Math221_Visualisation_PCA

January 25, 2018

Math 221 - Applied Statistics (Data Analysis)
Dr. Kamal Dingle (Spring 2018)
VISUALISING DATA
Principal Component Analysis (PCA)

Many data sets have several **variables** or **dimensions**, known as **multivariate data**. PCA is a method for visualising multivariate data (and dimensionality reduction).

In this course, we will want to visualise data in only 2 dimensions, so that we can view the data on an X-Y scatter plot. Hence, we will use PCA as a technique to help us view multivariate data on an X-Y scatter plot.

We start by importing numpy library and matplolib, as usual.

```
In [7]: import numpy as np
        import matplotlib.pyplot as plt
```

Let's make some random data with 3 variables/dimensions (d=3), and 100 samples (n=100).

```
In [8]: X_data = np.random.randn(100,3)# 100 rows (samples) and 3 columns (dimensions)

        print ('Look at the first 5 rows of the data\n',X_data[:5])
```

```
Look at the first 5 rows of the data
 [[-1.66290406 -0.85871536 -0.48109989]
 [ 1.39645708 -1.23504103 -0.51882934]
 [ 2.3958706  -0.9651854   0.14539949]
 [ 0.60563611 -0.67746544  1.17926007]
 [-0.16433801 -2.05583074 -0.56233758]]
```

Just for practice, let's save this data, and then load it in again

```
In [9]: np.savetxt('MyData.txt',X_data)
```

And then let's load in the data again, just for practice

```
In [10]: X = np.loadtxt('MyData.txt')
         print ('Look at the first 5 rows of the data\n',X[:5])# should be the same as above!
```

```
Look at the first 5 rows of the data
 [[-1.66290406 -0.85871536 -0.48109989]
 [ 1.39645708 -1.23504103 -0.51882934]
 [ 2.3958706  -0.9651854   0.14539949]
 [ 0.60563611 -0.67746544  1.17926007]
 [-0.16433801 -2.05583074 -0.56233758]]
```

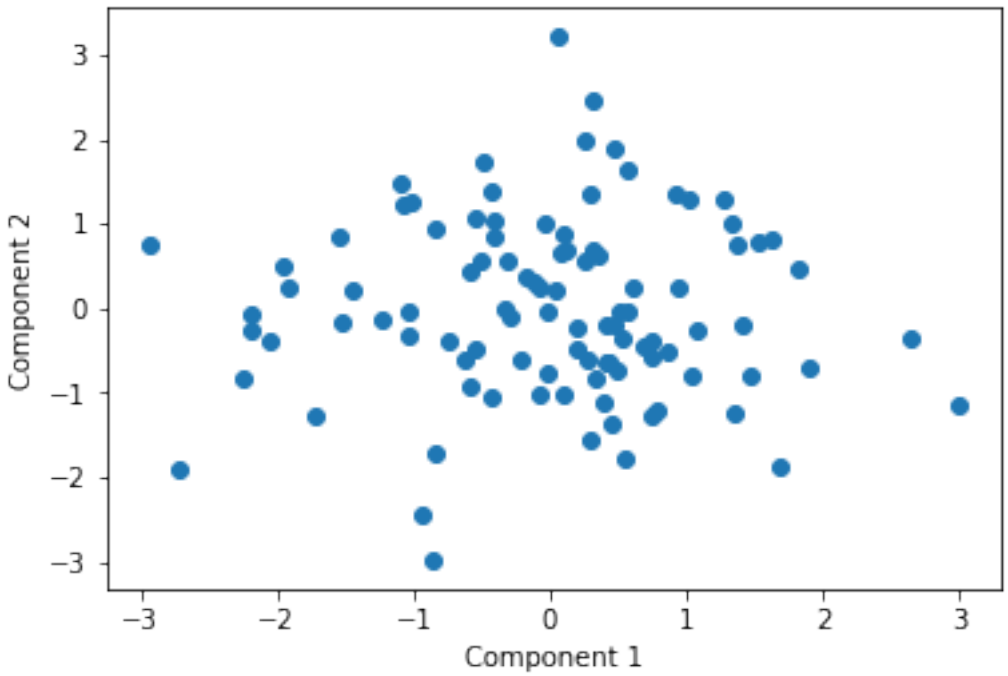
Performing PCA by hand would be very difficult - but in Python it is easy. We will use a popular machine learning library called **Scikit-learn** (http://scikit-learn.org/stable/).

```
In [11]: from sklearn.decomposition import PCA # import the PCA tool

         pca = PCA(n_components=2)# we want 2 components, because we want to draw a 2D scatter

         X_pca = pca.fit_transform(X)# perform the projection

         print ('Look at the first 5 rows of X_pca\n',X_pca[:5])
```

```
Look at the first 5 rows of X_pca
 [[ 0.44949596 -1.37164488]
 [ 1.52023762  0.77852945]
 [ 1.26644347  1.28846132]
 [ 0.20447772 -0.47112618]
 [ 1.90740413 -0.70157051]]
```

Notice there are only 2 columns - the 3 dimensional data has been projected onto just 2 variables/axes - now we can make a scatter plot.

```
In [12]: plt.figure()
         plt.scatter(X_pca[:,0],X_pca[:,1])
         plt.xlabel('Component 1')
         plt.ylabel('Component 2')
         plt.show()
```



As discussed in class, the PCA method will make the projection (or "shadow") in a way which best matches or fits the data. Naturally, by doing a projection, we lose some of the detail about the original data samples. This is similar to how a 2D shadow on a wall loses some of the detail of the 3D object which caused the shadow.

However, while some detail is always lost by projections, the PCA technique tries to minimise the amount of detail that is lost.

More precisely, the projection is made such that the total (squared) distance between the plane and the data samples is minimized. Because the projection seeks to minimize these distances, any **outliers** in the data should first be removed, to reduce their influence on the projection.

Optional: For a more detailed look at PCA, you can look at this by Jacob VanderPlas (one of the creators of scikit-learn): https://github.com/jakevdp/sklearn_tutorial/blob/master/notebooks/04.1-Dimensionality-PCA.ipynb