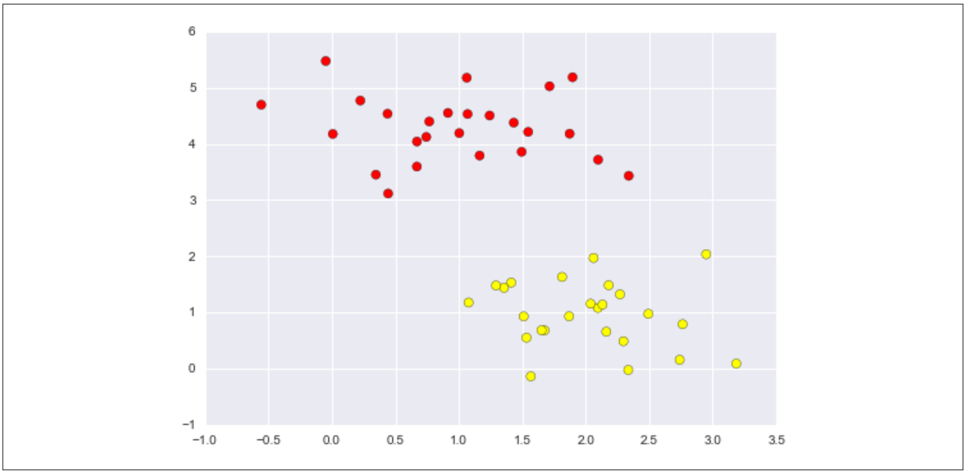

Figure 5-53. Simple data for classification

A linear discriminative classifier would attempt to draw a straight line separating the two sets of data, and thereby create a model for classification. For two-dimensional data like that shown here, this is a task we could do by hand. But immediately we see a problem: there is more than one possible dividing line that can perfectly discriminate between the two classes!

We can draw them as follows (Figure 5-54):

```
In[3]: xfit = np.linspace(-1, 3.5)
       plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
       plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

       for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
           plt.plot(xfit, m * xfit + b, '-k')

       plt.xlim(-1, 3.5);
```

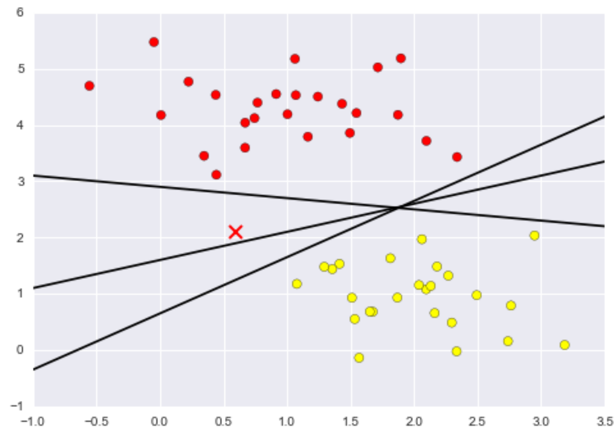

Figure 5-54. Three perfect linear discriminative classifiers for our data

These are three *very* different separators that, nevertheless, perfectly discriminate between these samples. Depending on which you choose, a new data point (e.g., the one marked by the "X" in Figure 5-54) will be assigned a different label! Evidently our simple intuition of "drawing a line between classes" is not enough, and we need to think a bit deeper.

Support Vector Machines: Maximizing the Margin

Support vector machines offer one way to improve on this. The intuition is this: rather than simply drawing a zero-width line between the classes, we can draw around each line a *margin* of some width, up to the nearest point. Here is an example of how this might look (Figure 5-55):

```
In[4]:
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none', color='#AAAAAA',
                     alpha=0.4)

plt.xlim(-1, 3.5);
```