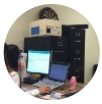
Principal Component Analysis (PCA)- Facial Recognition 😀🤓😳🤠



Hua Shi
Jul 14 · 7 min read

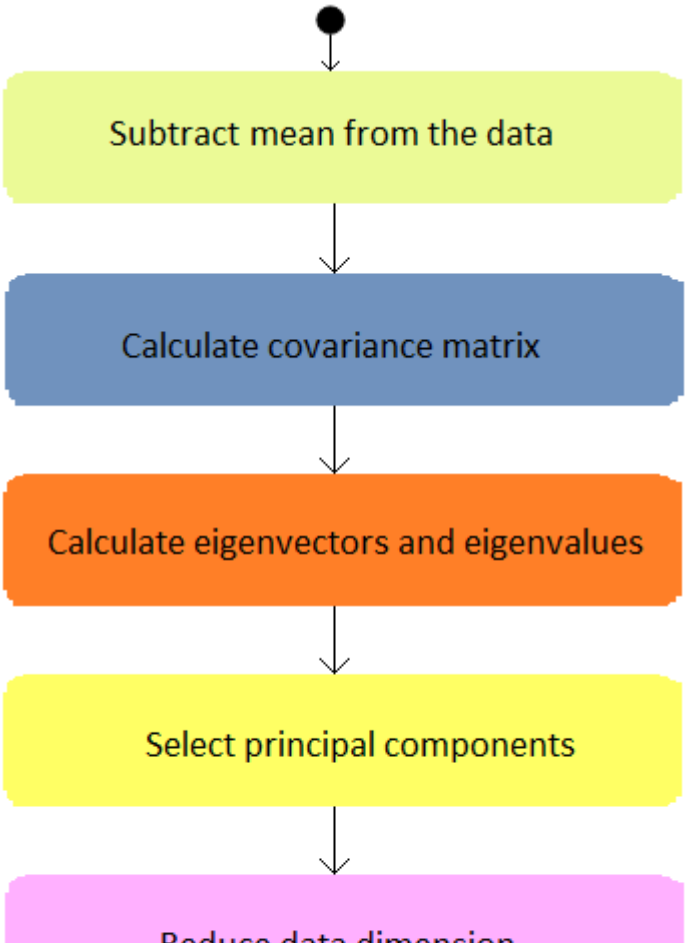
Nowadays facial recognition is widely used and it is commonly used for ID verification — Face ID. For example, Apple's Face ID is facial-recognition technology that launched on the iPhone X in 2017. Before your set up the Face ID, the iPhone will 'scan' your face from different angles and after you set it up, you are able to unlock your iPhones and it will also help you unlock some APPs. One more example, in some movies we saw that FIB, CIA, or similar investigation departments applied facial recognition technology to scan individuals' pictures and obtain their information. So the question is how does facial recognition work? — The technology of facial recognition is based on the information of facial images. Facial identity features are extracted and compared with known faces to identify different faces.

Definition

Facial recognition is a category of biometric software that maps an individual's facial features mathematically and stores the data as a faceprint. The software uses deep learning algorithms to compare a live capture or digital image to the stored faceprint in order to verify an individual's identity.

Principal Component Analysis (PCA)

PCA was invented in 1901 by Karl Pearson who is the creator of **Pearson's** Coefficient **Correlation which is a** mathematical method to measure the linear relationship between variables**.** After Karl Person, PCA has been developed many times. PCA is an algorithm of unsupervised learning and widely used to reduce the dimension of the data. PCA is an orthogonal linear transformation which means that PCA can transform data into a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first principal component, the second greatest variance on the second principal component, and so on. This is also called the process of dimensional reduction. The reason why it chooses the direction of bigger variance because it could significantly compress the dimension without losing a lot of information.



By Pablo Martin, Artelnics.

Code in Python

Data

There are some datasets for the facial recognition project. Here I just simply use the dataset from sklearn.

```
# Import necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.datasets import fetch_olivetti_faces

# Load the data
data = fetch_olivetti_faces()
data.keys()
>>> dict_keys(['data', 'images', 'target', 'DESCR'])

# declare inputs, target and images
inputs=data.data
target=data.target
images=data.images

inputs.shape
>>> (400, 4096)
```

The data from sklearn is ready to use which means that we do not need to do the part of data cleansing and data engineering. After we load the data we can see the data is a dictionary and there are four different keys — The “target” is an integer from 0 to 39 indicating the identity of the person pictured. The “DESCR” contains the attribute information. The ‘data’ is the 2D array (400 *4096) transformed data from “image”. The “image” contains 400 different matrics and each matric’s shape is 64 by 64 which is the “digital expression” of image. So if we reshape the “image” we can get “data”. The picture below can explain more about “reshape”. Array 1 is just like our “image” data

what I did here is I just reshaped the array 1 using each matric's shape 3*3 then simply use *reshape* function to transform the matrics into a 2D array — array 2. The reason why I explain it because the data from sklean and it is already cleaned up, however, if we have raw image data, we need to use packages *os* and *cv2* to load, resize, and reshape the image data.

```
array1= np.array([[[1,2,3]   , [4,5,6]  ,[7,8,9]],
                  [[10,11,12],[13,14,15],[16,17,18]],
                  [[19,20,21],[22,23,24],[25,26,27]]])
array1.shape
```

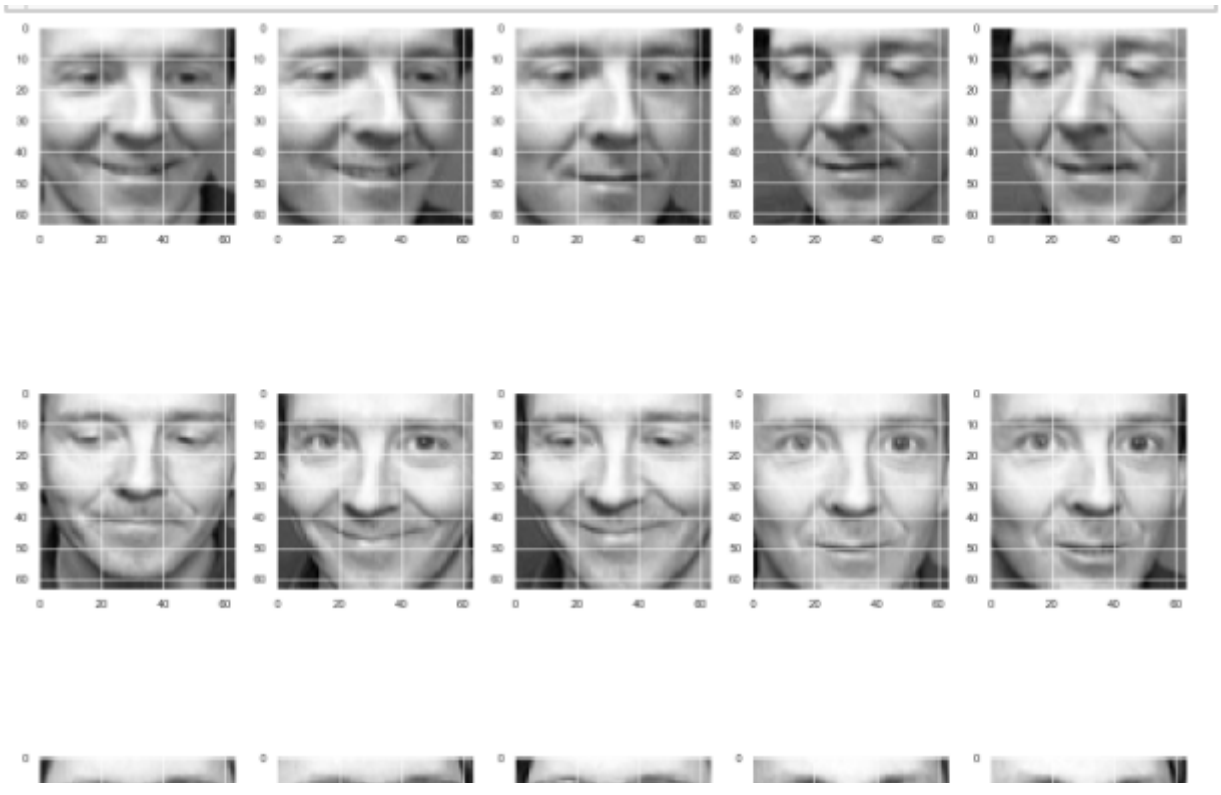
```
(3, 3, 3)
```

```
array2=array1.reshape(3,3*3)
array2
```

```
array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24, 25, 26, 27]])
```

Let us take look at some images from this sklearn dataset.

```
plt.figure(figsize=(25,30))
for i in range(30):
    plt.subplot(6,5,i+1)
    plt.imshow(data.images[i], cmap=plt.cm.gray)
plt.show()
```

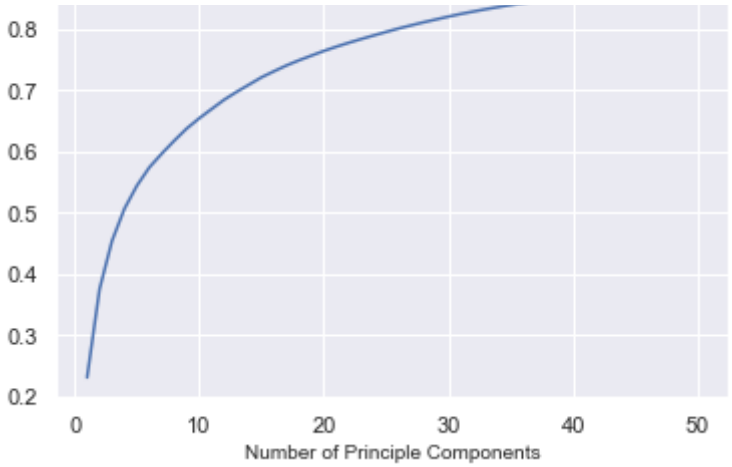


Credit to AT&T Laboratories Cambridge

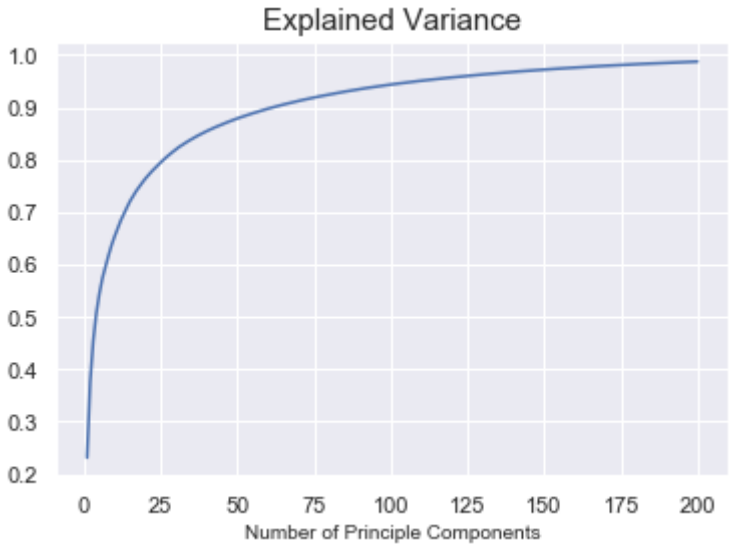
Each person represents one class and each class has 10 face images. All images for one person are different- some of them look at different directions and some of them wear glasses. However, after dimensional reduction, those images will also look different. Before we reduce the dimension, we also need to know how many components we need to set up.

PCA & Explained Variance

```
# import library
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

# train test split
x_train, x_test, y_train, y_test = train_test_split(inputs, target,
random_state=365)
# PCA transformation and reduce the dimension from 4096 to 50
pca = PCA(n_components=200, whiten=True)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

plt.plot(range(1,201), pca.explained_variance_ratio_.cumsum())
plt.title('Explained Variance',fontsize=15)
plt.xlabel('Number of Principle Components', fontsize=10)
plt.show()
```



`pca.explained_variance_ratio_` returns a vector of the variance explained by each dimension which also implies that the average distance between these points and the origin keeps growing when the dimension increases accordingly. Why this curve is important? Let us take a quick look at when *n_components=200*.



We can see that the ratio increased a lot when *n_components* is from 50 to 200 dimensions which means when data is compressed into 200 dimensions, those 200 dimensions can capture 99% of the overall variance. Briefly speaking, our inputs data has 4096 dimensions and if we compress the inputs data into 200 dimensions, we still can obtain most information from inputs data. Using this visualization method can help us to choose the number of *components* better.

Modeling

Support Vector Machines (SVM) are one of the most useful techniques in classification problems. SVM algorithm is commonly used for face recognition analysis. The code below is a simple SVM model. If you want to obtain higher accuracy you can apply GridSearch to tune the parameters. Even though best parameters were not applied in this model, the test accuracy already reached around 96%

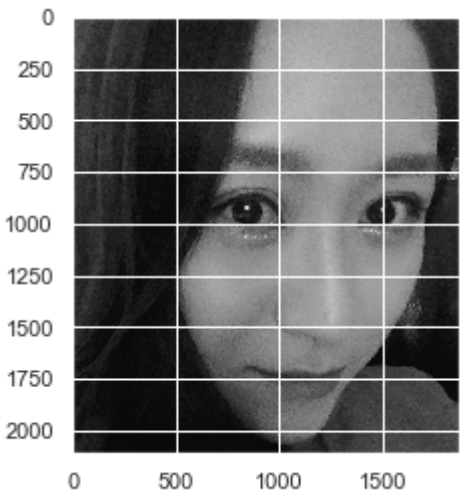
```
# fit the model
clf = svm.SVC(C=10, gamma=0.001)
%timeit clf.fit(X_train, y_train)

# obtian the accuracy
accuracy_train= clf.score(X_train, y_train)
accuracy_test= clf.score(X_test, y_test)
print('Accuracy – train data: {}'.format(accuracy_train))
print('Accuracy – test data : {}'.format( accuracy_test))

# prediction
test_pre=clf.predict(X_test)
train_pre=clf.predict(X_train)

# F1 _score
from sklearn.metrics import f1_score
f1_train=f1_score(y_train,train_pre,average='weighted')
f1_test=f1_score(y_test,test_pre,average='weighted')
print("f1 score – test data  : {}" .format(f1))

>>> 66.8 ms ± 2 ms per loop (mean ± std. dev. of 7 runs, 10 loops
each)
>>> Accuracy – train data: 1.0
>>> Accuracy – test data : 0.96
```

How to load your own image data and reshape them?

The first thing we need to do is to import packages — os and cv2. Then we need to find the path of the image folder. Next, we can choose to take a look at the first image from the folder before PCA.

```
# import packages
import cv2
import os

# find the path of your image folder
path='copy and paste the path of your image folder'
```

```
# check one image from the folder
for img in os.listdir(path):
   img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
    plt.imshow(img_array,cmap="gray")
    plt.show()
    break
```

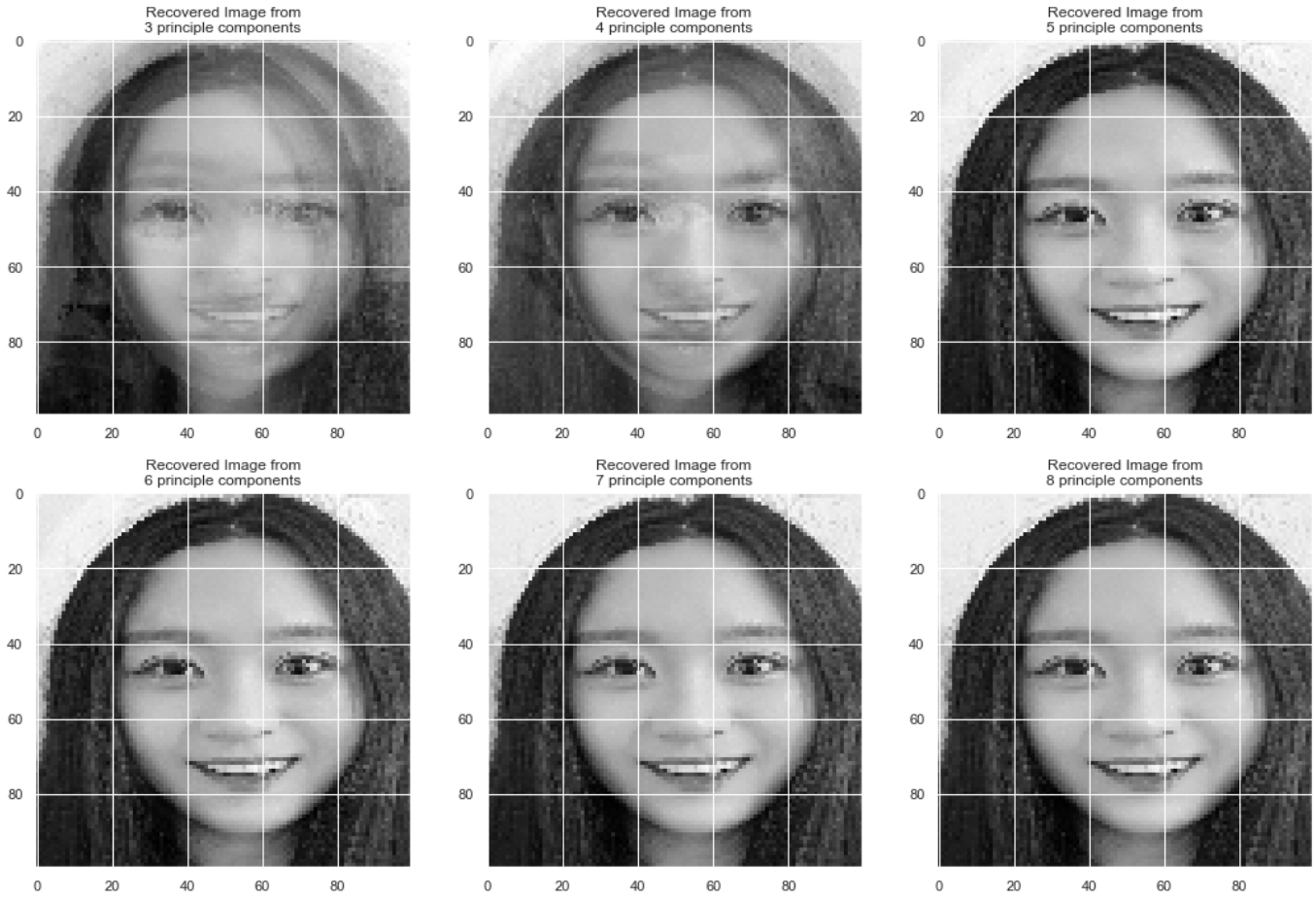


This is my selfie 😊

```
# create a function to create,load,resize, and re-color the data
def create_data(path):
    list_new_array=[]
    for img in os.listdir(path):
        try:

img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
            size=100
            new_array=cv2.resize(img_array,(size,size))
            list_new_array.append(new_array)
        except Exception as e:
            pass
    return list_new_array
```

Now it is time to reshape the data!! I already explained the idea behind the code in previous part in this blog. Let us take a look at the code. “hua” and “hua_reshape” are just like the sklearn face image data “image” and “data”. Then you can apply PCA and SVM with this data.

```
# I called the data is "hua" (my name)
hua=np.array(create_data(path))
hua_reshape=hua.reshape(8, 100*100).astype('float32')

plt.figure(figsize=(18,12))
for i in [3,4,5,6,7,8]:
    plt.subplot(2,3,i-2)
    n_feats = i
    pca = PCA(n_components=n_feats)
    pca.fit(hua_reshape)
    compressed = pca.transform(hua_reshape)
    plt.title('Recovered Image from\n{} principle
components'.format(n_feats))

plt.imshow(pca.inverse_transform(compressed[6]).reshape(100,100),
cmap=plt.cm.gray)
```



Recovered Image from different principle components

facial features mathematically and...

searchenterpriseai.techtarget.com

Principal component analysis

Given a collection of points in two, three, or higher dimensional space, a "best fitting" line can be defined as one...

en.wikipedia.org

Making sense of principal component analysis, eigenvectors & eigenvalues

begingroup$ After the excellent post by JD Long in this thread, I looked for a simple example, and the R code necessary...

stats.stackexchange.com

Understanding principal components analysis (PCA) | Neural Designer

By Pablo Martin, Artelnics. Principal components analysis (PCA) is a statistical technique that allows identifying...

www.neuraldesigner.com

21 Amazing Uses for Face Recognition - Facial Recognition Use Cases

While face recognition has been around in one form or another since the 1960s, recent technological developments have...

www.facefirst.com

Principal Component Facial Recognition Support Vector Machine Image Data