Guided Assignment On Feature Engineering

Name: **Shreya Shrivastava**

Course: **Artificial Intelligence And Machine Learning**

Batch Four

Duration: **12 Months**

Problem statement: Using **PCA** create a face recognition system that gives access to only certain people. To implement this, you can use **LFW_peoples dataset** provided in the **scikit-learn** library. Given this dataset, use only those classes that have a minimum (**use min_faces_per_person = 70, resize = 0.4**) 70 images (should give you only 11 classes). Given this subset of images, apply PA to obtain the corresponding eigen face for each class. You can additionally **train a classifier** for recognition purpose.

Prerequisites:

The libraries as well as things required in order for the program to work:

I. **Python 3.6** : The following url https://www.python.org/downloads/ can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/. Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic. Second option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this url : https://www.anaconda.com/download/

II. **ADDITIONAL PACKAGES :** You will also need to download and install below 3 packages- numpy, scikit learn and matplotlib after you install either python or anaconda from the steps above. If you have chosen to install python 3.6,then run the following commands in command prompt/terminal to install these packages :

NUMPY: pip install -U numpy

MATPLOTLIB: pip install -U matplotlib

SCIKIT LEARN: pip install -U scikit-learn

If using Anaconda then run the following commands in anaconda prompt to install these packages:

NUMPY: conda install -c anaconda numpy

MATPLOTLIB: conda install -c anaconda matplotlib

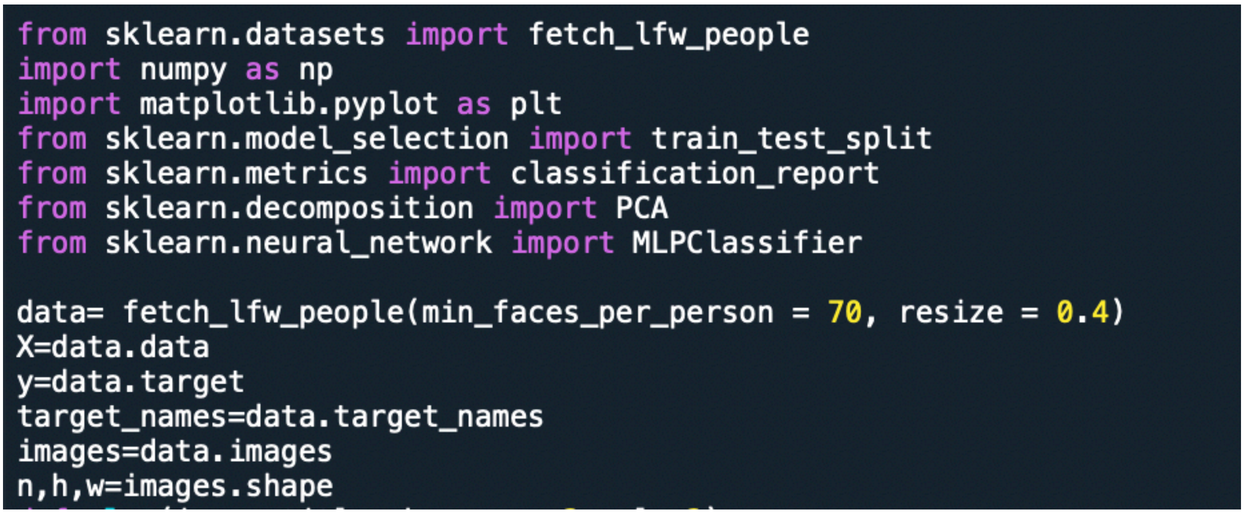
SCIKIT LEARN: conda install -c scikit-learn

III. METHODS USED:

A. PRINCIPAL COMPONENT ANALYSIS

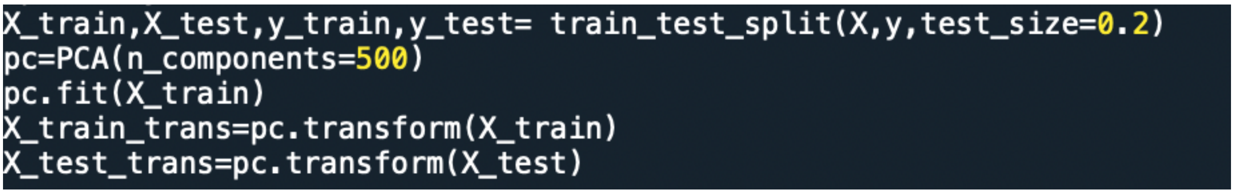
THE PROJECT :

1. Importing the libraries and loading the **LFW_peoples dataset.** We then proceed to load the data as well as the labels and store them in X and y respectively. Storing the dimensions of the images as well.



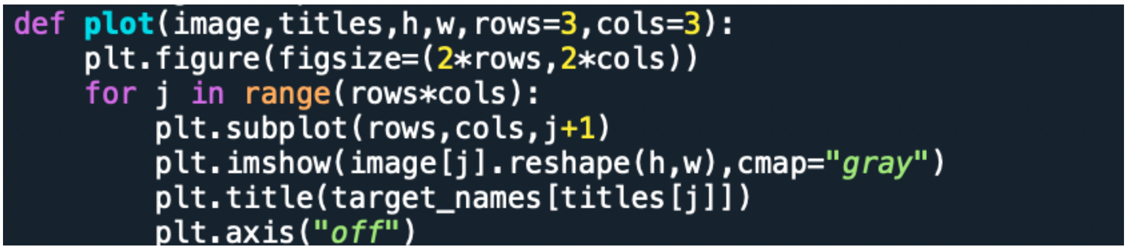
```
from sklearn.datasets import fetch_lfw_people
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier

data= fetch_lfw_people(min_faces_per_person = 70, resize = 0.4)
X=data.data
y=data.target
target_names=data.target_names
images=data.images
n,h,w=images.shape
```

2. Splitting the data into training and testing respectively. Then we perform PCA on the training set and store the transformed dataset.



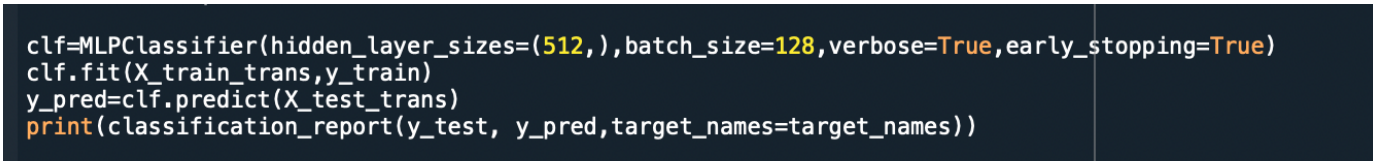
```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2)
pc=PCA(n_components=500)
pc.fit(X_train)
X_train_trans=pc.transform(X_train)
X_test_trans=pc.transform(X_test)
```

3. Plotting the original dataset.

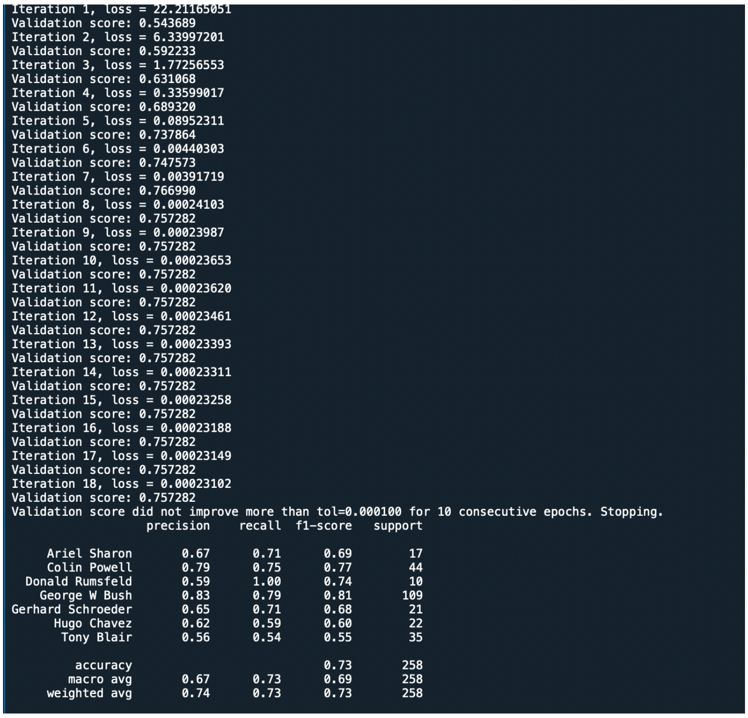


```
def plot(image,titles,h,w,rows=3,cols=3):
    plt.figure(figsize=(2*rows,2*cols))
    for j in range(rows*cols):
        plt.subplot(rows,cols,j+1)
        plt.imshow(image[j].reshape(h,w),cmap="gray")
        plt.title(target_names[titles[j]])
        plt.axis("off")
```

4. Training a MLP classifier and reporting the performance.



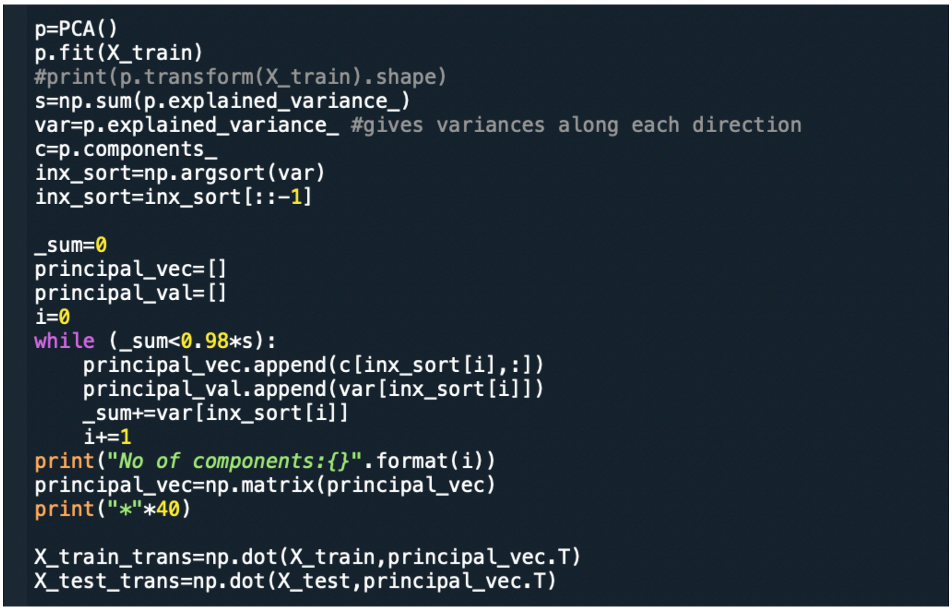
```
clf=MLPClassifier(hidden_layer_sizes=(512,),batch_size=128,verbose=True,early_stopping=True)
clf.fit(X_train_trans,y_train)
y_pred=clf.predict(X_test_trans)
print(classification_report(y_test, y_pred,target_names=target_names))
```



```
Iteration 1, loss = 22.21165051
Validation score: 0.543689
Iteration 2, loss = 6.33997201
Validation score: 0.592233
Iteration 3, loss = 1.77256553
Validation score: 0.631068
Iteration 4, loss = 0.33599017
Validation score: 0.689320
Iteration 5, loss = 0.08952311
Validation score: 0.737864
Iteration 6, loss = 0.00440303
Validation score: 0.747573
Iteration 7, loss = 0.00391719
Validation score: 0.766990
Iteration 8, loss = 0.00024103
Validation score: 0.757282
Iteration 9, loss = 0.00023987
Validation score: 0.757282
Iteration 10, loss = 0.00023653
Validation score: 0.757282
Iteration 11, loss = 0.00023620
Validation score: 0.757282
Iteration 12, loss = 0.00023461
Validation score: 0.757282
Iteration 13, loss = 0.00023393
Validation score: 0.757282
Iteration 14, loss = 0.00023311
Validation score: 0.757282
Iteration 15, loss = 0.00023258
Validation score: 0.757282
Iteration 16, loss = 0.00023188
Validation score: 0.757282
Iteration 17, loss = 0.00023149
Validation score: 0.757282
Iteration 18, loss = 0.00023102
Validation score: 0.757282
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
                   precision    recall  f1-score   support

     Ariel Sharon       0.67      0.71      0.69        17
     Colin Powell       0.79      0.75      0.77        44
  Donald Rumsfeld       0.59      1.00      0.74        10
    George W Bush       0.83      0.79      0.81       109
Gerhard Schroeder       0.65      0.71      0.68        21
      Hugo Chavez       0.62      0.59      0.60        22
       Tony Blair       0.56      0.54      0.55        35

         accuracy                           0.73       258
        macro avg       0.67      0.73      0.69       258
     weighted avg       0.74      0.73      0.73       258
```

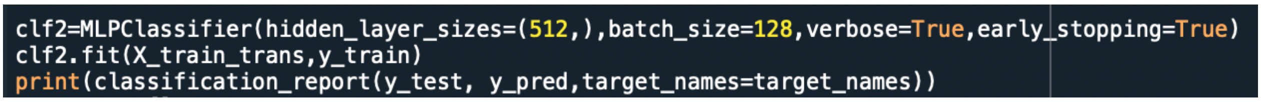
OUTPUT

5. Given this subset of images, we use PCA to get the principal vectors and values.

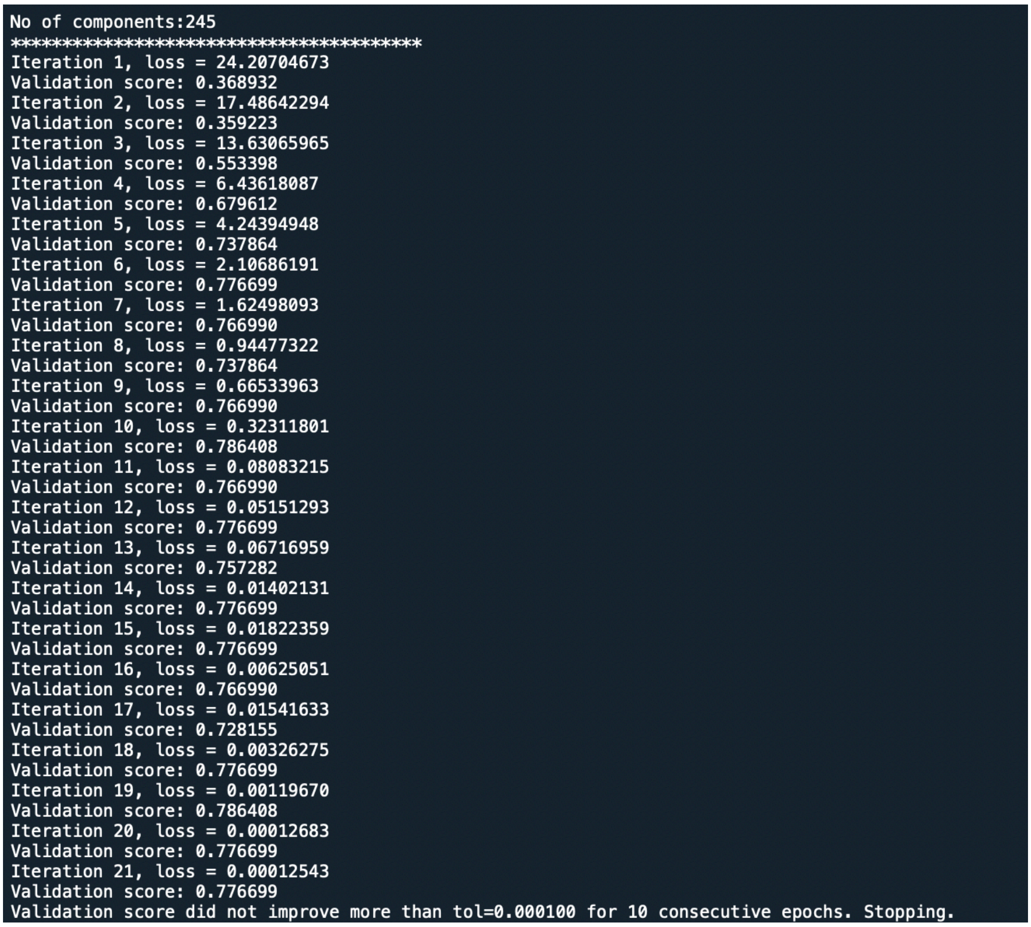


```
p=PCA()
p.fit(X_train)
#print(p.transform(X_train).shape)
s=np.sum(p.explained_variance_)
var=p.explained_variance_ #gives variances along each direction
c=p.components_
inx_sort=np.argsort(var)
inx_sort=inx_sort[::-1]

_sum=0
principal_vec=[]
principal_val=[]
i=0
while (_sum<0.98*s):
    principal_vec.append(c[inx_sort[i],:])
    principal_val.append(var[inx_sort[i]])
    _sum+=var[inx_sort[i]]
    i+=1
print("No of components:{}".format(i))
principal_vec=np.matrix(principal_vec)
print("*"*40)

X_train_trans=np.dot(X_train,principal_vec.T)
X_test_trans=np.dot(X_test,principal_vec.T)
```

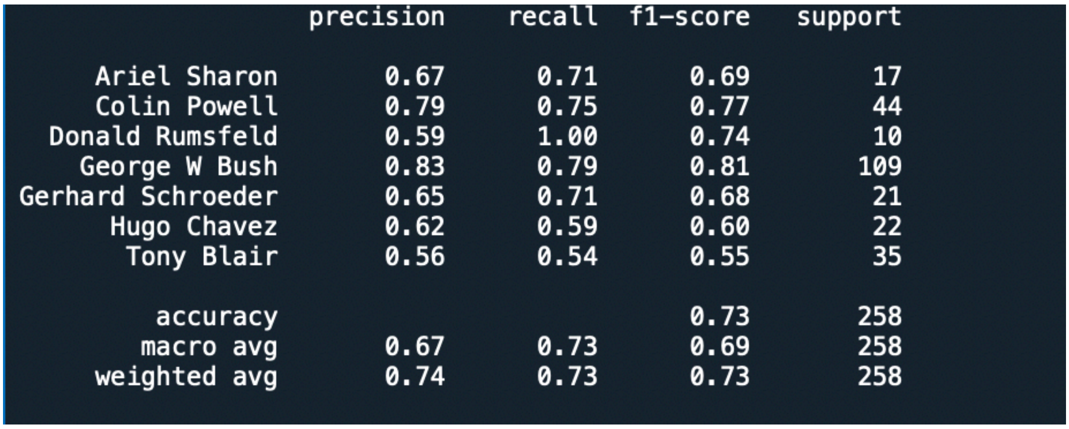
6. Training a MLP Classifier on the transformed dataset.



```
clf2=MLPClassifier(hidden_layer_sizes=(512,),batch_size=128,verbose=True,early_stopping=True)
clf2.fit(X_train_trans,y_train)
print(classification_report(y_test, y_pred,target_names=target_names))
```

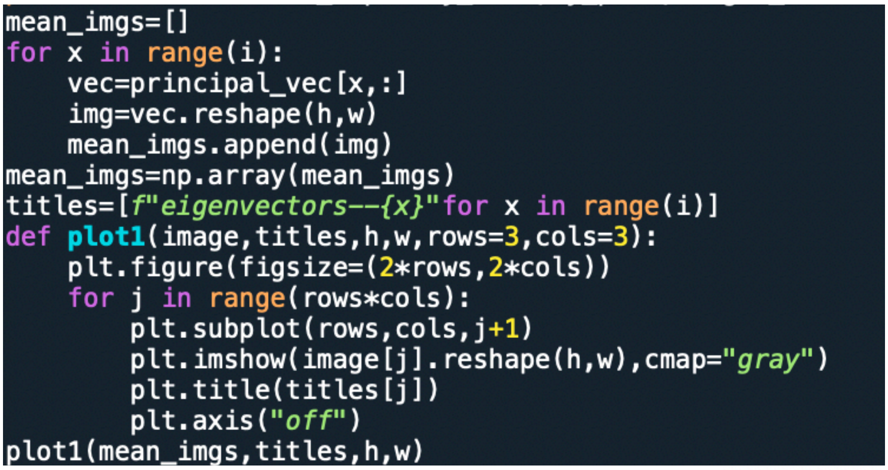


```
No of components:245
****************************************
Iteration 1, loss = 24.20704673
Validation score: 0.368932
Iteration 2, loss = 17.48642294
Validation score: 0.359223
Iteration 3, loss = 13.63065965
Validation score: 0.553398
Iteration 4, loss = 6.43618087
Validation score: 0.679612
Iteration 5, loss = 4.24394948
Validation score: 0.737864
Iteration 6, loss = 2.10686191
Validation score: 0.776699
Iteration 7, loss = 1.62498093
Validation score: 0.766990
Iteration 8, loss = 0.94477322
Validation score: 0.737864
Iteration 9, loss = 0.66533963
Validation score: 0.766990
Iteration 10, loss = 0.32311801
Validation score: 0.786408
Iteration 11, loss = 0.08083215
Validation score: 0.766990
Iteration 12, loss = 0.05151293
Validation score: 0.776699
Iteration 13, loss = 0.06716959
Validation score: 0.757282
Iteration 14, loss = 0.01402131
Validation score: 0.776699
Iteration 15, loss = 0.01822359
Validation score: 0.776699
Iteration 16, loss = 0.00625051
Validation score: 0.766990
Iteration 17, loss = 0.01541633
Validation score: 0.728155
Iteration 18, loss = 0.00326275
Validation score: 0.776699
Iteration 19, loss = 0.00119670
Validation score: 0.786408
Iteration 20, loss = 0.00012683
Validation score: 0.776699
Iteration 21, loss = 0.00012543
Validation score: 0.776699
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```



```
                   precision    recall  f1-score   support

     Ariel Sharon       0.67      0.71      0.69        17
     Colin Powell       0.79      0.75      0.77        44
  Donald Rumsfeld       0.59      1.00      0.74        10
    George W Bush       0.83      0.79      0.81       109
Gerhard Schroeder       0.65      0.71      0.68        21
      Hugo Chavez       0.62      0.59      0.60        22
       Tony Blair       0.56      0.54      0.55        35

         accuracy                           0.73       258
        macro avg       0.67      0.73      0.69       258
     weighted avg       0.74      0.73      0.73       258
```

7.Getting the corresponding eigenface and plotting them.

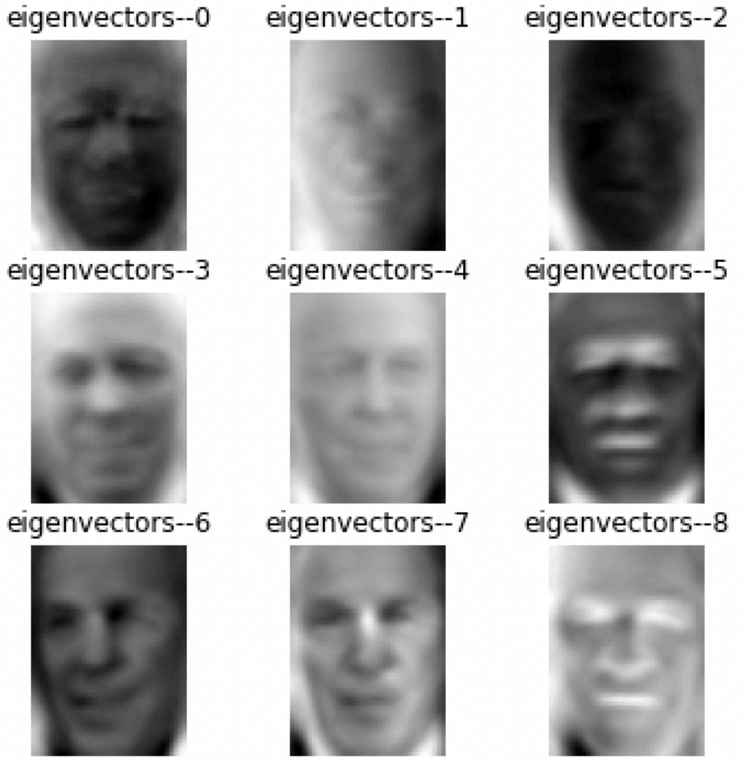


```
mean_imgs=[]
for x in range(i):
    vec=principal_vec[x,:]
    img=vec.reshape(h,w)
    mean_imgs.append(img)
mean_imgs=np.array(mean_imgs)
titles=[f"eigenvectors--{x}"for x in range(i)]
def plot1(image,titles,h,w,rows=3,cols=3):
    plt.figure(figsize=(2*rows,2*cols))
    for j in range(rows*cols):
        plt.subplot(rows,cols,j+1)
        plt.imshow(image[j].reshape(h,w),cmap="gray")
        plt.title(titles[j])
        plt.axis("off")
plot1(mean_imgs,titles,h,w)
```

Colin Powell Ariel Sharon Colin Powell

George W Bush Gerhard Schroeder George W Bush

ORIGINAL PHOTOS



eigenvectors--0 eigenvectors--1 eigenvectors--2

eigenvectors--3 eigenvectors--4 eigenvectors--5

eigenvectors--6 eigenvectors--7 eigenvectors--8

CORRESPONDING EIGEN-FACES