Understanding Principle Component Analysis(PCA) step by step.

The Nobles · Follow
Published in Analytics Vidhya
4 min read · Jan 7, 2020

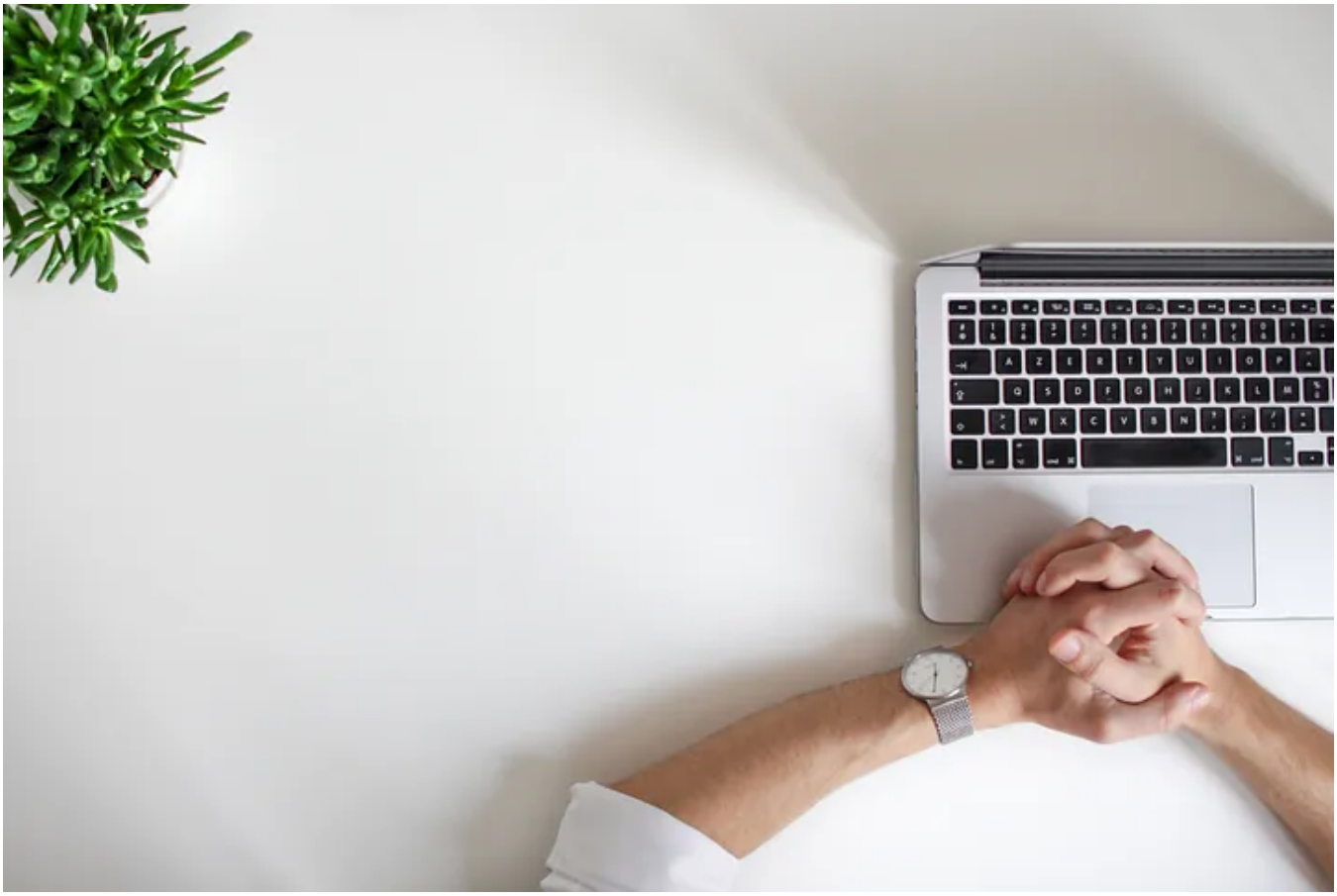


Photo by NordWood Themes on Unsplash

Introduction

Principal component analysis (PCA) is a statistical procedure that is used to reduce the dimensionality. It uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly

uncorrelated variables called principal components. It is often used as a dimensionality reduction technique.

Steps Involved in the PCA

Step 1: Standardize the dataset.

Step 2: Calculate the covariance matrix for the features in the dataset.

Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.

Step 4: Sort eigenvalues and their corresponding eigenvectors.

Step 5: Pick k eigenvalues and form a matrix of eigenvectors.

Step 6: Transform the original matrix.

Let's go to each step one by one.

1. Standardize the Dataset

Assume we have the below dataset which has 4 features and a total of 5 training examples.

f1	f2	f3	f4
1	2	3	4
5	5	6	7
1	4	2	3
5	3	2	1
8	1	2	2

Dataset matrix

First, we need to standardize the dataset and for that, we need to calculate the mean and standard deviation for each feature.

$$x_{new} = \frac{x - \mu}{\sigma}$$

Standardization formula

	f1	f2	f3	f4
μ =	4	3	3	3.4
σ =	3	1.58114	1.73205	2.30217

Mean and standard deviation before standardization

After applying the formula for each feature in the dataset is transformed as below:

f1	f2	f3	f4
-1	-0.63246	0	0.26062
0.33333	1.26491	1.73205	1.56374
-1	0.63246	-0.57735	-0.17375
0.33333	0	-0.57735	-1.04249
1.33333	-1.26491	-0.57735	-0.60812

Standardized Dataset

2. Calculate the covariance matrix for the whole dataset

The formula to calculate the covariance matrix:

For Population

$$Cov(x,y) = \frac{\sum (x_i - \bar{x}) * (y_i - \bar{y})}{N}$$

For Sample

$$Cov(x,y) = \frac{\sum (x_i - \bar{x}) * (y_i - \bar{y})}{(N-1)}$$

Covariance Formula

the covariance matrix for the given dataset will be calculated as below

	f1	f2	f3	f4
f1	var(f1)	cov(f1,f2)	cov(f1,f3)	cov(f1,f4)
f2	cov(f2,f1)	var(f2)	cov(f2,f3)	cov(f2,f4)
f3	cov(f3,f1)	cov(f3,f2)	var(f3)	cov(f3,f4)
f4	cov(f4,f1)	cov(f4,f2)	cov(f4,f3)	var(f4)

Since we have standardized the dataset, so the **mean for each feature is 0** and the standard deviation is 1.

var(f1) = $((-1.0-0)^2 + (0.33-0)^2 + (-1.0-0)^2 + (0.33-0)^2 + (1.33-0)^2)/5$
var (f1) = 0.8

cov(f1,f2) =
((-1.0–0)*(-0.632456-0) +
(0.33–0)*(1.264911-0) +
(-1.0–0)* (0.632456-0)+
(0.33–0)*(0.000000 -0)+
(1.33–0)*(-1.264911–0))/5
cov(f1,f2 = -0.25298

In the similar way be can calculate the other covariances and which will result in the below covariance matrix

	f1	f2	f3	f4
f1	0.8	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8	0.51121	0.4945
f3	0.03849	0.51121	0.8	0.75236
f4	-0.14479	0.4945	0.75236	0.8

covariance matrix (population formula)

3. Calculate eigenvalues and eigen vectors.

An **eigenvector** is a nonzero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding **eigenvalue** is the factor by which the eigenvector is scaled.

Let A be a square matrix (in our case the covariance matrix), ν a vector and λ a scalar that satisfies $Aν = λν$, then λ is called eigenvalue associated with eigenvector ν of A.
Rearranging the above equation,

$$Aν-λν =0 ; (A-λI)ν = 0$$

Since we have already know ν is a non- zero vector, only way this equation can be equal to zero, if

$$det(A-λI) = 0$$

	f1	f2	f3	f4
f1	0.8 - λ	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8- λ	0.51121	0.4945
f3	0.03849	0.51121	0.8 - λ	0.75236
f4	-0.14479	0.4945	0.75236	0.8 - λ

A-λI = 0

Solving the above equation = 0

λ = 2.51579324 , 1.0652885 , 0.39388704 , 0.02503121

Eigenvectors:

Solving the (A-λI)ν = 0 equation for ν vector with different λ values:

$$\begin{pmatrix} 0.800000-\lambda & -(0.252982) & 0.038490 & -(0.144791) \\ -(0.252982) & 0.800000-\lambda & 0.511208 & 0.494498 \\ 0.038490 & 0.511208 & 0.800000-\lambda & 0.752355 \\ -(0.144791) & 0.494498 & 0.752355 & 0.800000-\lambda \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = 0$$

For λ = *2.51579324, solving the above equation using Cramer's rule, the values for ν vector are*

v1 = 0.16195986
v2 = -0.52404813
v3 = -0.58589647
v4 = -0.59654663

Going by the same approach, we can calculate the eigen vectors for the other eigen values. We can from a matrix using the eigen vectors.

e1	e2	e3	e4
0.161960	-0.917059	-0.307071	0.196162
-0.524048	0.206922	-0.817319	0.120610
-0.585896	-0.320539	0.188250	-0.720099
-0.596547	-0.115935	0.449733	0.654547

eigenvectors(4 * 4 matrix)

4. Sort eigenvalues and their corresponding eigenvectors.

Since eigenvalues are already sorted in this case so no need to sort them again.

5. Pick k eigenvalues and form a matrix of eigenvectors

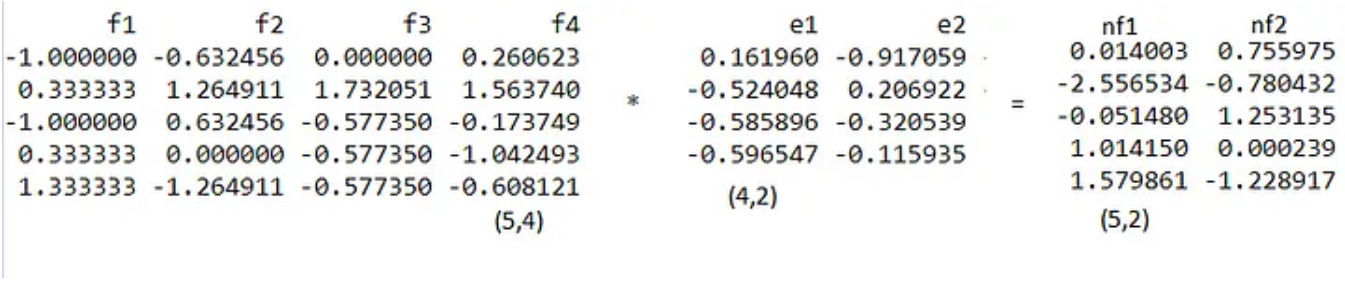
If we choose the top 2 eigenvectors, the matrix will look like this:

e1	e2
0.161960	-0.917059
-0.524048	0.206922
-0.585896	-0.320539
-0.596547	-0.115935

Top 2 eigenvectors(4*2 matrix)

6. Transform the original matrix.

Feature matrix * top k eigenvectors = Transformed Data



Data Transformation

Compare with sklearn library

```
import numpy as np
import pandas as pd
A = np.matrix([[1,2,3,4],
               [5,5,6,7],
               [1,4,2,3],
               [5,3,2,1],
               [8,1,2,2]])

df = pd.DataFrame(A,columns  = ['f1','f2','f3','f4'])
df_std  = (df - df.mean()) / (df.std())
n_components = 2
from sklearn.decomposition import PCA
pca = PCA(n_components=n_components)
principalComponents = pca.fit_transform(df_std)
principalDf = pd.DataFrame(data=principalComponents,columns=['nf'+str(i+1) for i in range(n_components)])
print(principalDf)
```

```
        nf1       nf2
0 -0.014003  0.755975
1  2.556534 -0.780432
2  0.051480  1.253135
3 -1.014150  0.000239
4 -1.579861 -1.228917
```

code snippet for PCA using Sklearn

The results are the same, only change in the direction of the PC1, which according to me, doesn't make any difference as mentioned here also. So we have successfully converted our data from 4 dimensional to 2 dimensional. PCA is mostly useful when data features are highly correlated.

Since this is my first blog, I am open to suggestions and please do check out the code version of above on my GitHub.

Data Science | Machine Learning | Pca | Statistics | Dimensionality Reduction