PRML PA 3

ANUSHKA DADHICH
B22CS097

QUESTION - 1
PERCEPTRON

Task 0:

The following functions are created with their descriptions -

1. generate_data(num_samples, weights) - generates the random data points as per the number of num_samples using random.randint(0, 10) i.e integers between 0 and 10. Then using the cost function of the weights array assigns the labels to the data points and returns the data in the specified format as given.
2. save_data(data, filename) - returns the ‘data.txt’ file by writing the data generated in the prev function in the filename provided.

Outputs -

```
Synthetic dataset generated and saved to data.txt
```

```
3000
8 10 5 3 1
4 1 9 2 0
8 7 5 6 0
6 0 3 4 0
7 3 9 1 0
5 10 2 9 0
9 4 4 3 1
8 0 8 0 0
6 6 0 2 1
1 1 5 0 0
3 8 5 7 0
5 2 2 4 0
10 2 10 3 0
3 0 8 0 0
9 8 6 9 0
9 8 6 10 0
1 4 9 2 0
1 8 3 9 0
2 8 1 10 0
9 1 1 1 1
0 6 6 9 0
2 8 9 0 0
4 3 9 4 0
2 0 5 7 0
6 3 2 2 1
10 7 1 6 1
```

Task 1:

Creation of the following functions and their corresponding descriptions -

1. normalize_data(data) - Uses Z-score normalization to normalize the input data.
2. perceptron_train(data, learning_rate=0.1, max_epochs=1000) - the PLA code that firstly splits the input data into X (features) and y (target labels) and then for the total number of epochs (iterations) it keeps on updating the weights by weights += learning_rate * error * X_with_bias.

Output -
Just functions created (train.py)
Would return list which stores weights of the PLA on train.txt

```
PS C:\Users\Anushka\OneDrive\Desktop\PRML\Lab3> python train.py
The weights on trained data are -  [-56428.17013026222, 29276.59308566843, 43309.77267552037, -49998.44036997994, -51312.52
83953114]
```

Task 2:

Creation of the following functions and their corresponding descriptions -

1. load_data(filename) - this converts the file data into numpy array and returns it
2. predict(X,weights) - this returns the predicted labels list as per the input weights of the PLA

Output -
Would return the list of predicted labels as per data from test.txt

```
PS C:\Users\Anushka\OneDrive\Desktop\PRML\Lab3> python test.py
Predicted Labels =  [0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
```

Task 3:

Splitting the data.txt into test and train then storing the last column of labels of test data as y_true, then using perceptron_train function to get weights of the train.txt each time with given %age of split and then finding the predicted labels and using accuracy function to compute accuracy.

Taking the split number as training dataset and the 30% of the ‘data.txt’ as the test.txt, the outputs are -

```
Accuracy for split 0.2 :  0.9520833333333333
Accuracy for split 0.5 :  0.9573333333333334
Accuracy for split 0.7 :  0.9388888888888889
```

%age of data.txt as train.txt	Accuracy
20	0.952
50	0.957
70	0.938

This is because-

1. Randomness in Data Splitting - In each split random data is allocated.
2. As the number of data points increases, the total iterations of the data i.e the rows in the X array increases and thus the weights get updated that many extra times and this might possibly increase the accuracy so that to learn the general pattern, but as these increases, there might be chances of extra iterations, i.e fitting more data points and leading to a lower accuracy as the total data points keep on increasing.
3. Smaller training sets may result in less representative models

QUESTION 2
EIGENFACES USING PCA

Task 1:
Data Preprocessing

1. Loaded the dataset using the sklearn's
   lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
   And then separation of features X, and target y, printing the n_samples, n_classes, n_features.
2. Splitting the dataset into 80:20 using the train_test_split() and then scaled the data using the default StandardScaler().

Output -

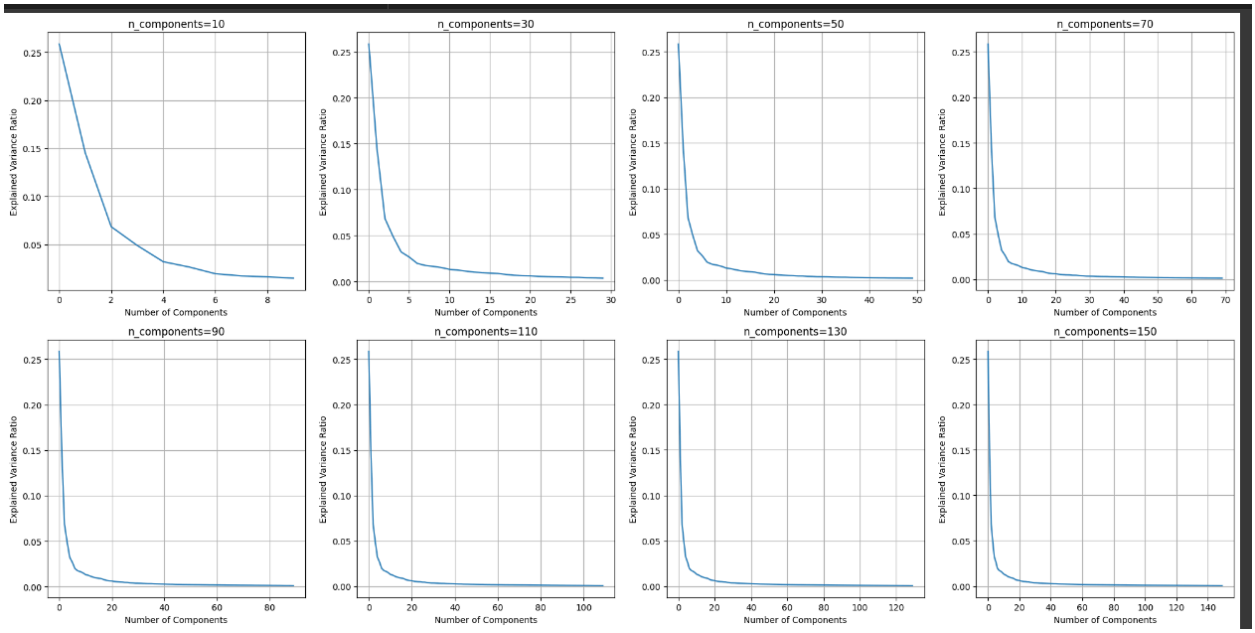
```
Total dataset size:
n_samples: 1288
n_features: 1850
n_classes: 7
```

Task 2:

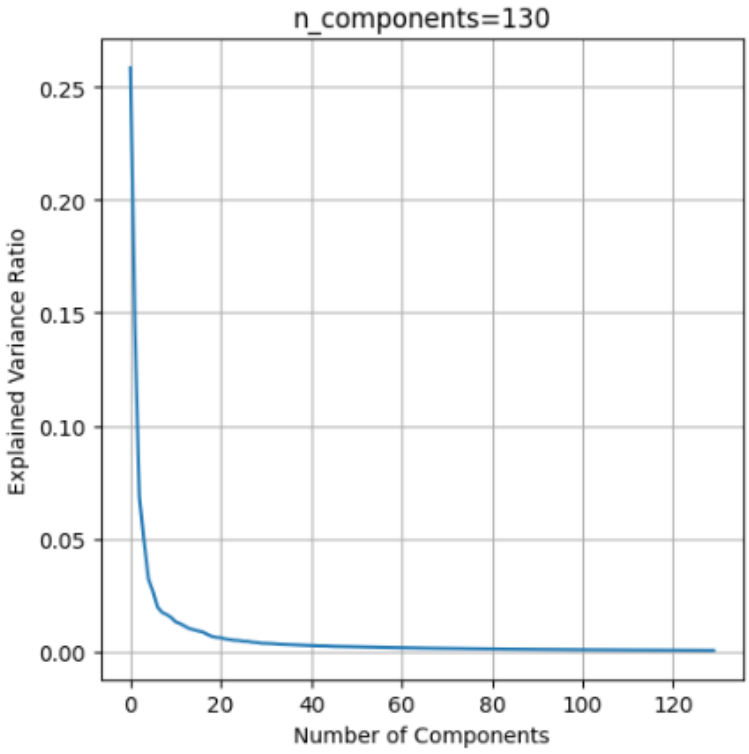
Eigenfaces Implementation

1. Why and how I choose 130 as the number of components:
   Plotted the graph of Explained Variance Ratio vs Number of Components for a range of values of n_components.
   Observed that the line becomes constant after around n_components=130 plot, so any higher value would not add much value to the change in variance ratio, but unnecessarily increase computation. So I took a value of 130 (rounded off approx visually)

   Plotted graphs -



   Graph of n = 130:



2. Implementing PCA with n_components=130: Used the predefined functions:
   pca = PCA(n_components=n_components, whiten=True, random_state=42)
   pca.fit(X_train)

Outputs -

```
Extracted the top 130 eigenfaces from 1030 faces
```

Task 3:
Model Training

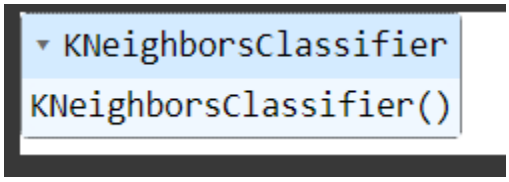
Used knn classifier to train the model using sklearn
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_pca, y_train)

*Comparion:

I compared knn and svm classifiers (with n_components = 130) and the accuracy with svm is better (but in the question knn is said so I have proceeded further with knn to avoid confusions)

```
Accuracy with KNN: 0.7441860465116279
Accuracy with SVM: 0.7906976744186046
SVM classifier is better.
```

Outputs -



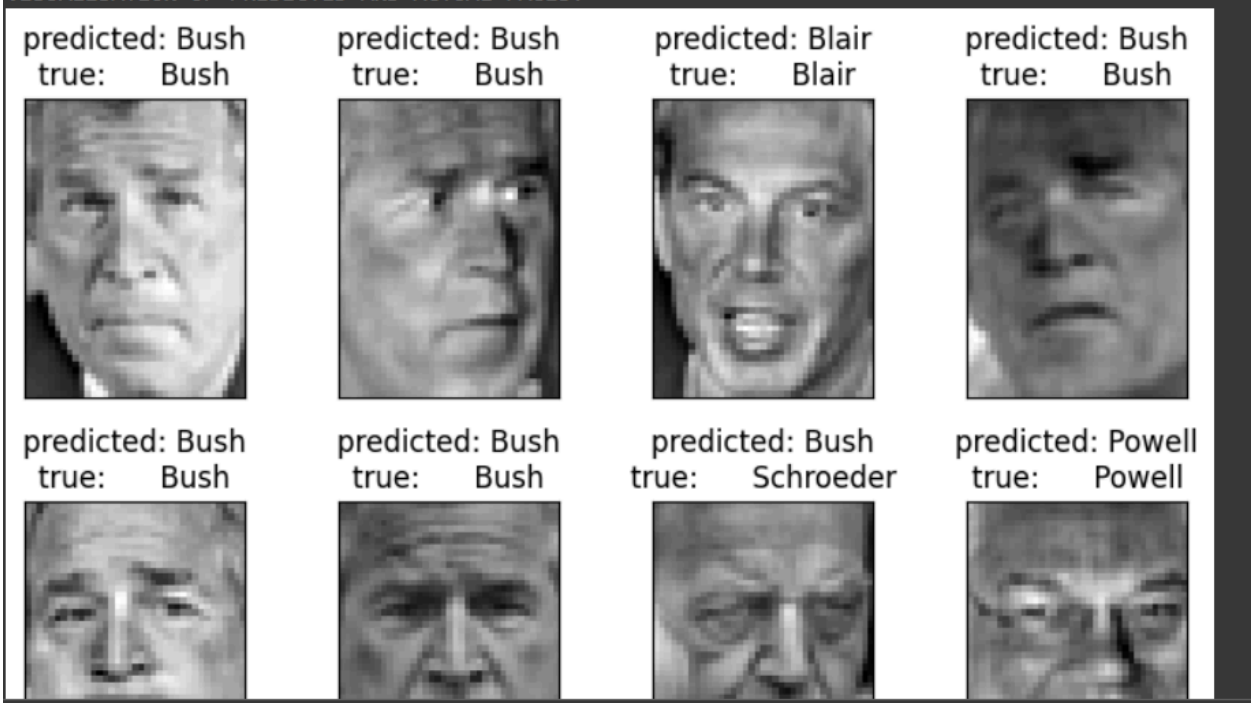
Task 4:
Model Evaluation

1. Using y_pred = knn_classifier.predict(X_test_pca), finding the predictions on the test data.
2. Using the accuracy = accuracy_score(y_test, y_pred) calculating the accuracy of the model and also other parameters by classification_report(y_test, y_pred, target_names=target_names).
3. Created helper functions to visualize the eigenfaces as given in the sample code.

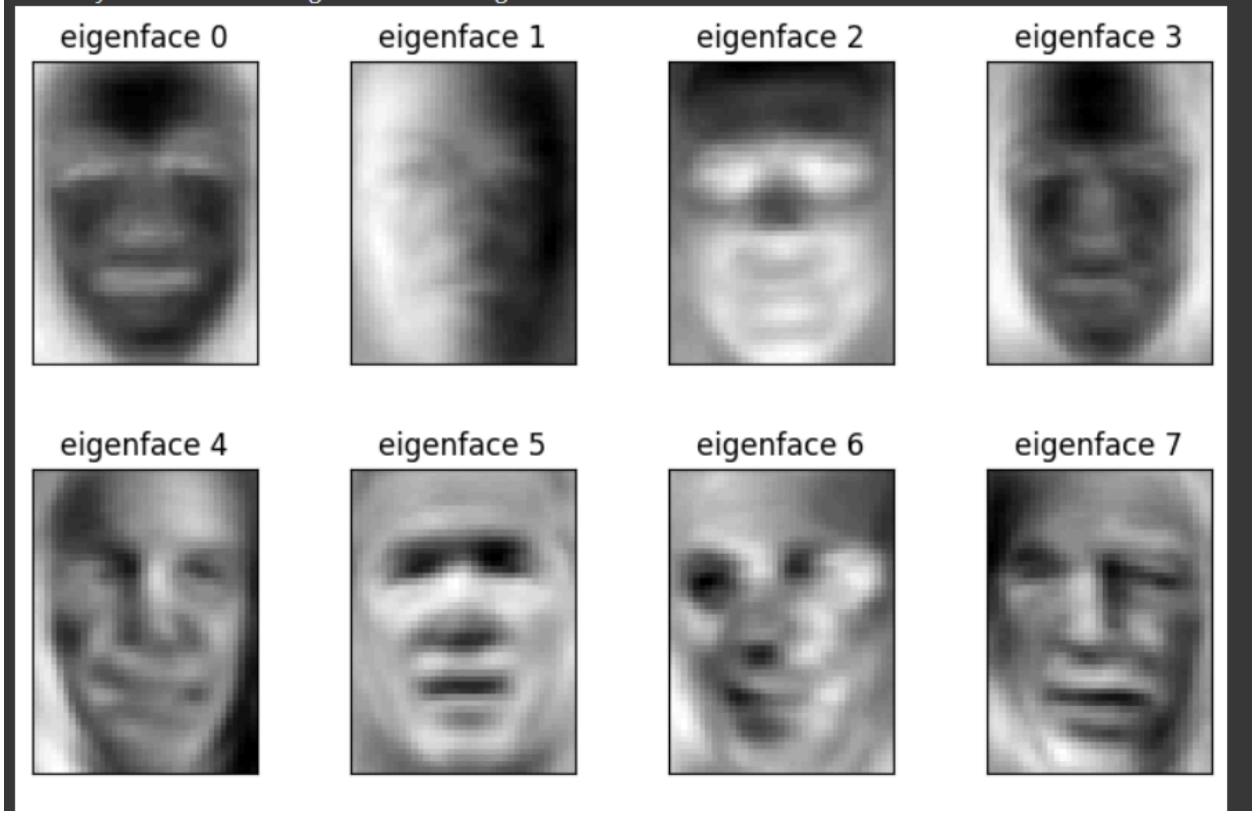
Outputs -

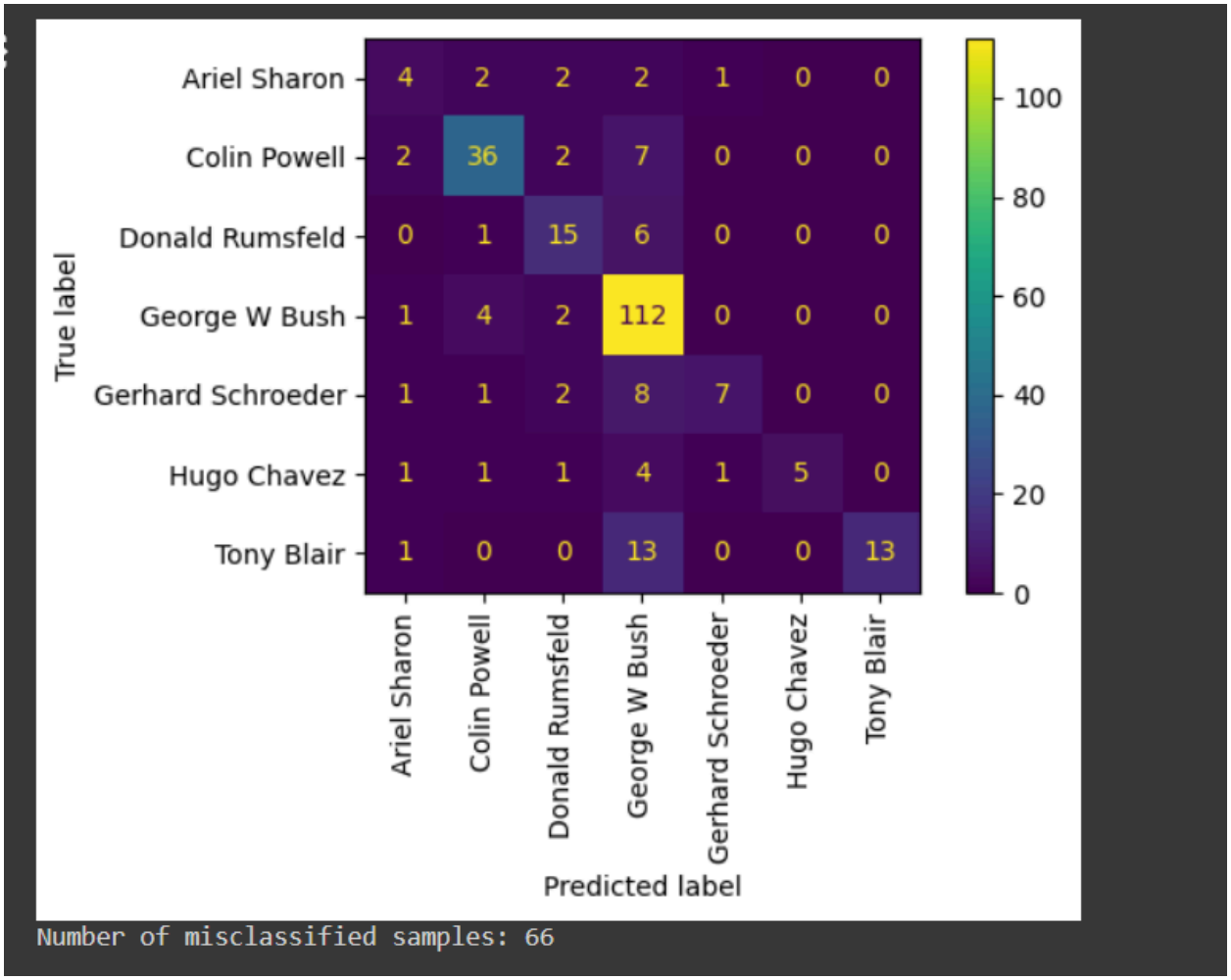
```
Predicting people's names on the test set
done in 0.038s
Accuracy: 0.7441860465116279
                   precision    recall  f1-score   support

     Ariel Sharon       0.40      0.36      0.38        11
     Colin Powell       0.80      0.77      0.78        47
  Donald Rumsfeld       0.62      0.68      0.65        22
    George W Bush       0.74      0.94      0.83       119
Gerhard Schroeder       0.78      0.37      0.50        19
      Hugo Chavez       1.00      0.38      0.56        13
       Tony Blair       1.00      0.48      0.65        27

         accuracy                           0.74       258
        macro avg       0.76      0.57      0.62       258
     weighted avg       0.77      0.74      0.73       258
```

VISUALISATION OF PREDICTED AND ACTUAL FACES:



Gallery of the most significative eigenfaces





Observations:

1. Looking at the confusion matrix, the most mis-classified sample is the Gerhard Schroeder, which is not able to classify correctly
2. To make this better, more samples of this class must be there in the train dataset so that it is easier to generalize this class.
3. A different classifier (such as SVM as mentioned) can be used for better performance.

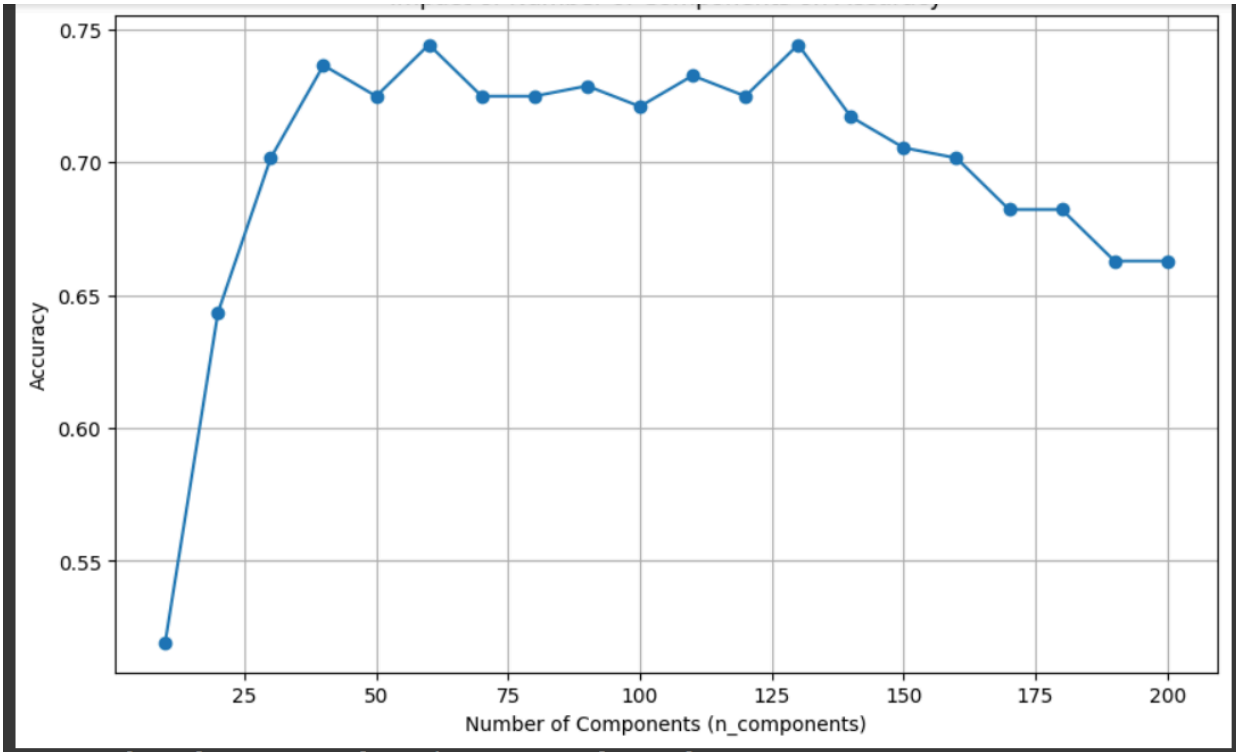
Task 5;

Experiment with different values of n_components:

Loop iteration over all values of n_components in a range and then stored the corresponding accuracies and then using matplotlib plotted the graph of accuracy vs n_componets.

It also shows that the value of 130 chosen was corresponding to a higher accuracy.

Outputs -




```
Top two values of n_components for maximum accuracy: [130, 60]
Corresponding maximum accuracies: [0.7441860465116279, 0.7441860465116279]
```