Choosing the number of components

A vital part of using PCA in practice is the ability to estimate how many components are needed to describe the data. We can determine this by looking at the cumulative *explained variance ratio* as a function of the number of components (Figure 5-87):

```
In[12]: pca = PCA().fit(digits.data)
        plt.plot(np.cumsum(pca.explained_variance_ratio_))
        plt.xlabel('number of components')
        plt.ylabel('cumulative explained variance');
```

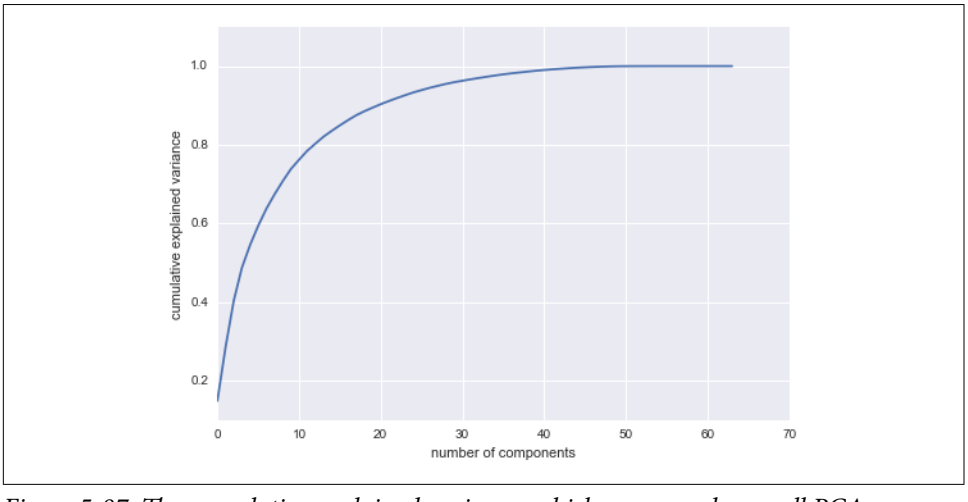

Figure 5-87. The cumulative explained variance, which measures how well PCA preserves the content of the data

This curve quantifies how much of the total, 64-dimensional variance is contained within the first *N* components. For example, we see that with the digits the first 10 components contain approximately 75% of the variance, while you need around 50 components to describe close to 100% of the variance.

Here we see that our two-dimensional projection loses a lot of information (as measured by the explained variance) and that we'd need about 20 components to retain 90% of the variance. Looking at this plot for a high-dimensional dataset can help you understand the level of redundancy present in multiple observations.

PCA as Noise Filtering

PCA can also be used as a filtering approach for noisy data. The idea is this: any components with variance much larger than the effect of the noise should be relatively unaffected by the noise. So if you reconstruct the data using just the largest subset of principal components, you should be preferentially keeping the signal and throwing out the noise.

Let's see how this looks with the digits data. First we will plot several of the input noise-free data (Figure 5-88):

```
In[13]: def plot_digits(data):
            fig, axes = plt.subplots(4, 10, figsize=(10, 4),
                                     subplot_kw={'xticks':[], 'yticks':[]},
                                     gridspec_kw=dict(hspace=0.1, wspace=0.1))
            for i, ax in enumerate(axes.flat):
                ax.imshow(data[i].reshape(8, 8),
                          cmap='binary', interpolation='nearest',
                          clim=(0, 16))
        plot_digits(digits.data)
```

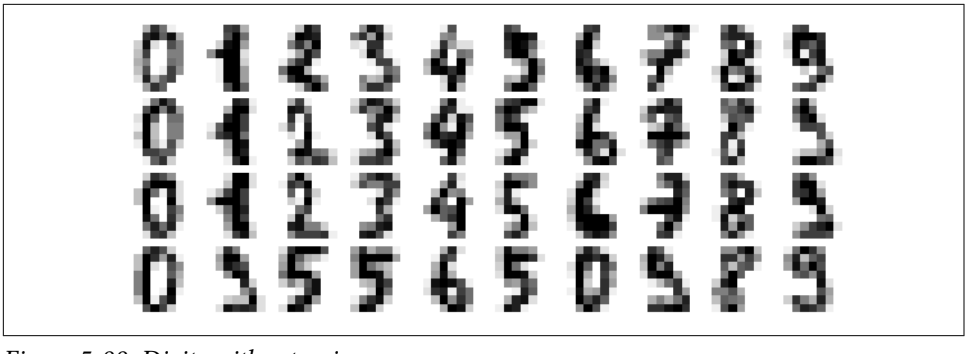

Figure 5-88. Digits without noise

Now let's add some random noise to create a noisy dataset, and replot it (Figure 5-89):

```
In[14]: np.random.seed(42)
        noisy = np.random.normal(digits.data, 4)
        plot_digits(noisy)
```

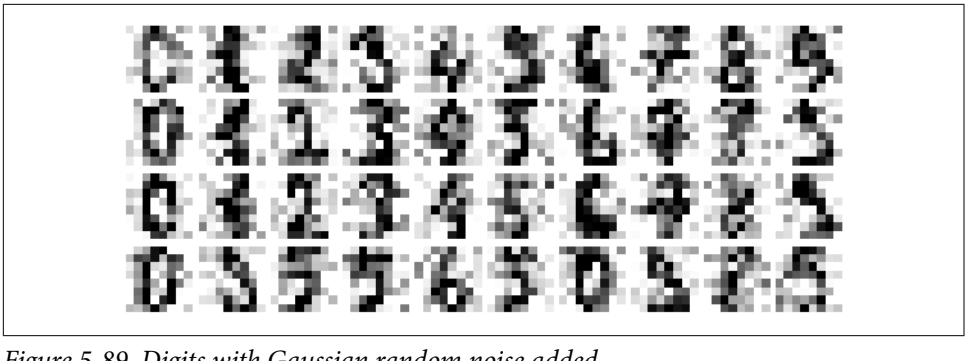

Figure 5-89. Digits with Gaussian random noise added

It's clear by eye that the images are noisy, and contain spurious pixels. Let's train a PCA on the noisy data, requesting that the projection preserve 50% of the variance:

```
In[15]: pca = PCA(0.50).fit(noisy)
        pca.n_components_
```

```
Out[15]: 12
```

Here 50% of the variance amounts to 12 principal components. Now we compute these components, and then use the inverse of the transform to reconstruct the filtered digits (Figure 5-90):

```
In[16]: components = pca.transform(noisy)
        filtered = pca.inverse_transform(components)
        plot_digits(filtered)
```

Figure 5-90. Digits "denoised" using PCA

This signal preserving/noise filtering property makes PCA a very useful feature selection routine—for example, rather than training a classifier on very high-dimensional data, you might instead train the classifier on the lower-dimensional representation, which will automatically serve to filter out random noise in the inputs.

Example: Eigenfaces

Earlier we explored an example of using a PCA projection as a feature selector for facial recognition with a support vector machine ("In-Depth: Support Vector Machines" on page 405). Here we will take a look back and explore a bit more of what went into that. Recall that we were using the Labeled Faces in the Wild dataset made available through Scikit-Learn:

```
In[17]: from sklearn.datasets import fetch_lfw_people
        faces = fetch_lfw_people(min_faces_per_person=60)
        print(faces.target_names)
        print(faces.images.shape)
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

Let's take a look at the principal axes that span this dataset. Because this is a large dataset, we will use `RandomizedPCA`—it contains a randomized method to approxi-