Guided Project Report

Face detection and recognition

Name: Shruti Verma
Course: AI and ML
(Batch 4)
Duration: 10 months

Problem Statement: Build a machine learning model using PCA for face detection and recognition.

Prerequisites

What things you need to install the software and how to install them:

Python 3.8 or higher versions This setup requires that your machine has latest version of python. 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 and easier 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/ You will also need to download and install below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.8 then run below commands in command prompt/terminal to install these packages pip install -U scikit-learn pip install numpy pip install scipy if you have chosen to install anaconda then run below commands in anaconda prompt to install these packages conda install -c scikit-learn conda install -c anaconda numpy conda install -c anaconda scipy

Dataset used

The data source is LFW_peoples(labelled faces in wild) dataset provided in the scikit-learn library. The dataset is used with the condition, classes that have a minimum (use min_faces_per_person = 70, resize = 0.4) 70 images. The data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name

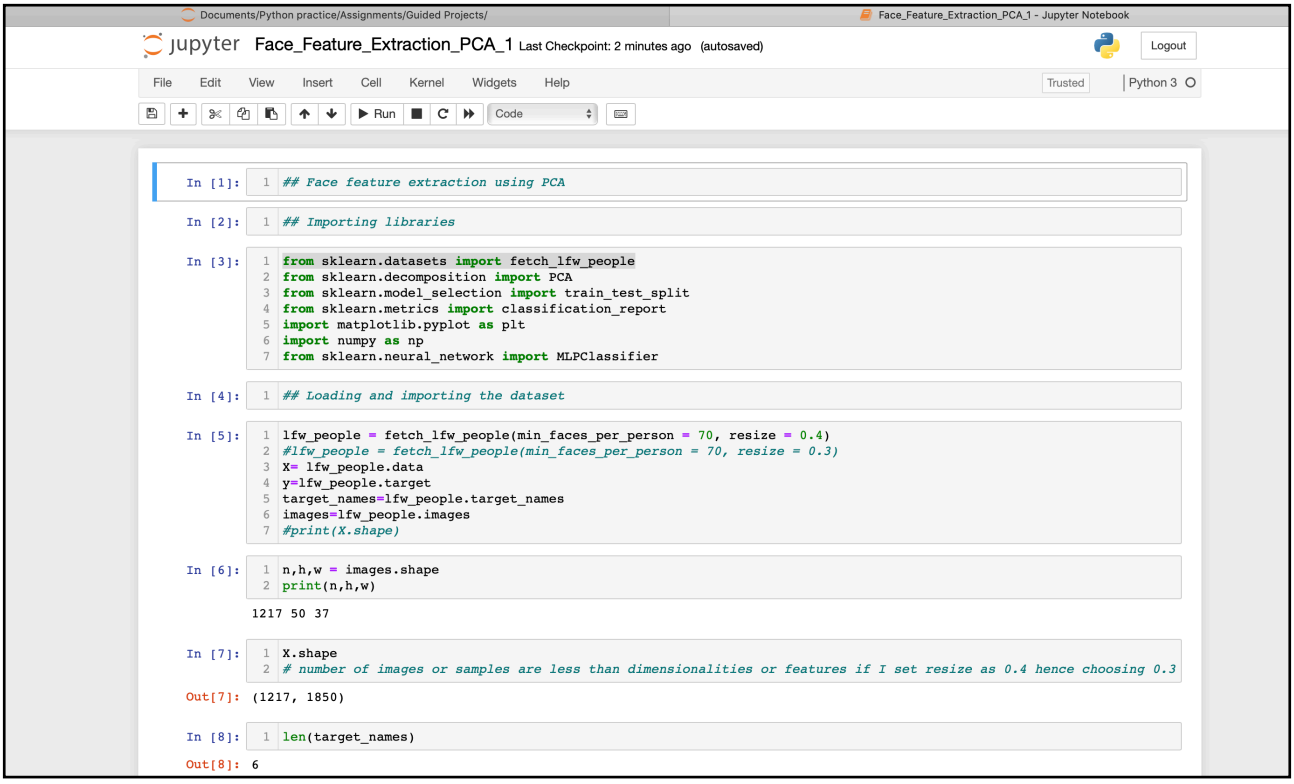
of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set. There are close 6,000 dimensionality.

Method used for detection

Principal component analysis

Flatten image dataset -> PCA -> MLP classification

Importing the libraries and capturing images:



Plotting the images

Documents/Python practice/Assignments/Guided Projects/ | Face_Feature_Extraction_PCA_1 - Jupyter Notebook

jupyter Face_Feature_Extraction_PCA_1 Last Checkpoint: 8 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

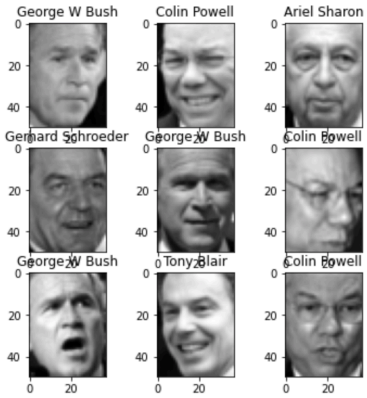
Run Code

In [8]:
```
len(target_names)
```
Out[8]: 6

In [9]:
```
np.unique(y,return_counts=True)
```
Out[9]: (array([0, 1, 2, 3, 4, 5]), array([77, 236, 121, 530, 109, 144]))

In [10]:
```
## plotting the data
```

In [11]:
```
def plot_lfw(images, titles,h,w,rows=3,cols=3):
    plt.figure(figsize=(2*cols,2*rows))
    for i in range(rows*cols):
        plt.subplot(rows,cols,i+1)
        plt.imshow(images[i].reshape(h,w),cmap='gray')
        plt.title(target_names[titles[i]])
plot_lfw(X,y,h,w)
```



Splitting the training and testing data into 90 :10.

This is hyper parameter and can be varied to 85:15 or 80:20 ratio also.

Documents/Python practice/Assignments/Guided Projects/ | Face_Feature_Extraction_PCA_1 - Jupyter Notebook

jupyter Face_Feature_Extraction_PCA_1 Last Checkpoint: 12 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [12]:
```
# spilting the train and test data
```

In [13]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.1)
```

In [14]:
```
X_train.shape
```

Out[14]:
```
(1095, 1850)
```

In [15]:
```
pca=PCA(n_components=120)
pca.fit(X_train)
```

Out[15]:
```
PCA(n_components=120)
```

In [16]:
```
X_train_trans_1 = pca.transform(X_train)
X_test_trans_1 = pca.transform(X_test)
```

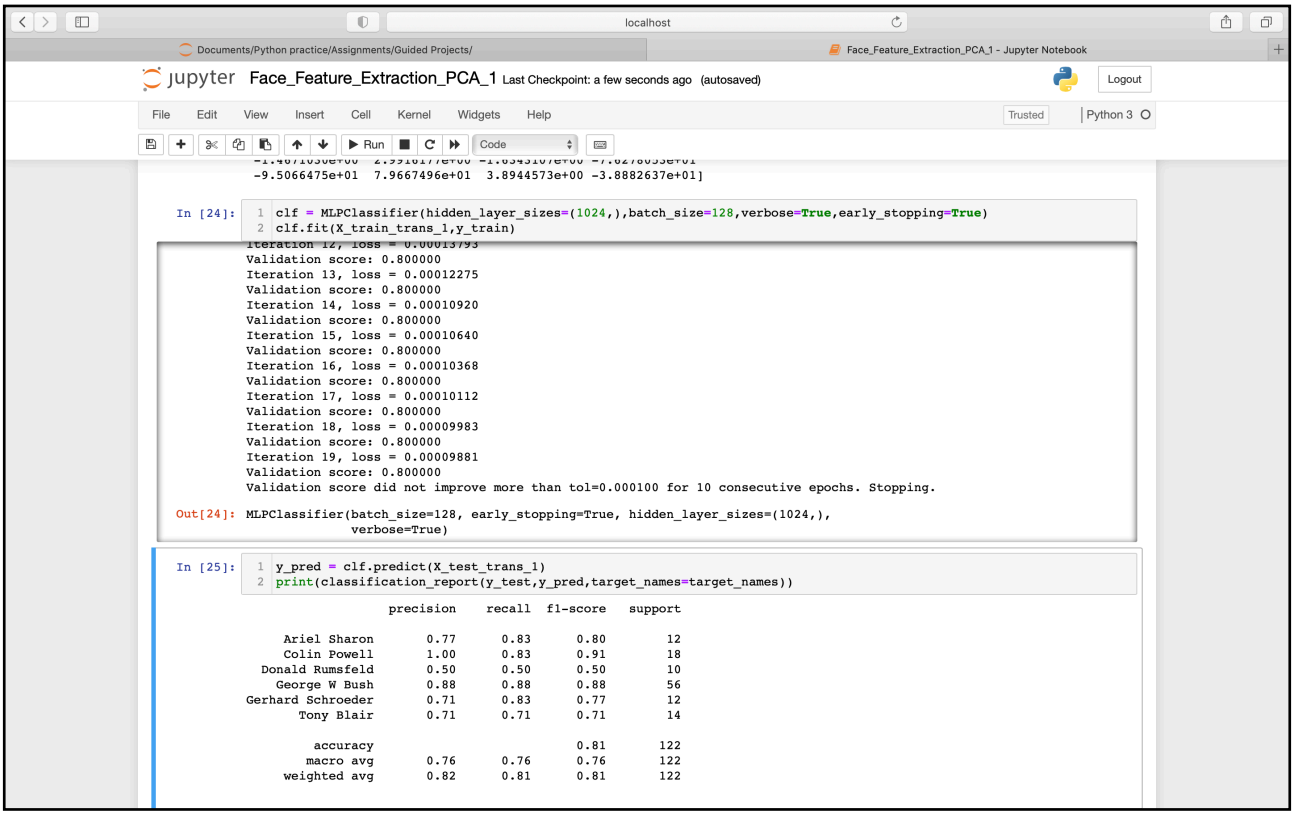
In [17]:
```
## checking the shape of transformed trained data
X_train_trans_1.shape
```

Out[17]:
```
(1095, 120)
```

In [18]:
```
print("Sample Data point after applying PCA\n", X_train_trans_1[0])
```

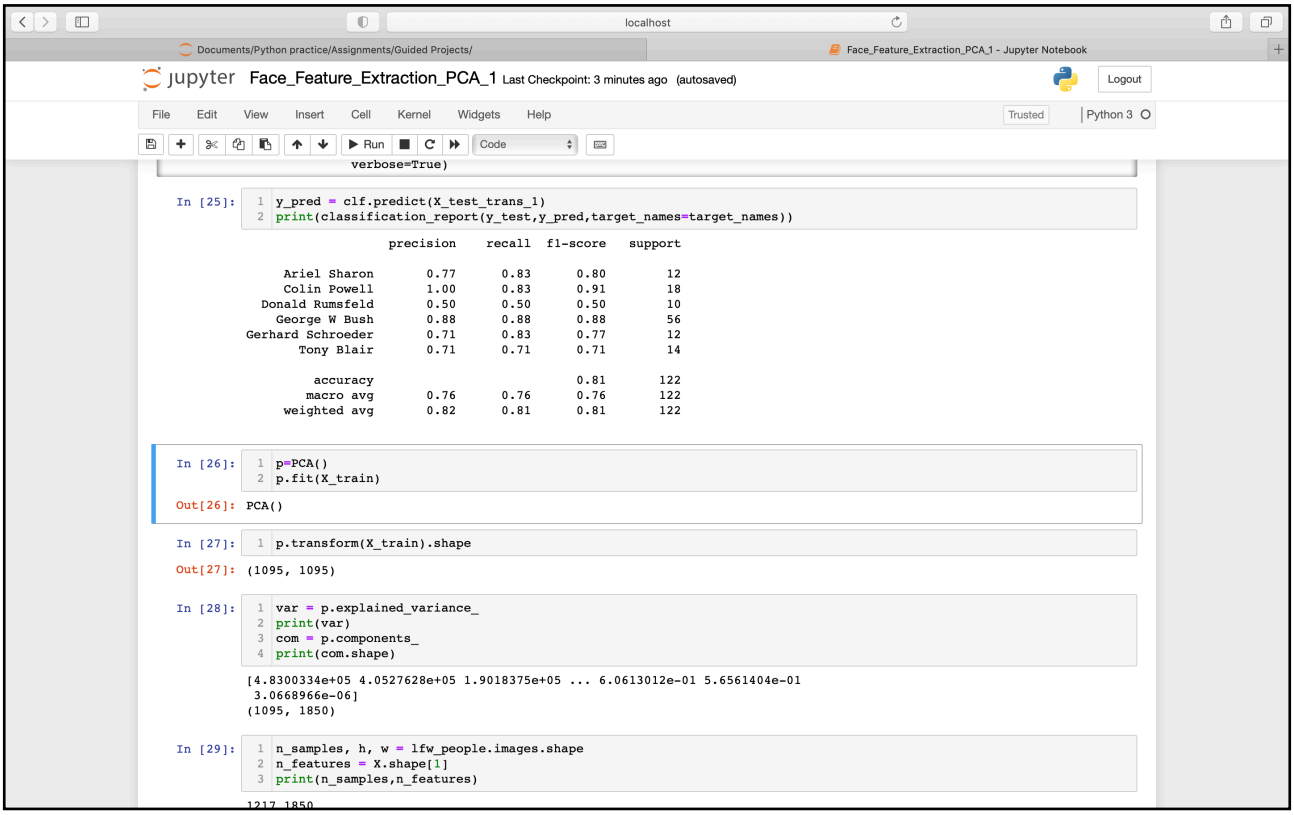
```
Sample Data point after applying PCA
 [ 1.8843967e+03  1.2173498e+02 -8.8246045e+02 -6.4566980e+02
  2.7310669e+02 -2.6214212e+02  5.4722803e+02  7.9852470e+01
 -3.6369867e+02  9.0613365e+00  2.0212723e+02  1.4928101e+02
 -2.9516705e+02 -8.8462608e+01 -8.6939026e+01 -2.5131271e+02
  1.3522586e+02 -3.6708542e+01  1.3752499e+02 -4.9527328e+01
 -2.5368770e+02  1.6897423e+01  5.2662872e+01  1.3238553e+02
 -1.7678476e+02  1.3557181e+02  1.1225069e+02 -1.7076546e+01
 -4.4958664e+01 -7.7479645e+01 -3.7796860e+01 -6.5475006e+01
 -3.9000286e+01 -9.0301659e+01  4.6496651e+01  3.3838230e+01
 -1.0851994e+02  6.7854546e+01 -1.2507789e+02  1.0922619e+02
  8.4207901e+01 -1.3380695e+02 -1.8661429e+02  2.6434549e+01
  3.2155800e+01  1.1505450e+01 -2.2860565e+00 -7.7489708e+01
  7.3646294e+01 -6.5107040e+01  1.1867351e+01 -7.8272713e+01
 -6.7270584e+00  1.8410522e+01  1.9893097e+01  4.2238930e+01
  5.3691704e+01 -9.9404938e+01  1.7036217e+02 -2.2563335e+01
```

MLP Clarification and prediction



Hidden layer size can be set to 128 or 512 to check the best f1 scores .

PCA



Considering features which are having 98 percentage contribution

```
1217 1850
```

In [30]:
```
# find the higher variance
val_sum = np.sum(p.explained_variance_)
print(val_sum)
sort_ind = np.argsort(var) # return indices of variance in ascending order
sort_ind = sort_ind[::-1] # descending order
print(sort_ind)
```

```
2584120.0
[   0    1    2 ... 1092 1093 1094]
```

In [31]:
```
## considering the features contributing to 98 percentage of total
temp_sum = 0
principal_val =[]
principal_vec =[]
i = 0
while(temp_sum<0.98*val_sum):
    principal_vec.append(com[sort_ind[i],:])
    principal_val.append(var[sort_ind[i]])
    temp_sum+=var[sort_ind[i]]
    i+=1
print("Number of components is:",i)
```

```
Number of components is: 247
```

In [32]:
```
principal_vec = np.matrix(principal_vec)
```

In [33]:
```
print(principal_vec.shape)
```

```
(247, 1850)
```

In [34]:
```
## transforming dataset
X_train_trans = np.dot(X_train,principal_vec.T)
X_test_trans = np.dot(X_test,principal_vec.T)
```

In [35]:
```
X_train_trans.shape
```

Out[35]:
```
(1095, 247)
```

Classifying transformed data


```
    principal_vec.append(com[sort_ind[i],:])
    principal_val.append(var[sort_ind[i]])
    temp_sum+=var[sort_ind[i]]
    i+=1
print("Number of components is:",i)
```

```
Number of components is: 247
```

In [32]:
```
principal_vec = np.matrix(principal_vec)
```

In [33]:
```
print(principal_vec.shape)
```

```
(247, 1850)
```

In [34]:
```
## transforming dataset
X_train_trans = np.dot(X_train,principal_vec.T)
X_test_trans = np.dot(X_test,principal_vec.T)
```

In [35]:
```
X_train_trans.shape
```

Out[35]:
```
(1095, 247)
```


In [36]:
```
clf = MLPClassifier(hidden_layer_sizes=(512,),batch_size=128,verbose=True,early_stopping=True)
clf.fit(X_train_trans,y_train)
```

```
Iteration 1, loss = 25.93009459
Validation score: 0.245455
Iteration 2, loss = 23.71320629
Validation score: 0.454545
Iteration 3, loss = 18.50820056
Validation score: 0.527273
Iteration 4, loss = 12.88955734
Validation score: 0.545455
Iteration 5, loss = 6.88299778
Validation score: 0.700000
Iteration 6, loss = 3.64305107
Validation score: 0.727273
Iteration 7, loss = 2.62400173
Validation score: 0.781818
Iteration 8, loss = 1.52689223
Validation score: 0.781818
```

Classification report

```
Iteration 17, loss = 0.00014661
Validation score: 0.790909
Iteration 18, loss = 0.00013325
Validation score: 0.790909
Iteration 19, loss = 0.00012636
Validation score: 0.790909
Iteration 20, loss = 0.00012204
Validation score: 0.790909
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

Out[36]:
```
MLPClassifier(batch_size=128, early_stopping=True, hidden_layer_sizes=(512,),
              verbose=True)
```

In [37]:
```
y_pred = clf.predict(X_test_trans)
print(classification_report(y_test,y_pred,target_names=target_names))
```

```
                    precision    recall  f1-score   support

       Ariel Sharon       0.77      0.83      0.80        12
       Colin Powell       0.75      0.67      0.71        18
    Donald Rumsfeld       0.75      0.60      0.67        10
      George W Bush       0.89      0.86      0.87        56
  Gerhard Schroeder       0.65      0.92      0.76        12
         Tony Blair       0.79      0.79      0.79        14

           accuracy                           0.80       122
          macro avg       0.77      0.78      0.76       122
       weighted avg       0.81      0.80      0.80       122
```

In [38]:
```
def plot_lfw(images, titles,h,w,rows=3,cols=3):
    plt.figure(figsize=(2*cols,2*rows))
    for i in range(rows*cols):
        plt.subplot(rows,cols,i+1)
        plt.imshow(images[i].reshape(h,w),cmap='gray')
        plt.title(titles[i])

```

In [39]:
```
# appending 247 images of size 50*37
n_components=247
mean_imgs = []
```

Plotting the 247 features

```
        plt.imshow(images[i].reshape(h,w),cmap='gray')
        plt.title(titles[i])

```

In [39]:
```
# appending 247 images of size 50*37
n_components=247
mean_imgs = []
for i in range(n_components):
    v = principal_vec[i,:]
    img = v.reshape((h,w))
    mean_imgs.append(img)
mean_imgs = np.array(mean_imgs)
print(mean_imgs.shape)
```

```
(247, 50, 37)
```

In [40]:
```
pca_tiles = [f"eigenvector-{i}" for i in range(n_components)]
#print(pca_tiles)
plot_lfw(mean_imgs, pca_tiles, h,w)
```

In [ ]: