CAPSTONE PROJECT REPORT

Name - Paleti Samuel Yashaswi
Course - Machine learning and AI
Duration - 10 months
Question - 1

Face Feature Extraction

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.

Data Set Used - dataset which is used is imported from the scikit-learn library

Method used is Principal Component Analysis

Importing datasets

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import sklearn
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_lfw_people
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV



people=fetch_lfw_people(min_faces_per_person=70,resize=0.4)

n_samples,h,w=people.images.shape

X=people.data
print(X.shape)
n_features=X.shape[1]

y=people.target

target_names=people.target_names
n_classes=target_names.shape[0]


print("total dataset size:")

print("no of samples: %d" % n_samples)
print("no of features: %d" % n_features)
print("no of classes: %d" % n_classes)
```

Spliting the data(train, test) , performing PCA and training data using SVC

```
X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.3,random_state=42)SSS
n_components = 150

print("Extracting the top %d eigenfaces from %d faces"
      % (n_components, X_train.shape[0]))
pca = PCA(n_components=n_components, svd_solver='randomized',whiten=True).fit(X_train)

eigenfaces = pca.components_.reshape((n_components, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

```
Extracting the top 150 eigenfaces from 851 faces
Projecting the input data on the eigenfaces orthonormal basis
```

```
print("Fitting the classifier to the training set")
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid)
clf = clf.fit(X_train_pca, y_train)
print("Best estimator found by grid search:")
print(clf.best_estimator_)
```

Accuracy of the face which is predicted

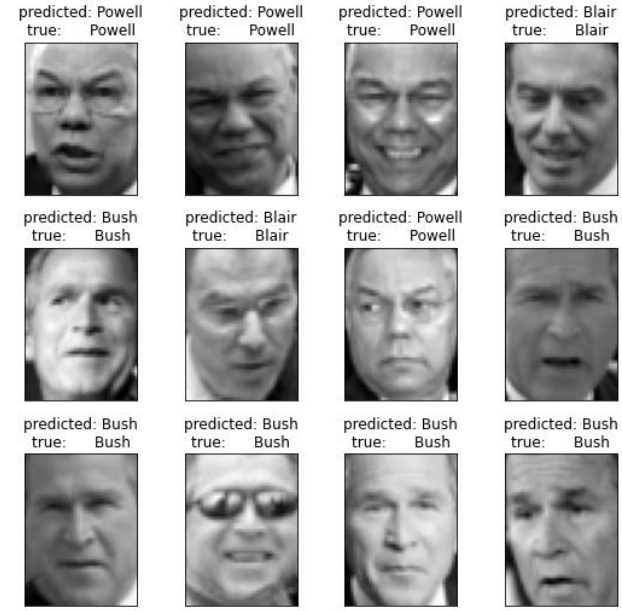
```
print("Predicting people's names on the test set")
y_pred = clf.predict(X_test_pca)

print(classification_report(y_test, y_pred, target_names=target_names))
```

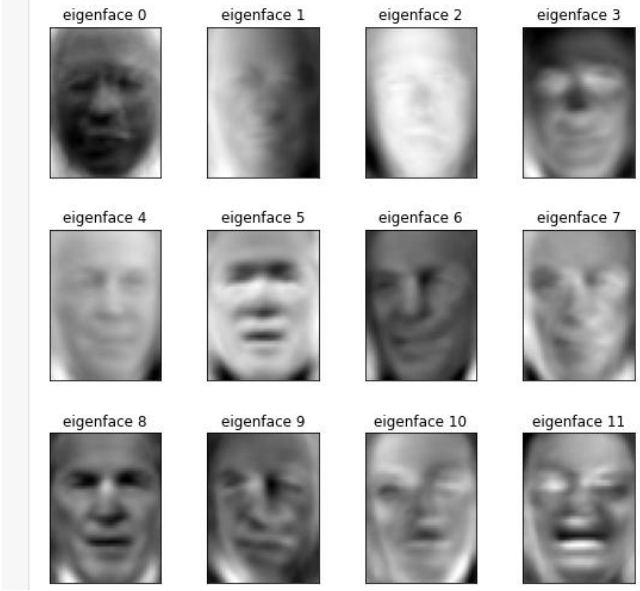
```
Predicting people's names on the test set
                   precision    recall  f1-score   support

     Ariel Sharon       0.92      0.60      0.73        20
     Colin Powell       0.91      0.79      0.84        75
  Donald Rumsfeld       0.90      0.69      0.78        39
    George W Bush       0.78      0.98      0.87       163
Gerhard Schroeder 1.00 0.74 0.85 27
       Tony Blair       0.97      0.74      0.84        42

         accuracy                           0.84       366
        macro avg       0.91      0.76      0.82       366
     weighted avg       0.86      0.84      0.84       366
```

Plotting the faces after testing and plotting eigen faces

```
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())

def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(' ', 1)[-1]
    true_name = target_names[y_test[i]].rsplit(' ', 1)[-1]
    return 'predicted: %s\ntrue:      %s' % (pred_name, true_name)


prediction_titles = [title(y_pred, y_test, target_names, i)
                     for i in range(y_pred.shape[0])]

plot_gallery(X_test, prediction_titles, h, w)

eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)

plt.show()
```

Output - After traning the data, It can recognise the faces with a certain accuracy
And the eigen faces which are used for recognition are displayed




eigenface 0
eigenface 1
eigenface 2
eigenface 3
eigenface 4
eigenface 5
eigenface 6
eigenface 7
eigenface 8
eigenface 9
eigenface 10
eigenface 11