Lab 18 - PCA in Python

April 25, 2016

This lab on Principal Components Analysis is a python adaptation of p. 401-404, 408-410 of "Introduction to Statistical Learning with Applications in R" by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. Original adaptation by J. Warmenhoven, updated by R. Jordan Crouser at Smith College for SDS293: Machine Learning (Spring 2016).

```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib as mpl
        import matplotlib.pyplot as plt

        %matplotlib inline
```

1 10.4: Principal Components Analysis

In this lab, we perform PCA on the `USArrests` data set. The rows of the data set contain the 50 states, in alphabetical order:

```
In [ ]: df = pd.read_csv('USArrests.csv', index_col=0)
        df.head()
```

The columns of the data set contain four variables relating to various crimes:

```
In [ ]: df.info()
```

Let's start by taking a quick look at the column means of the data:

```
In [ ]: df.mean()
```

We see right away the the data have **vastly** different means. We can also examine the variances of the four variables:

```
In [ ]: df.var()
```

Not surprisingly, the variables also have vastly different variances: the `UrbanPop` variable measures the percentage of the population in each state living in an urban area, which is not a comparable number to the number of crimes committeed in each state per 100,000 individuals. If we failed to scale the variables before performing PCA, then most of the principal components that we observed would be driven by the `Assault` variable, since it has by far the largest mean and variance.

Thus, it is important to standardize the variables to have mean zero and standard deviation 1 before performing PCA. We can do this using the `scale`() function from `sklearn`:

```
In [ ]: from sklearn.preprocessing import scale
        X = pd.DataFrame(scale(df), index=df.index, columns=df.columns)
```

Now we'll use the `PCA`() function from `sklearn` to compute the loading vectors:

```
In [ ]: from sklearn.decomposition import PCA

        pca_loadings = pd.DataFrame(PCA().fit(X).components_.T, index=df.columns, columns=['V1', 'V2',
        pca_loadings
```

We see that there are four distinct principal components. This is to be expected because there are in general $\texttt{min}(\texttt{n}-1,\texttt{p})$ informative principal components in a data set with n observations and p variables.

Using the $\texttt{fit}_\texttt{t}\texttt{ransform}()$ function, we can get the principal component scores of the original data. We'll take a look at the first few states:

```
In [ ]: # Fit the PCA model and transform X to get the principal components
        pca = PCA()
        df_plot = pd.DataFrame(pca.fit_transform(X), columns=['PC1', 'PC2', 'PC3', 'PC4'], index=X.inde
        df_plot.head()
```

We can construct a **biplot** of the first two principal components using our loading vectors:

```
In [ ]: fig , ax1 = plt.subplots(figsize=(9,7))

        ax1.set_xlim(-3.5,3.5)
        ax1.set_ylim(-3.5,3.5)

        # Plot Principal Components 1 and 2
        for i in df_plot.index:
            ax1.annotate(i, (-df_plot.PC1.loc[i], -df_plot.PC2.loc[i]), ha='center')

        # Plot reference lines
        ax1.hlines(0,-3.5,3.5, linestyles='dotted', colors='grey')
        ax1.vlines(0,-3.5,3.5, linestyles='dotted', colors='grey')

        ax1.set_xlabel('First Principal Component')
        ax1.set_ylabel('Second Principal Component')

        # Plot Principal Component loading vectors, using a second y-axis.
        ax2 = ax1.twinx().twiny()

        ax2.set_ylim(-1,1)
        ax2.set_xlim(-1,1)
        ax2.set_xlabel('Principal Component loading vectors', color='red')

        # Plot labels for vectors. Variable 'a' is a small offset parameter to separate arrow tip and t
        a = 1.07
        for i in pca_loadings[['V1', 'V2']].index:
            ax2.annotate(i, (-pca_loadings.V1.loc[i]*a, -pca_loadings.V2.loc[i]*a), color='red')

        # Plot vectors
        ax2.arrow(0,0,-pca_loadings.V1[0], -pca_loadings.V2[0])
        ax2.arrow(0,0,-pca_loadings.V1[1], -pca_loadings.V2[1])
        ax2.arrow(0,0,-pca_loadings.V1[2], -pca_loadings.V2[2])
        ax2.arrow(0,0,-pca_loadings.V1[3], -pca_loadings.V2[3])
```

The PCA() function also outputs the variance explained by of each principal component. We can access these values as follows:

```
In [ ]: pca.explained_variance_
```

We can also get the proportion of variance explained:

```
In [ ]: pca.explained_variance_ratio_
```

We see that the first principal component explains 62.0% of the variance in the data, the next principal component explains 24.7% of the variance, and so forth. We can plot the PVE explained by each component as follows:

```
In [ ]: plt.figure(figsize=(7,5))
        plt.plot([1,2,3,4], pca.explained_variance_ratio_, '-o')
        plt.ylabel('Proportion of Variance Explained')
        plt.xlabel('Principal Component')
        plt.xlim(0.75,4.25)
        plt.ylim(0,1.05)
        plt.xticks([1,2,3,4])
```

We can also use the function `cumsum()`, which computes the cumulative sum of the elements of a numeric vector, to plot the cumulative PVE:

```
In [ ]: plt.figure(figsize=(7,5))
        plt.plot([1,2,3,4], np.cumsum(pca.explained_variance_ratio_), '-s')
        plt.ylabel('Proportion of Variance Explained')
        plt.xlabel('Principal Component')
        plt.xlim(0.75,4.25)
        plt.ylim(0,1.05)
        plt.xticks([1,2,3,4])
```

2 10.6: NCI60 Data Example

Let's return to the `NCI60` cancer cell line microarray data, which consists of 6,830 gene expression measurements on 64 cancer cell lines:

```
In [ ]: df2 = pd.read_csv('NCI60.csv').drop('Unnamed: 0', axis=1)
        df2.columns = np.arange(df2.columns.size)
        df2.info()
```

```
In [ ]: # Read in the labels to check our work later
        y = pd.read_csv('NCI60_y.csv', usecols=[1], skiprows=1, names=['type'])
```

3 10.6.1 PCA on the NCI60 Data

We first perform PCA on the data after scaling the variables (genes) to have standard deviation one, although one could reasonably argue that it is better not to scale the genes:

```
In [ ]: # Scale the data
        X = pd.DataFrame(scale(df2))
        X.shape

        # Fit the PCA model and transform X to get the principal components
        pca2 = PCA()
        df2_plot = pd.DataFrame(pca2.fit_transform(X))
```

We now plot the first few principal component score vectors, in order to visualize the data. The observations (cell lines) corresponding to a given cancer type will be plotted in the same color, so that we can see to what extent the observations within a cancer type are similar to each other:

```
In [ ]: fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,6))

        color_idx = pd.factorize(y.type)[0]
        cmap = mpl.cm.hsv

        # Left plot
        ax1.scatter(df2_plot.iloc[:,0], df2_plot.iloc[:,1], c=color_idx, cmap=cmap, alpha=0.5, s=50)
        ax1.set_ylabel('Principal Component 2')

        # Right plot
        ax2.scatter(df2_plot.iloc[:,0], df2_plot.iloc[:,2], c=color_idx, cmap=cmap, alpha=0.5, s=50)
        ax2.set_ylabel('Principal Component 3')

        # Custom legend for the classes (y) since we do not create scatter plots per class (which could
        handles = []
        labels = pd.factorize(y.type.unique())
        norm = mpl.colors.Normalize(vmin=0.0, vmax=14.0)

        for i, v in zip(labels[0], labels[1]):
            handles.append(mpl.patches.Patch(color=cmap(norm(i)), label=v, alpha=0.5))

        ax2.legend(handles=handles, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

        # xlabel for both plots
        for ax in fig.axes:
            ax.set_xlabel('Principal Component 1')
```

On the whole, cell lines corresponding to a single cancer type do tend to have similar values on the first few principal component score vectors. This indicates that cell lines from the same cancer type tend to have pretty similar gene expression levels.

We can generate a summary of the proportion of variance explained (PVE) of the first few principal components:

```
In [ ]: pd.DataFrame([df2_plot.iloc[:,:5].std(axis=0, ddof=0).as_matrix(),
                      pca2.explained_variance_ratio_[:5],
                      np.cumsum(pca2.explained_variance_ratio_[:5])],
                     index=['Standard Deviation', 'Proportion of Variance', 'Cumulative Proportion'],
                     columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5'])
```

Using the `plot()` function, we can also plot the variance explained by the first few principal components:

```
In [ ]: df2_plot.iloc[:,:10].var(axis=0, ddof=0).plot(kind='bar', rot=0)
        plt.ylabel('Variances')
```

However, it is generally more informative to plot the PVE of each principal component (i.e. a **scree plot**) and the cumulative PVE of each principal component. This can be done with just a little tweaking:

```
In [ ]: fig , (ax1,ax2) = plt.subplots(1,2, figsize=(15,5))

        # Left plot
        ax1.plot(pca2.explained_variance_ratio_, '-o')
        ax1.set_ylabel('Proportion of Variance Explained')
        ax1.set_ylim(ymin=-0.01)

        # Right plot
```

```
ax2.plot(np.cumsum(pca2.explained_variance_ratio_), '-ro')
ax2.set_ylabel('Cumulative Proportion of Variance Explained')
ax2.set_ylim(ymax=1.05)

for ax in fig.axes:
    ax.set_xlabel('Principal Component')
    ax.set_xlim(-1,65)
```

We see that together, the first seven principal components explain around 40% of the variance in the data. This is not a huge amount of the variance. However, looking at the scree plot, we see that while each of the first seven principal components explain a substantial amount of variance, there is a marked decrease in the variance explained by further principal components. That is, there is an **elbow** in the plot after approximately the seventh principal component. This suggests that there may be little benefit to examining more than seven or so principal components (phew! even examining seven principal components may be difficult).