CPE 646 Final Project

Face Reconstruction via Principal Component Analysis

Professor: Hong Man
Student: Hsuan-Kai Liu

1. Abstract

Principal component analysis (PCA) is a technique that is useful for the compression and classification of data. The purpose is to reduce the dimensionality of a data set (sample) by finding a new set of variables, smaller than the original set of variables, that nonetheless retains most of the sample's information.

The objective of the project is to make computers construct human faces by principal component analysis (PCA) method. The main idea of using PCA for face recognition is to express the large 1-D vector of pixels constructed from the 2-D facial image into the compact principal components (PC) of the feature space as Figure 1. shows. And we can use different number of features to rebuild the images.

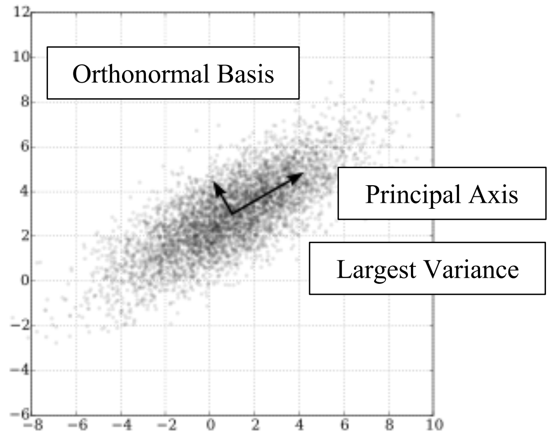
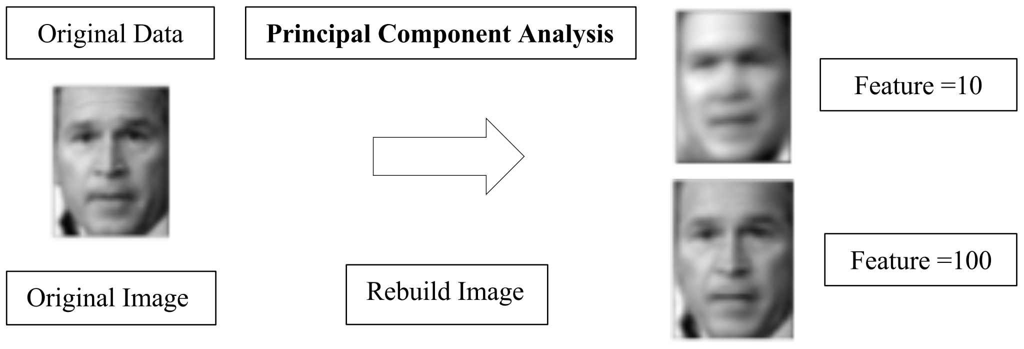


Figure 1. The concept of PCA

Application: Image Compression

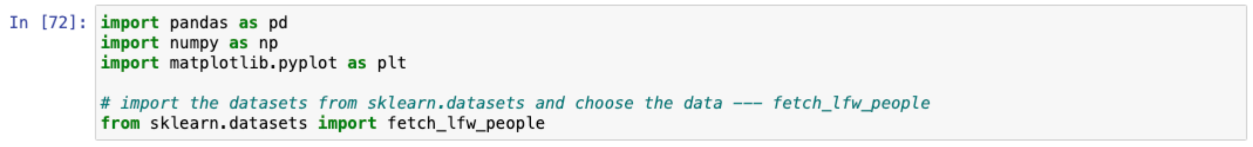
Scenario: Due to the limitation of size of communication (such as email), we have to compress the image we want. In the image compression process, we can decide how many features (critical information) we want to contain.



To construct the faces using eigenvalues, I used Python (Jupiter Notebook), NumPy, Pandas, and Matplotlib as my major tools. The most important part is I build the Principal Component Analysis (PCA) function by myself instead of using the already built-in package. Finally, I used different numbers of features to construct the images and compare the differences of them.

2. Descriptions of the work accomplished, with screenshots, photos and/or numerical results

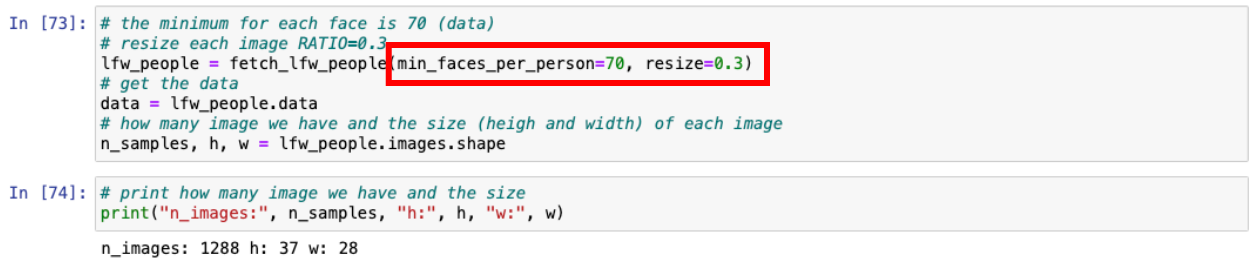
First of all, we have to load our data. I take the data from the Labeled Faces in the Wild by using sklearn dataset library, fetch_lfw_people. It is a matrix of 1288 * 5828, which means that there are 1288 images with a total 5828 pixels for each image.



```
In [72]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt

         # import the datasets from sklearn.datasets and choose the data --- fetch_lfw_people
         from sklearn.datasets import fetch_lfw_people
```

Using the min_faces_per_person = 70 parameter, we run fetch_lfw_people () to choose individuals' faces with 70 photographs, then resize the image with ratio 0.3 to the original images. The final output dataset is a 1288 x 1036 matrix for the data.

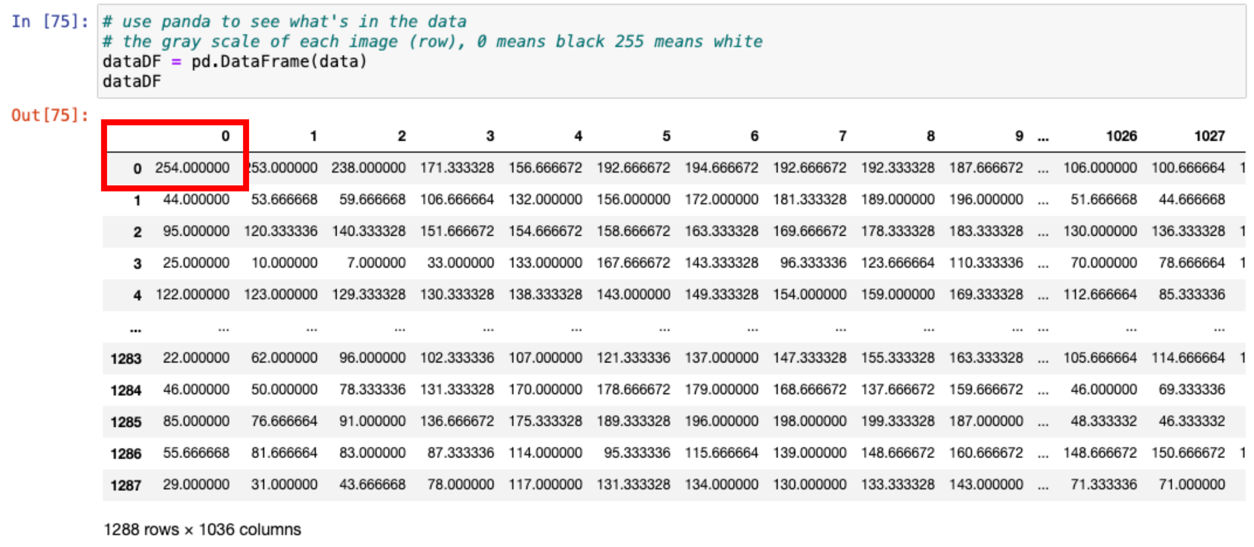


```
In [73]: # the minimum for each face is 70 (data)
         # resize each image RATIO=0.3
         lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.3)
         # get the data
         data = lfw_people.data
         # how many image we have and the size (heigh and width) of each image
         n_samples, h, w = lfw_people.images.shape

In [74]: # print how many image we have and the size
         print("n_images:", n_samples, "h:", h, "w:", w)

         n_images: 1288 h: 37 w: 28
```

I use DataFrame in Pandas to see the data in our images. The intensity of the black color (gray scale) is measured by the number in each cell of the matrix. The gray scale of each image (row) have the following meaning: 0 means black & 255 means white. The top left corner pixel of the human face is 254 in the illustration below. As a result, we can observe that the color in that area is white.



```
In [75]: # use panda to see what's in the data
         # the gray scale of each image (row), 0 means black 255 means white
         dataDF = pd.DataFrame(data)
         dataDF

Out[75]:
```

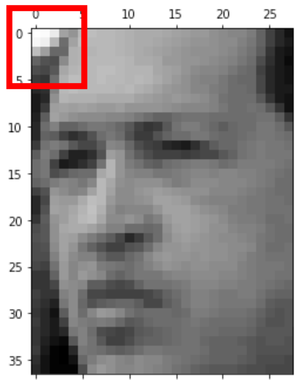
	0	1	2	3	4	5	6	7	8	9	...	1026	1027
0	254.000000	253.000000	238.000000	171.333328	156.666672	192.666672	194.666672	192.666672	192.333328	187.666672	...	106.000000	100.666664
1	44.000000	53.666668	59.666668	106.666664	132.000000	156.000000	172.000000	181.333328	189.000000	196.000000	...	51.666668	44.666668
2	95.000000	120.333336	140.333328	151.666672	154.666672	158.666672	163.333328	169.666672	178.333328	183.333328	...	130.000000	136.333328
3	25.000000	10.000000	7.000000	33.000000	133.000000	167.666672	143.333328	96.333336	123.666664	110.333336	...	70.000000	78.666664
4	122.000000	123.000000	129.333328	130.333328	138.333328	143.000000	149.333328	154.000000	159.000000	169.333328	...	112.666664	85.333336
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1283	22.000000	62.000000	96.000000	102.333336	107.000000	121.333336	137.000000	147.333328	155.333328	163.333328	...	105.666664	114.666664
1284	46.000000	50.000000	78.333336	131.333328	170.000000	178.666672	179.000000	168.666672	137.666672	159.666672	...	46.000000	69.333336
1285	85.000000	76.666664	91.000000	136.666672	175.333328	189.333328	196.000000	198.000000	199.333328	187.000000	...	48.333332	46.333332
1286	55.666668	81.666664	83.000000	87.333336	114.000000	95.333336	115.666664	139.000000	148.666672	160.666672	...	148.666672	150.666672
1287	29.000000	31.000000	43.666668	78.000000	117.000000	131.333328	134.000000	130.000000	133.333328	143.000000	...	71.333336	71.000000

1288 rows × 1036 columns

```
In [76]: # we can see the information of each image by changing the data[] number
         # if we choose image 0, as we can see, on the top left corner, it's almost white
         # so the corresponding number in the above table is 254
         plt.gray()
         plt.matshow(data[0].reshape(h,w))

Out[76]: <matplotlib.image.AxesImage at 0x7f8011910c40>

         <Figure size 432x288 with 0 Axes>
```



Principal Component Analysis:

In a bunch of data, the major purpose of PCA is to project these data to another basis to reduce the dimension (e.g. from 2-D to 1-D). During the projection process, we should keep the variance being the biggest value and keep the errors (between new positions and old positions) being the smallest value. Therefore, we can keep the property (difference) of the original data as possible as we can.

In the following figures, the blue dots represent the original data, the red dots on the axis represent the new projection position, the red lines between blue dots and new basis (axis) represent the errors and the scatter intensity of red dots represent the variance which should be maximized.

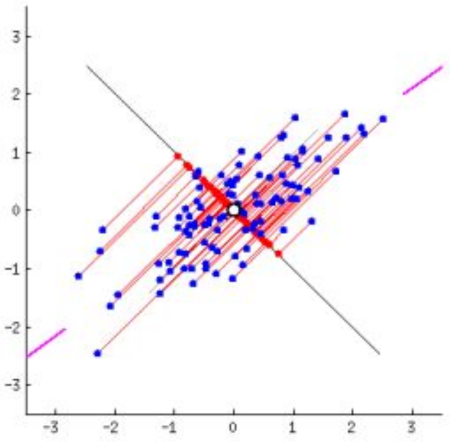


Figure 2. Variances are small

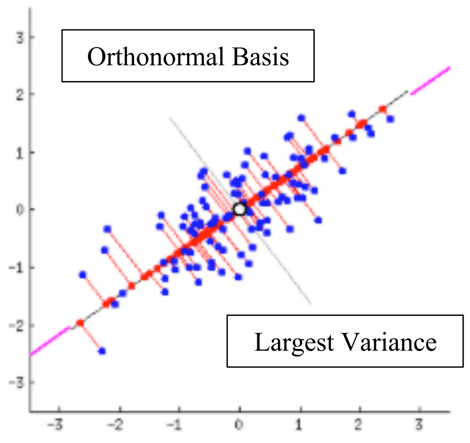


Figure 3. Variances are big (Desired)

Suppose we have an dataset x. Using the PCA process, I use the function

$$\tilde{x} = m + a_k e$$

as new position ($\tilde{x}$) on new basis, m means the mean value of original data x (we can use the concept y = a*x + b to better understand the formula) and e means the basis vector.

The purpose of PCA is to minimize the error and maximize the variance. Hence, we can define

$$Error\ E(a_k, e) = \sum_{k=1}^{n} \|(m + a_k e) - x_k\|^2$$

To solve a_k, I set

$$\frac{\partial E}{\partial a_k} = 0$$

Then we will get

$$a_k = e^T(x_k - m)$$

And we can define Scatter Matrix S to represent the variance of the total data

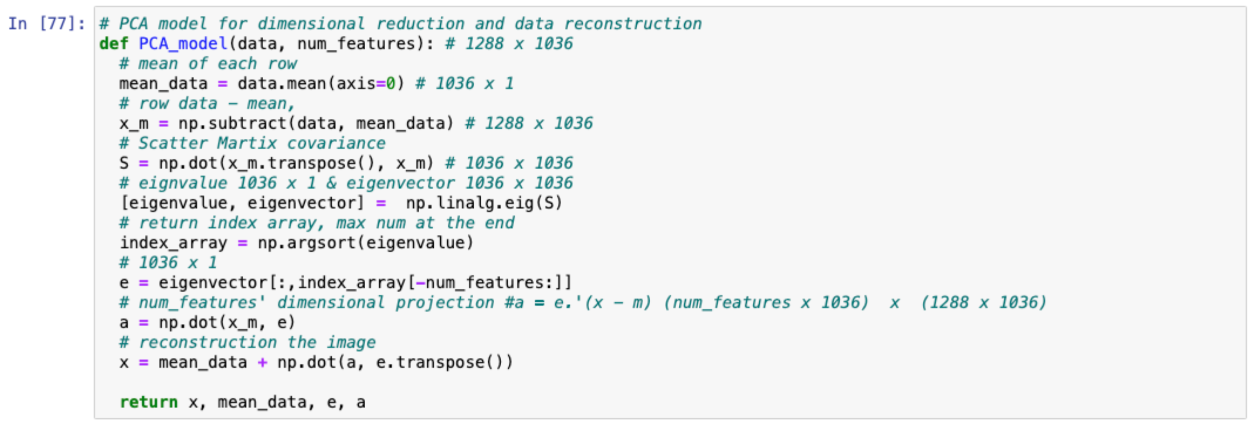
$$S = \sum_{k=1}^{n} (x_k - m)^T\,(x_k - m)$$

Finally, we have to solve the eigenvalues and eigenvectors by using Lagrange Multipliers

$$Se = \lambda e$$

Select *e* as the eigenvector that corresponding to the largest eigenvalue of the scatter matrix

Below is the corresponding code in Jupiter Notebook

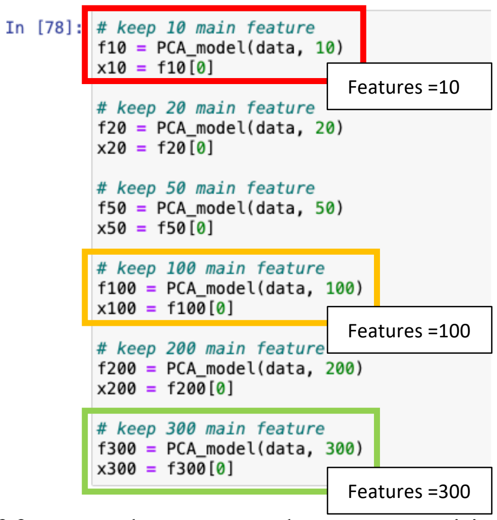


```
In [77]: # PCA model for dimensional reduction and data reconstruction
         def PCA_model(data, num_features): # 1288 x 1036
           # mean of each row
           mean_data = data.mean(axis=0) # 1036 x 1
           # row data - mean,
           x_m = np.subtract(data, mean_data) # 1288 x 1036
           # Scatter Martix covariance
           S = np.dot(x_m.transpose(), x_m) # 1036 x 1036
           # eignvalue 1036 x 1 & eigenvector 1036 x 1036
           [eigenvalue, eigenvector] =  np.linalg.eig(S)
           # return index array, max num at the end
           index_array = np.argsort(eigenvalue)
           # 1036 x 1
           e = eigenvector[:,index_array[-num_features:]]
           # num_features' dimensional projection #a = e.'(x - m) (num_features x 1036)  x  (1288 x 1036)
           a = np.dot(x_m, e)
           # reconstruction the image
           x = mean_data + np.dot(a, e.transpose())

           return x, mean_data, e, a
```

I calculate the mean of all image features to get a 1*1036 data as our mean data. Then I try to calculate the scattered matrix of differences between all image data and mean data. Therefore, I could get eigenvalues and eigenvectors from the scattered matrix in order to get the importance of 1036 features.

By sorting the eigenvalues, I get the highest eigenvalue indexes and corresponding eigenvectors as the e vectors for image data projections. Then I can calculate the projection of the image data with e dimensions, and reconstruct the image with e main features.



```
In [78]: # keep 10 main feature
         f10 = PCA_model(data, 10)
         x10 = f10[0]

         # keep 20 main feature
         f20 = PCA_model(data, 20)
         x20 = f20[0]

         # keep 50 main feature
         f50 = PCA_model(data, 50)
         x50 = f50[0]

         # keep 100 main feature
         f100 = PCA_model(data, 100)
         x100 = f100[0]

         # keep 200 main feature
         f200 = PCA_model(data, 200)
         x200 = f200[0]

         # keep 300 main feature
         f300 = PCA_model(data, 300)
         x300 = f300[0]
```

As the final result shows, once I increase the number of features that means I keep more critical information of the images. The following Figure 4. shows the number of features from 10 to 300.

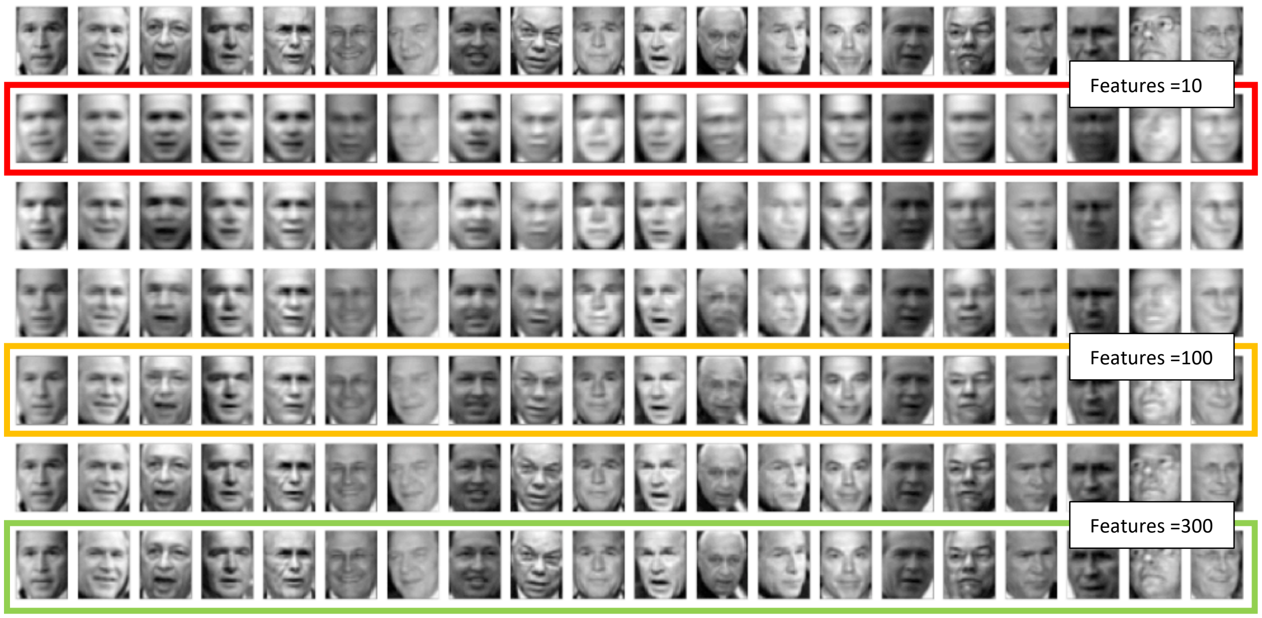


Figure 4. Different number of features

3. Descriptions of how to compile and run the submitted code

1. Use Jupyter Notebook to run CPE 646 Final Project.ipynb.
2. Execute the code by line to get the result at the end of the code.
3. Note: It may take some time to run code since there are 140 images need to be processed.

4. In the case of a team project, specify the contribution from each group member. Contribution less than a reasonable percentage will result in zero credit for the project

In the past few weeks, I was hesitated to choose which topics should I pick as my final project. In the beginning, I chose LDA transformation as my topic because it can better avoid the problem of PCA. The reason I chose PCA is that I still think PCA is the most basic concept in machine learning. Although it may not be good for discrimination purpose, the resulting components are good representation of the data in each class. In my opinion, how to rebuild the data that is similar to the original one is very important in Machine Learning field. Also, the application such as image compression is very interesting. Therefore, I chose PCA as my final project.

During the research process, the most important thing is build the PCA by myself. Although there do have some built-in functions I can use, I still think derive the formulas and understand the fundamental concepts by myself can better help me to realize the essence of PCA. After that, by using Python, I can make the mathematical formulas visualizable. Understanding the concept of gray scale, how does the number of features impact the final results, those are very treasurable experience for me.

5. Reference

[1] Real World Datasets

https://scikit-learn.org/stable/datasets/real_world.html

[2] Faces recognition example using eigenfaces and SVMs

https://scikit-learn.sourceforge.net/0.8/auto_examples/applications/face_recognition.html

[3] PCA: Labeled Faces in the Wild (LFW) datasets

https://www.stats.ox.ac.uk/~sejdinov/teaching/dmml17/PCA_eigenfaces.html#PCA:-Labeled-Faces-in-the-Wild-(LFW)-dataset

[4] Labeled Faces in the Wild http://vis-www.cs.umass.edu/lfw/

[5] an introduction to Principal Component Analysis

https://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/PrincipalComponentAnalysis.pdf

[6] PCA

https://ithelp.ithome.com.tw/articles/10211877

[7] [筆記]主成分分析(PCA)

https://www.zhihu.com/question/41120789