```
In[13]: # project the digits into 2 dimensions using IsoMap
        from sklearn.manifold import Isomap
        iso = Isomap(n_components=2)
        projection = iso.fit_transform(digits.data)
```

We'll use our discrete colormap to view the results, setting the `ticks` and `clim` to improve the aesthetics of the resulting colorbar (Figure 4-58):

```
In[14]: # plot the results
        plt.scatter(projection[:, 0], projection[:, 1], lw=0.1,
                    c=digits.target, cmap=plt.cm.get_cmap('cubehelix', 6))
        plt.colorbar(ticks=range(6), label='digit value')
        plt.clim(-0.5, 5.5)
```

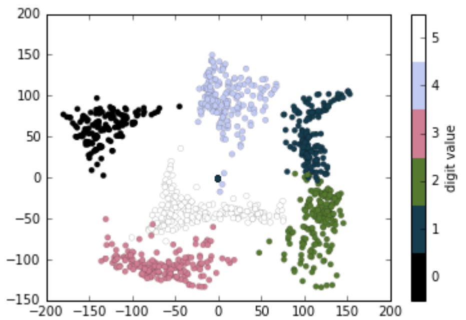

Figure 4-58. Manifold embedding of handwritten digit pixels

The projection also gives us some interesting insights on the relationships within the dataset: for example, the ranges of 5 and 3 nearly overlap in this projection, indicating that some handwritten fives and threes are difficult to distinguish, and therefore more likely to be confused by an automated classification algorithm. Other values, like 0 and 1, are more distantly separated, and therefore much less likely to be confused. This observation agrees with our intuition, because 5 and 3 look much more similar than do 0 and 1.

We'll return to manifold learning and digit classification in Chapter 5.

Multiple Subplots

Sometimes it is helpful to compare different views of data side by side. To this end, Matplotlib has the concept of *subplots*: groups of smaller axes that can exist together within a single figure. These subplots might be insets, grids of plots, or other more complicated layouts. In this section, we'll explore four routines for creating subplots in Matplotlib. We'll start by setting up the notebook for plotting and importing the functions we will use:

```
In[1]: %matplotlib inline
       import matplotlib.pyplot as plt
       plt.style.use('seaborn-white')
       import numpy as np
```

plt.axes: Subplots by Hand

The most basic method of creating an axes is to use the `plt.axes` function. As we've seen previously, by default this creates a standard axes object that fills the entire figure. `plt.axes` also takes an optional argument that is a list of four numbers in the figure coordinate system. These numbers represent [*bottom*, *left*, *width*, *height*] in the figure coordinate system, which ranges from 0 at the bottom left of the figure to 1 at the top right of the figure.

For example, we might create an inset axes at the top-right corner of another axes by setting the *x* and *y* position to 0.65 (that is, starting at 65% of the width and 65% of the height of the figure) and the *x* and *y* extents to 0.2 (that is, the size of the axes is 20% of the width and 20% of the height of the figure). Figure 4-59 shows the result of this code:

```
In[2]: ax1 = plt.axes()  # standard axes
       ax2 = plt.axes([0.65, 0.65, 0.2, 0.2])
```

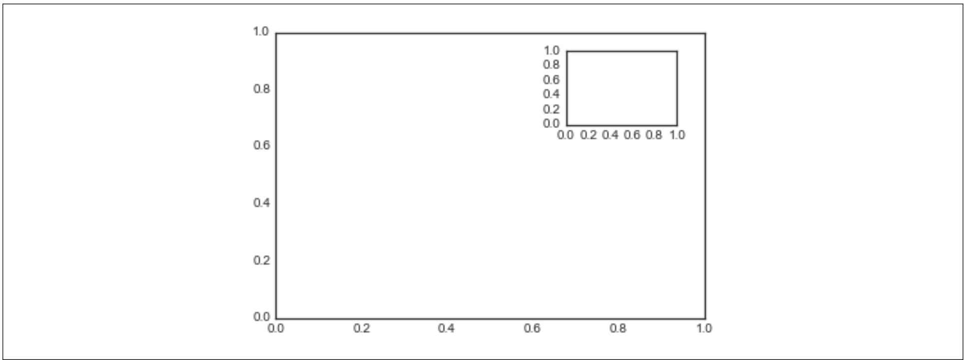

Figure 4-59. Example of an inset axes

The equivalent of this command within the object-oriented interface is `fig.add_axes()`. Let's use this to create two vertically stacked axes (Figure 4-60):

```
In[3]: fig = plt.figure()
       ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4],
                          xticklabels=[], ylim=(-1.2, 1.2))
       ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4],
                          ylim=(-1.2, 1.2))

       x = np.linspace(0, 10)
       ax1.plot(np.sin(x))
       ax2.plot(np.cos(x));
```