Reducing dimensionality with PCA

Dimensionality Reduction

Dimensionality reduction is motivated by several problems. First, it can be used to mitigate problems caused by the curse of dimensionality. Second, dimensionality reduction can be used to compress data while minimizing the amount of information that is lost. Third, understanding the structure of data with hundreds of dimensions can be difficult; data with only two or three dimensions can be visualized easily.

Principal Component Analysis

> The main purposes of a principal component analysis are the analysis of data to identify patterns and finding patterns to reduce the dimensions of the dataset with minimal loss of information.

Principal components analysis (PCA) is a very popular technique for dimensionality reduction. Given a set of data on n dimensions, PCA aims to find a linear subspace of dimension d lower than n such that the data points lie mainly on this linear subspace. Such a reduced subspace attempts to maintain most of the variability of the data.

> PCA reduces a set of possibly-correlated, high-dimensional variables to a lower-dimensional set of linearly uncorrelated synthetic variables called principal components. The lower-dimensional data will preserve as much of the variance of the original data as possible.

- Principal component analysis (PCA) is the first somewhat advanced technique discussed in this book.

- While everything else thus far has been simple statistics, PCA will combine statistics and linear algebra to produce a preprocessing step that can help to reduce dimensionality, which can be the enemy of a simple model.

Getting ready

PCA is a member of the decomposition module of scikit-learn. There are several other decomposition methods available, which will be covered later. Let's use the iris dataset, but it's better if you use your own data:

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> iris_X = iris.data
```

First, import the decomposition module. Then instantiate a default PCA object:

```
>>> from sklearn import decomposition
>>> pca = decomposition.PCA()
>>> pca
PCA(copy=True, n_components=None, whiten=False)
```

- Compared to other objects in scikit-learn, PCA takes relatively few arguments.
- Now that the PCA object is created, simply transform the data by calling the `fit_transform` method, with `iris_X` as the argument:

```
>>> iris_pca = pca.fit_transform(iris_X)
>>> iris_pca[:5]
```

- Now that the PCA has been fit, we can see how well it has done at explaining the variance (explained in the following How it works... section):

```
>>> pca.explained_variance_ratio_
array([ 0.925, 0.053, 0.017, 0.005])
```

0.1 Theoretical Basis

- PCA has a general mathematic definition and a specific use case in data analysis. PCA finds the set of orthogonal directions that represent the original data matrix.
- Generally, PCA works by mapping the original dataset into a new space where the new column vectors of the matrix are each orthogonal. From a data analysis perspective, PCA transforms the covariance matrix of the data into column vectors that can "explain" certain percentages of the variance.
- For example, with the iris dataset, 92.5 percent of the variance of the overall dataset can be explained by the first component.
- This is extremely useful because dimensionality is problematic in data analysis.
- Quite often, algorithms applied to high-dimensional datasets will overfit on the initial training, and thus loose generality to the test set.
- If most of the underlying structure of the data can be faithfully represented by fewer dimensions, then it's generally considered a worthwhile trade-off.

- To demonstrate this, we'll apply the PCA transformation to the iris dataset and only include two dimensions.

The iris example

The iris dataset can normally be separated quite well using all the dimensions:

```
>>> pca = decomposition.PCA(n_components=2)
>>> iris_X_prime = pca.fit_transform(iris_X)
>>> iris_X_prime.shape
(150, 2)
```

Our data matrix is now 150 x 2, instead of 150 x 4. The usefulness of two dimensions is that it is now very easy to plot. The separability of the classes remain even after reducing the number of dimensionality by two. We can see how much of the variance is represented by the two components that remain:

```
>>> pca.explained_variance_ratio_.sum()
0.9776
```

The PCA object can also be created with the amount of explained variance in mind from the start. For example, if we want to be able to explain at least 98 percent of the variance, the PCA object will be created as follows:

```
>>> pca = decomposition.PCA(n_components=.98)
>>> iris_X_prime = pca.fit(iris_X)
```

```
>>> pca.explained_variance_ratio_.sum()
1.0
```

Since we wanted to explain variance slightly more than the two component examples, a third was included.

1 Using factor analysis for decomposition

- Factor analysis is another technique we can use to reduce dimensionality. However, factor analysis makes assumptions and PCA does not.
- The basic assumption is that there are implicit features responsible for the features of the dataset.
- This recipe will boil down to the explicit features from our samples in an attempt to understand the independent variables as much as the dependent variables.

1.0.1 Getting ready

To compare PCA and factor analysis, let's use the iris dataset again, but we'll first need to load the factor analysis class:

```
>>> from sklearn.decomposition import FactorAnalysis
```

How to do it... From a programming perspective, factor analysis isn't much different from PCA:

```
>>> fa = FactorAnalysis(n_components=2)
>>> iris_two_dim = fa.fit_transform(iris.data)
>>> iris_two_dim[:5]
array([[-1.33125848, 0.55846779],
[-1.33914102, -0.00509715],
[-1.40258715, -0.307983 ],
[-1.29839497, -0.71854288],
[-1.33587575, 0.36533259]])
Downloading the example code
```

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you Compare the following plot to the plot in the last section: Since factor analysis is a probabilistic transform, we can examine different aspects such as the log likelihood of the observations under the model, and better still, compare the log likelihoods across models. Factor analysis is not without flaws. The reason is that you're not fitting a model to predict an outcome, you're fitting a model as a preparation step. This isn't a bad thing per se, but errors here compound when training the actual model.

1.1 Comparing Factor Analysis and PCA

- Factor analysis is similar to PCA, which was covered previously. However, there is an important distinction to be made.
- PCA is a linear transformation of the data to a different space where the first component "explains" the variance of the data, and each subsequent component is orthogonal to the first component.
- For example, you can think of PCA as taking a dataset of N dimensions and going down to some space of M dimensions, where $M < N$.
- Factor analysis, on the other hand, works under the assumption that there are only M important features and a linear combination of these features (plus noise) creates the dataset in N dimensions.
- To put it another way, you don't do regression on an outcome variable, you do regression on the features to determine the latent factors of the dataset.