ISEP
IF.1105 - IF.2301 Data Science
January, 2024

Principal Component Analysis (PCA)

1 Part I : Exercise

Exercise 1. Objective : Perform PCA step by step with a toy example.

Consider the following dataset with two variables, X_1 and X_2, and 4 observations :

Obs	X_1	X_2
1	1	2
2	5	2
3	6	5
4	9	6

1. Standardize the variables in the dataset.
2. Calculate the variance matrix of the standardized variables. Then, diagonalize the variance matrix to find its eigenvectors and eigenvalues. Deduce the principal component loading vectors.

3. Calculate the principal component scores by multiplying X by each principal component loading vector. What are the new coordinates of observation 4 ? What about observation 2 ? Interpret these results.

4. Using the eigenvalues, calculate the amount of variance explained by each component. What is the percentage of explained variance (PVE) by each component ?

   You can compare the obtained results to those obtained using the `prcomp()` by executing the following code (where X is the matrix of the original dataset) :

```
X.pca=prcomp(X,scale=T)
biplot(X.pca, scale=0)
```

2 Part II : PCA with real data

In this part you will perform Principal Component Analysis on a real dataset. In moodle you will find a csv file. Open this file to get familiar with. This file contains descriptive statistics about arrests per 100,000 residents for assault, murder, and rape committed in 1973 in each state of the US. There is also the percent of the population living in urban areas.

— In `R`, import the data using the command `read.table()` with the right options (`header`, `sep`, etc.).

— In Python, import the data using the command `read_csv()` from the pandas library with the right options (`index_col`, `sep`, etc.).

You will call the dataset `USArrests`.

1. Describe the dataset. How many variables are there ? How many observations are there ?

2. Calculate the empirical mean of each variable. Interpret the results.
   — In R, The `apply()` function allows to apply a function to each row or column of the data set. Execute the command `apply(USArrests , 2, mean)`.
   — In Python, you can use the `describe()` function to calculate descriptive statistics for all the variables in a dataframe.

3. Examine the variances of the four variables. Do you think it is necessary to standardize the variables before performing *PCA* for this dataset ? Why ?
   You can use the functions used in the previous question.
4. Perform PCA (Principal Component Analysis) using the 4 variables in the dataset.
   — In R execute the command : `pr.out=prcomp(USArrests, scale=TRUE)`. The option `scale` allows to previously standardize the variables. The result of `pr.out$rotation` is called the *rotation matrix*, the column vectors of this matrix are the principal components. For instance, the command `pr.out$rotation[,1]` returns the first principal component loading vector.
   — In Python you will need to manually standardize the variables before performing PCA. For that you will need the `sklearn.preprocessing` package :

```
from sklearn import preprocessing
USArrests_std = preprocessing.StandardScaler().fit_transform(USArrests)
```

   Warning ! The scaled values obtained using the `prcomp` from `R` and the `StandardScaler()` from scikit-learn package in Python differ. Indeed, the R function standardizes based on the unbiased variance whereas scikit-learn uses the biased variance. Fortunately, this does not affect the PCA analysis because the loading vectors are the same as well as the percentage of variance explained by each component. However, the new principal component scores differ (since they result from multiplying the scaled values by the the loading vectors). Fortunately, this does not have any impact in the interpretation of the results.

   Finally, in order to perform PCA in Python you can use the `PCA` function from the `sklearn.decomposition` module as follows :

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(USArrests_std)
```

   (a) What is the percentage of variance explained *(PVE)* by each component ? Plot the screeplot (*PVE* explained by each component), as well as the cumulative *PVE*. How many components would you keep ? Why ?

   — In `R` the standard deviation explained by each principal component can be found using the command `pr.out$sdev`. Then, for the plots you can run the commands :

```
plot(pve, xlab="Principal Component",
ylab="Proportion of Variance Explained ",
 ylim=c(0,1),type="b")
plot(cumsum(pve), xlab="Principal Component ",
ylab=" Cumulative Proportion of Variance Explained ", ylim=c(0,1), type="b")\\
```

— In Python, the command `pca.explained_variance_ratio_` returns the percentage of variance explained by each component. In order to obtain the scree-plot you can execute the code :

```
import matplotlib.pyplot as plt
per_var= np.round(pca.explained_variance_ratio_*100,decimals=1)
print(per_var)
print(np.cumsum(per_var))
labels =["PC1","PC2","PC3","PC4"]
plt.plot([1,2,3,4],per_var)
plt.ylabel("Percentage of variance explained (PVE)")
plt.xlabel("Principal component")
plt.title("Scree plot")
```

(b) Calculate the two first principal component loading vectors as follows :

— In `R` the result of `pr.out$rotation` is called the *rotation matrix*, the column vectors of this matrix are the principal components. For instance, the command `pr.out$rotation[,1]` returns the first principal component loading vector.

— In Python the command `loadings = pca.components_.T` allows to get a matrix denoted *loadings* containing the loading vectors associated to each principal component in columns and the features in rows.

In addition, you can support your interpretation plotting the first two principal loading vectors with the commands :

— In R you can obtain a biplot of the principal component scores and the principal component loadings.

```
pr.out$rotation=-pr.out$rotation
pr.out$x=-pr.out$x
biplot(pr.out, scale=0)
```

— In Python you can obtain the correlation circle plot by executing the commands :

```
fig,axis = plt.subplots(figsize=(5,5))
axis.set_xlim(-1,1)
axis.set_ylim(-1,1)
plt.plot([-1,1],[0,0],color="silver",linestyle="-",linewidth=1)
plt.plot([0,0],[-1,1],color="silver",linestyle="-",linewidth=1)
for j in range(0,4)
    plt.arrow(0, 0, loadings[j,0], loadings[j,1],
    head_width=0.02,width=0.001,color="red")
    plt.annotate(USArrests.columns[j],(loadings[j,0], loadings[j,1]))
cercle = plt.Circle((0,0),1,color='blue',fill=False)
axis.add_artist(cercle)
```

Measure the quality of representation of variables by interpreting the weight attributed to each variable by the first and second principal component loading vectors. Remind that for each loading vector the square of the weights sum up to 1. Present the results in a table.

(c) You can obtain the principal component scores (coordinates of the original data in the rotated coordinate system) as well as plot the observations by running :

— In `R`, you can use the command `pr.out$x`. Then you can plot the observations with the code `biplot(pr.out , scale =0)`.

— in Python, the get the new coordinates you should run the code : `USArrests_std_Phi=pca.transform(USArrests_std)` Next, to plot the obserations, you can use the following code :

```
fig = plt.figure(figsize = (10,10))
labels=["PC1","PC2","PC3","PC4"]
USArrests_std_Phi = pd.DataFrame(USArrests_std_Phi,index=USArrests.index,columns=labels)
plt.scatter(USArrests_std_Phi.PC1, USArrests_std_Phi.PC2)
plt.title("USArrests, PCA")
plt.xlabel("PC1")
plt.ylabel("PC2")
for obs in USArrests_std_Phi.index:
    plt.annotate(obs,(USArrests_std_Phi.PC1.loc[obs],USArrests_std_Phi.PC2.loc[obs]))
plt.plot([−3,3],[0,0],color='silver',linestyle="−",linewidth=1)
plt.plot([0,0],[2.5,−1.5],color='silver',linestyle="−",linewidth=1)
```

Interpret the new representation of the following states : *California*, *New Hampshire*, *Vermont* and *Virginia*. Give the new coordinates of the these states.

Finally, freely comment on / interpret the PCA plot if you see any remarkably facts.