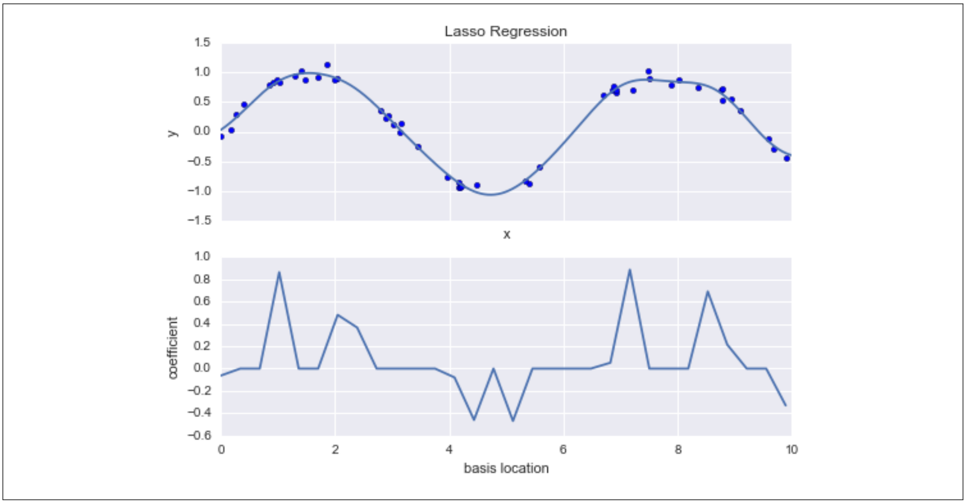

Figure 5-50. Lasso (L_1) regularization applied to the overly complex model (compare to Figure 5-48)

With the lasso regression penalty, the majority of the coefficients are exactly zero, with the functional behavior being modeled by a small subset of the available basis functions. As with ridge regularization, the α parameter tunes the strength of the penalty, and should be determined via, for example, cross-validation (refer back to "Hyperparameters and Model Validation" on page 359 for a discussion of this).

Example: Predicting Bicycle Traffic

As an example, let's take a look at whether we can predict the number of bicycle trips across Seattle's Fremont Bridge based on weather, season, and other factors. We have seen this data already in "Working with Time Series" on page 188.

In this section, we will join the bike data with another dataset, and try to determine the extent to which weather and seasonal factors—temperature, precipitation, and daylight hours—affect the volume of bicycle traffic through this corridor. Fortunately, the NOAA makes available their daily weather station data (I used station ID USW00024233) and we can easily use Pandas to join the two data sources. We will perform a simple linear regression to relate weather and other information to bicycle counts, in order to estimate how a change in any one of these parameters affects the number of riders on a given day.

In particular, this is an example of how the tools of Scikit-Learn can be used in a statistical modeling framework, in which the parameters of the model are assumed to have interpretable meaning. As discussed previously, this is not a standard approach within machine learning, but such interpretation is possible for some models.

Let's start by loading the two datasets, indexing by date:

```
In[14]:
import pandas as pd
counts = pd.read_csv('fremont_hourly.csv', index_col='Date', parse_dates=True)
weather = pd.read_csv('599021.csv', index_col='DATE', parse_dates=True)
```

Next we will compute the total daily bicycle traffic, and put this in its own `DataFrame`:

```
In[15]: daily = counts.resample('d', how='sum')
        daily['Total'] = daily.sum(axis=1)
        daily = daily[['Total']] # remove other columns
```

We saw previously that the patterns of use generally vary from day to day; let's account for this in our data by adding binary columns that indicate the day of the week:

```
In[16]: days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
        for i in range(7):
            daily[days[i]] = (daily.index.dayofweek == i).astype(float)
```

Similarly, we might expect riders to behave differently on holidays; let's add an indicator of this as well:

```
In[17]: from pandas.tseries.holiday import USFederalHolidayCalendar
        cal = USFederalHolidayCalendar()
        holidays = cal.holidays('2012', '2016')
        daily = daily.join(pd.Series(1, index=holidays, name='holiday'))
        daily['holiday'].fillna(0, inplace=True)
```

We also might suspect that the hours of daylight would affect how many people ride; let's use the standard astronomical calculation to add this information (Figure 5-51):

```
In[18]: def hours_of_daylight(date, axis=23.44, latitude=47.61):
            """Compute the hours of daylight for the given date"""
            days = (date - pd.datetime(2000, 12, 21)).days
            m = (1. - np.tan(np.radians(latitude))
                 * np.tan(np.radians(axis) * np.cos(days * 2 * np.pi / 365.25)))
            return 24. * np.degrees(np.arccos(1 - np.clip(m, 0, 2))) / 180.

        daily['daylight_hrs'] = list(map(hours_of_daylight, daily.index))
        daily[['daylight_hrs']].plot();
```

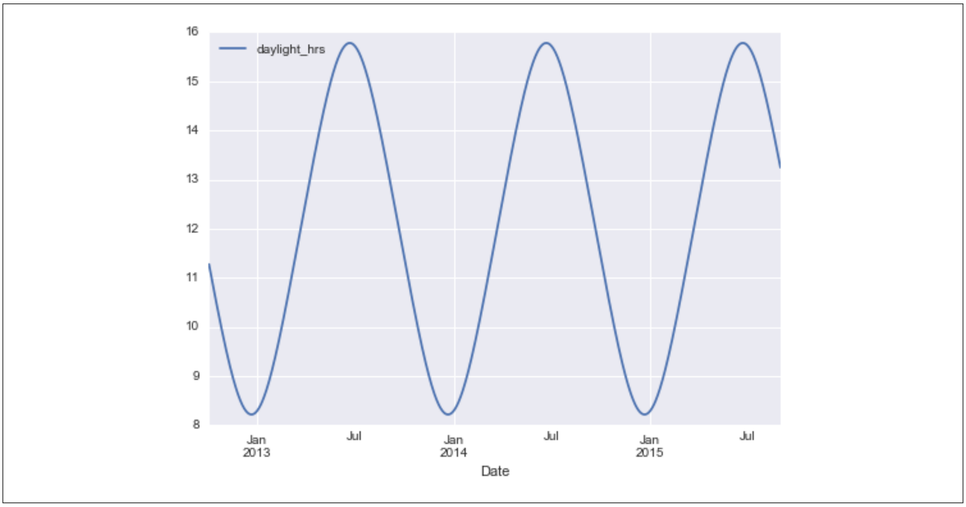

Figure 5-51. Visualization of hours of daylight in Seattle

We can also add the average temperature and total precipitation to the data. In addition to the inches of precipitation, let's add a flag that indicates whether a day is dry (has zero precipitation):

```
In[19]: # temperatures are in 1/10 deg C; convert to C
        weather['TMIN'] /= 10
        weather['TMAX'] /= 10
        weather['Temp (C)'] = 0.5 * (weather['TMIN'] + weather['TMAX'])

        # precip is in 1/10 mm; convert to inches
        weather['PRCP'] /= 254
        weather['dry day'] = (weather['PRCP'] == 0).astype(int)

        daily = daily.join(weather[['PRCP', 'Temp (C)', 'dry day']])
```

Finally, let's add a counter that increases from day 1, and measures how many years have passed. This will let us measure any observed annual increase or decrease in daily crossings:

```
In[20]: daily['annual'] = (daily.index - daily.index[0]).days / 365.
```

Now our data is in order, and we can take a look at it:

```
In[21]: daily.head()

Out[21]:
            Total  Mon  Tue  Wed  Thu  Fri  Sat  Sun  holiday  daylight_hrs  \\
Date
2012-10-03   3521    0    0    1    0    0    0    0        0     11.277359
2012-10-04   3475    0    0    0    1    0    0    0        0     11.219142
2012-10-05   3148    0    0    0    0    1    0    0        0     11.161038
2012-10-06   2006    0    0    0    0    0    1    0        0     11.103056
```

```
2012-10-07   2142    0    0    0    0    0    0    1        0     11.045208

            PRCP  Temp (C)  dry day    annual
Date
2012-10-03     0     13.35        1  0.000000
2012-10-04     0     13.60        1  0.002740
2012-10-05     0     15.30        1  0.005479
2012-10-06     0     15.85        1  0.008219
2012-10-07     0     15.85        1  0.010959
```

With this in place, we can choose the columns to use, and fit a linear regression model to our data. We will set `fit_intercept = False`, because the daily flags essentially operate as their own day-specific intercepts:

```
In[22]:
column_names = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'holiday',
                'daylight_hrs', 'PRCP', 'dry day', 'Temp (C)', 'annual']
X = daily[column_names]
y = daily['Total']

model = LinearRegression(fit_intercept=False)
model.fit(X, y)
daily['predicted'] = model.predict(X)
```

Finally, we can compare the total and predicted bicycle traffic visually (Figure 5-52):

```
In[23]: daily[['Total', 'predicted']].plot(alpha=0.5);
```

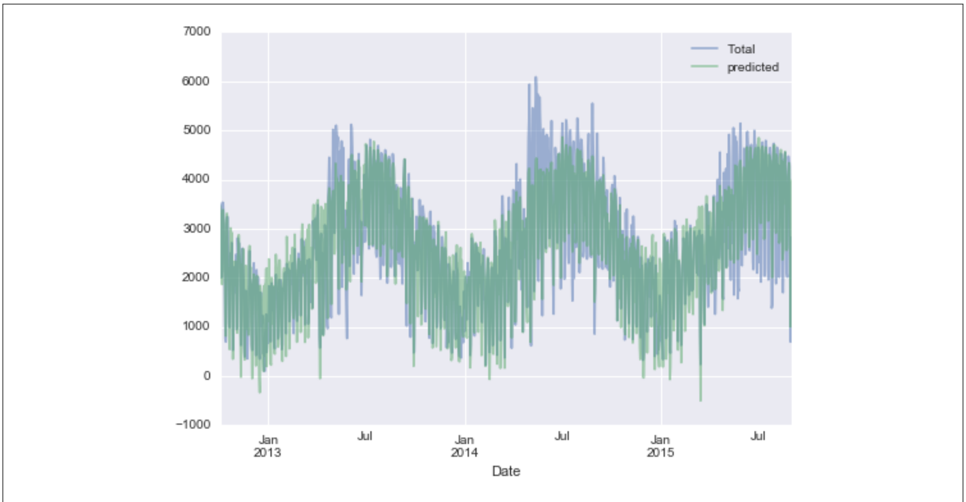

Figure 5-52. Our model's prediction of bicycle traffic

It is evident that we have missed some key features, especially during the summer time. Either our features are not complete (i.e., people decide whether to ride to work based on more than just these) or there are some nonlinear relationships that we have failed to take into account (e.g., perhaps people ride less at both high and low temperatures). Nevertheless, our rough approximation is enough to give us some insights, and we can take a look at the coefficients of the linear model to estimate how much each feature contributes to the daily bicycle count:

```
In[24]: params = pd.Series(model.coef_, index=X.columns)
        params
```

```
Out[24]: Mon              503.797330
         Tue              612.088879
         Wed              591.611292
         Thu              481.250377
         Fri              176.838999
         Sat            -1104.321406
         Sun            -1134.610322
         holiday        -1187.212688
         daylight_hrs     128.873251
         PRCP            -665.185105
         dry day          546.185613
         Temp (C)          65.194390
         annual            27.865349
         dtype: float64
```

These numbers are difficult to interpret without some measure of their uncertainty. We can compute these uncertainties quickly using bootstrap resamplings of the data:

```
In[25]: from sklearn.utils import resample
        np.random.seed(1)
        err = np.std([model.fit(*resample(X, y)).coef_
                      for i in range(1000)], 0)
```

With these errors estimated, let's again look at the results:

```
In[26]: print(pd.DataFrame({'effect': params.round(0),
                            'error': err.round(0)}))
```

```
              effect  error
Mon              504     85
Tue              612     82
Wed              592     82
Thu              481     85
Fri              177     81
Sat            -1104     79
Sun            -1135     82
holiday        -1187    164
daylight_hrs     129      9
PRCP            -665     62
dry day          546     33
Temp (C)          65      4
annual            28     18
```

We first see that there is a relatively stable trend in the weekly baseline: there are many more riders on weekdays than on weekends and holidays. We see that for each 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):