PRML LAB 5,6 REPORT

PROBLEM 1:

TASK-0:

Synthetic dataset generation:
We created a synthetic dataset with four dimensions (x1, x2, x3, x4) utilizing the NumPy module. The dataset consists of 1000 samples, each including four features. The features were created as random numbers(integers) ranging from -99 to 100(excluded), providing a wide variety of values to depict the dataset.

Linear function defined:
To establish a relationship between the features and the target variable, we initialized a linear function:

$$f(x) = w0x0 + w1x1 + w2x2 + w3x3 + w4x4$$

Where w1, w2, w3, w4, w0 are randomly chosen weights. These weights were assigned to mimic the influence of each feature on the target variable.

Labeling data points:
Every data point was assessed using the linear function f(x). If the output of function f(x) was non-negative, the data point was classified as a positive example; otherwise, it was classified as a negative example. We used a binary classification method to assign each data point to a category based on its correlation with the linear function.

Link to the main code:
main.ipynb

TASK-1:

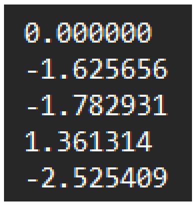
Data Normalisation:
We normalized the input data using min-max normalization to guarantee accurate training and convergence of the perceptron learning algorithm. This method rescales the features to a range of [0, 1], maintaining the relative relationships between distinct features and avoiding numerical instabilities caused by variable feature magnitudes.

Bias term:
A bias term (x0) was added to the feature matrix to adjust for any deviation in the decision boundary. Setting the bias term to a fixed value of 1 allows the perceptron to learn a suitable bias weight during training, enhancing its ability to accurately represent the data.

Perceptron learning algorithm:
We used the perceptron learning approach to train the model using the normalized dataset. The algorithm continuously modifies the weights to reduce the classification error. The perceptron calculates the predicted output (y_pred) in each iteration by utilizing the current weights and then compares it with the true label (y). The weights are adjusted by calculating the difference between the predicted and true labels, then multiplying this difference by the learning rate (lr) and the input characteristics (X).

Training process:
The training method includes cycling over the dataset for a set number of epochs. The perceptron adjusts its weights using the whole training set at each epoch. Upon completion of training, the final weights are saved, which signify the acquired parameters of the perceptron model.

weights:



```
0.000000
 -1.625656
 -1.782931
1.361314
 -2.525409
```

TASK-2:

In this task we generate labels for the testing data incorporating the weights that we obtain by training the model in task-1. And thus we can calculate the accuracy for our testing data

TASK-3:
This task expects us to report accuracy for our model taking into account 20%, 50% and 70% of our synthetic data as training data and comparing them:

Percentage of data	Accuracy
20	0.9809523809523809
50	1.0
70	0.9809523809523809

PROBLEM 2:

TASK-1:

In this task we load the LFW dataset provided by sklearn library. In the function we set a threshold of minimum 70 images per target and resize those images to 40% of its original size.

Then we extract the size of our dataset and height and width of each image, and other information about our dataset such as the total features and total unique target variables.

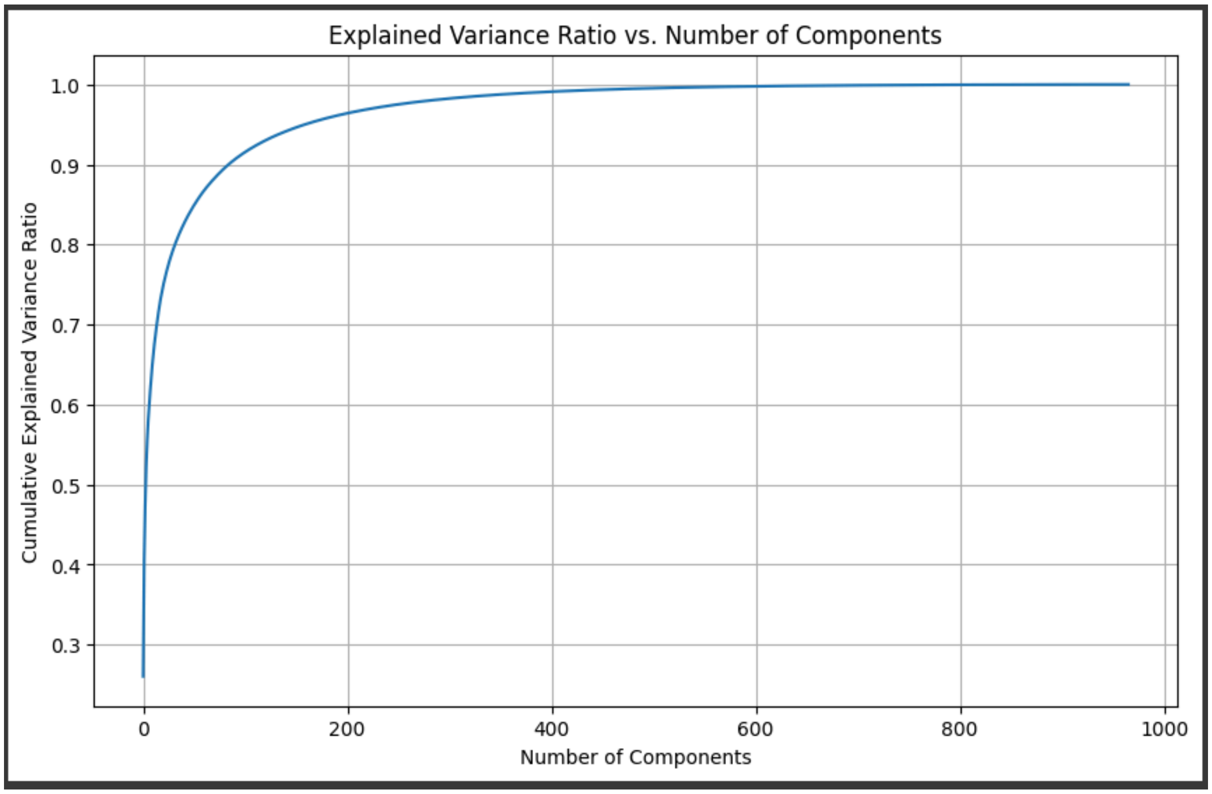
After extracting information about our dataset we display one the resized images using matplotlib.

Further using test_train_split library function we divide out dataset in test (25%) and train (75%).

TASK-2:

In this task we are supposed to implement eigenfaces using PCA using an appropriate value for the number of components to retain for dimensionality reduction.

We do this by plotting a graph between the number of features (x-axis) and the measured variance (y-axis).



The plot illustrates the correlation between the quantity of primary components and the cumulative explained variance ratio. The graph shows that:

The curve initially has a sharp incline, suggesting that incorporating additional components accounts for a substantial amount of the variability in the data.
No changes needed. As the quantity of components grows, the rate of explained variance rise diminishes progressively.

The curve plateaus, suggesting that further components have diminishing impact on the total explained variance.

The point where the curve begins to level off indicates the ideal number of components required to capture the majority of the variance in the data, approximately 95%.

Therefore, we may infer that we should maintain our number of components to be greater than or equal to 100 but less than 200. An reasonable value for us would be 150 based on the graph, as the covariance ratio is close to 95%.

We utilise Eigenfaces through Principal Component Analysis (PCA) for reducing dimensionality. The process begins by initialising the number of principal components (pca_comp) to 150 and displaying a message confirming the extraction of the top 150 eigenfaces from the training faces.

PCA is then implemented with the required number of components (n_components=pca_comp) utilising the PCA class from scikit-learn. The PCA constructor is supplied the parameters svd_solver="auto" and whiten=True. The parameter svd_solver="auto" automatically selects the most efficient decomposition technique.

The PCA model is applied to the training data (X_train) containing facial photos. This process calculates the primary components and reduces dimensionality.

The eigenfaces are derived from the principal components after applying the PCA model. Eigenfaces are the major components that directly depict the fundamental patterns found in the face images. The eigenfaces are resized to fit the dimensions of the original images.

The transformed representations of the training and testing data are produced using the fitted PCA model. The converted representations, X_train_pca and X_test_pca, comprise low-dimensional versions of the original face photos that capture the most important differences in the data by lowering dimensionality.

TASK-3:

Task-3 involved training a classifier for Eigenfaces by utilising Principal Component Analysis (PCA) for dimensionality reduction and the K-Nearest Neighbours (KNN) algorithm for classification.

K-Nearest Neighbours (KNN) is selected for face identification with Eigenfaces because to its simplicity, intuitiveness, and non-parametric nature. It is well-suited for high-dimensional data fields, making it appropriate for tasks such as face recognition. KNN's local decision boundary method allows it to recognise intricate patterns, and its instance-based learning does not need a specific training phase, making it computationally economical for small to medium-sized datasets.

We utilised the KNeighborsClassifier class from scikit-learn to create the KNN classifier with a predefined number of neighbours at 5.

The classifier was trained using the altered training data produced by performing PCA (X_train_pca) and their corresponding target labels (y_train).

TASK-4:

Using the adjusted testing data (X_test_pca), we evaluated the trained model's performance in this job. The scikit-learn accuracy_score function was utilised to determine the classifier's accuracy. Additionally, we looked into how the classification performance changed when the number of main components (n_components) was altered. To further understand how the selection of principal components affects the KNN classifier's accuracy for Eigenfaces, we repeatedly tested the model with varying values and retrained it for each test.

Examining Model Errors:

Testing photos where these errors occurred can help us understand why a model made erroneous predictions. Look for recurring themes in these pictures, like:

1. Pose variations: The model may have difficulties when dealing with faces in non-frontal orientations, such as profile views, because of variances in the way features are projected.

2. Changes in illumination: Photos with notable lighting anomalies, like deep shadows, might distort facial characteristics and result in incorrect classification.

3. Occlusions occur when face features are partially covered by spectacles or hair, making identification difficult.

Enhancing the Model:

To improve the model's performance, we can implement the following strategies:

1. Data Augmentation involves generating synthetic versions of original training photos through processes like rotation, cropping, or modifying brightness. This can enhance the model's capacity to manage posture, lighting, and occlusion fluctuations.

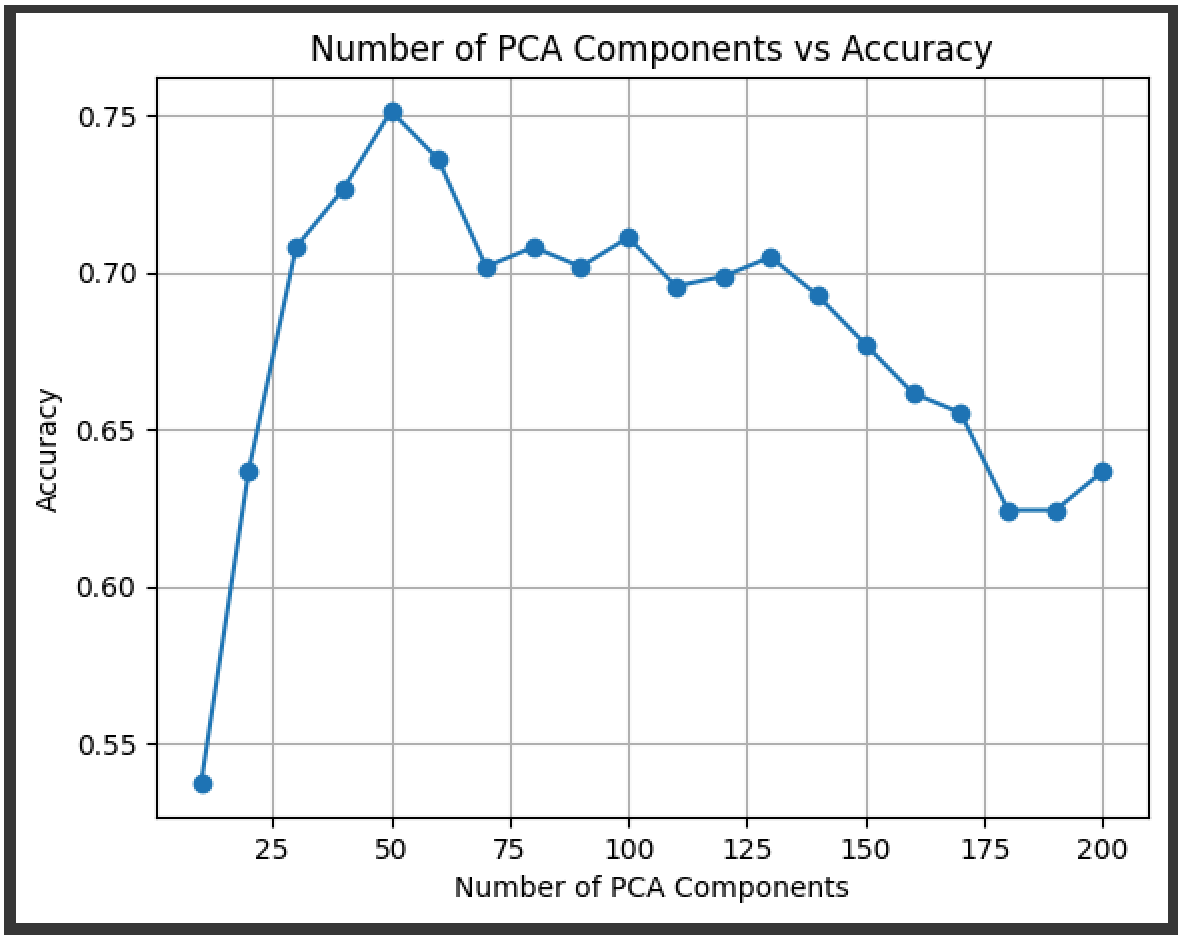
2. Feature Engineering: Investigate supplementary features like Gabor filters to capture specific facial attributes such as eyes, nose, and mouth, in addition to Eigenfaces.

3. Hyperparameter Tuning: Test various values for hyperparameters in both PCA and KNN, including the number of components or neighbors, to enhance the model's performance.

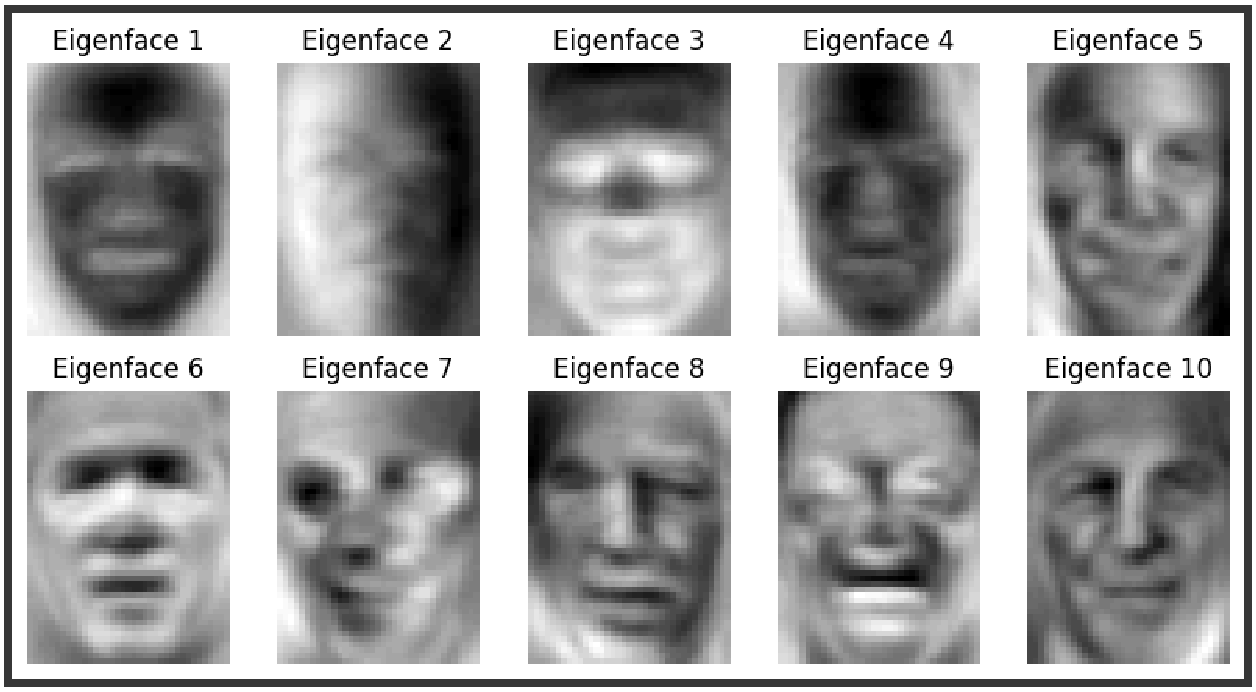
4. When exploring classifiers, consider using other models such as SVMs or deep learning models instead of KNN, depending on the dataset's complexity and your performance objectives.

TASK-5:

In this task we visualize the data experimenting with different values of n_components computing their confusion matrix and their accuracies.

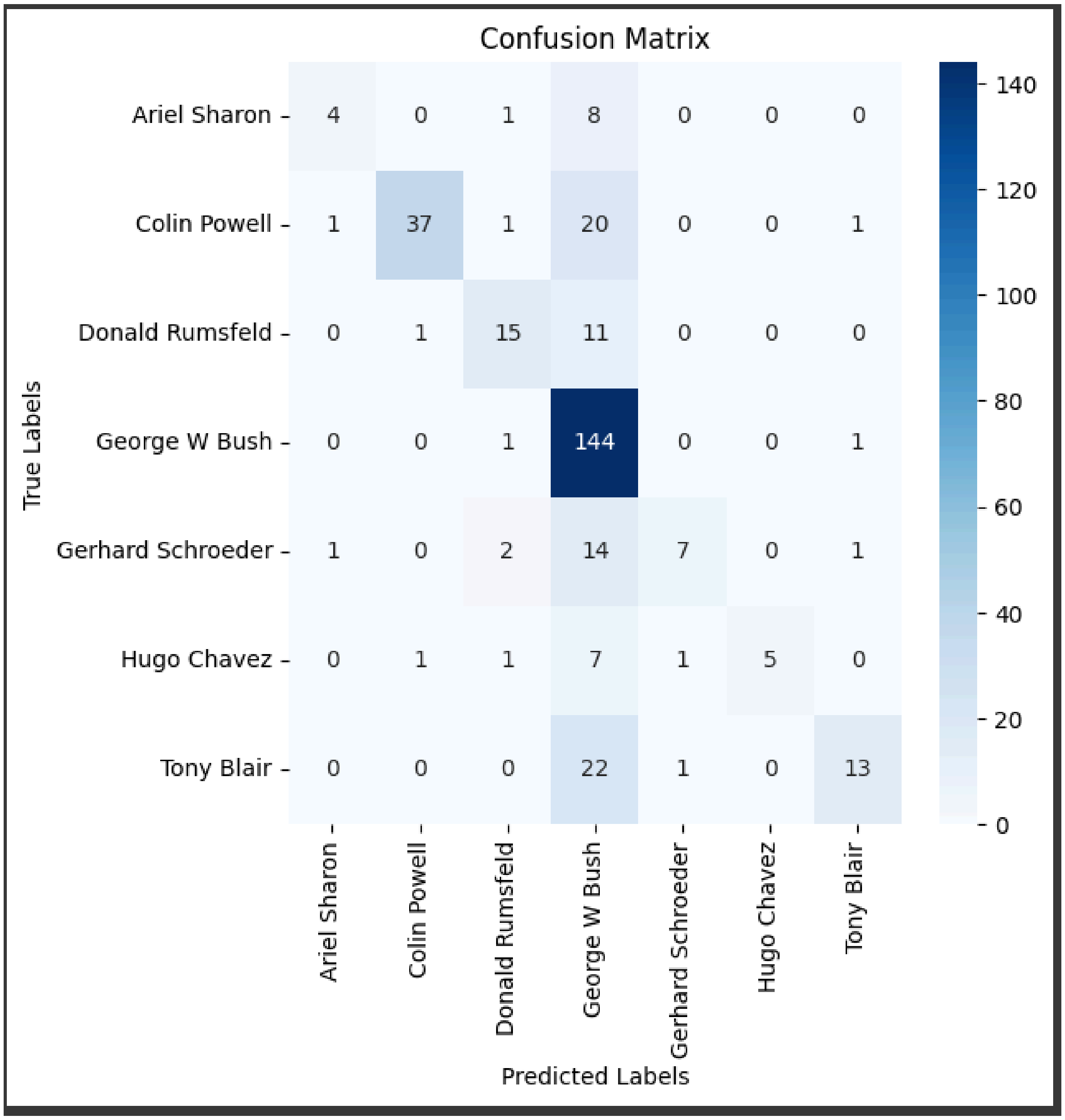


We also display the top 10 eigen faces that we have computed:



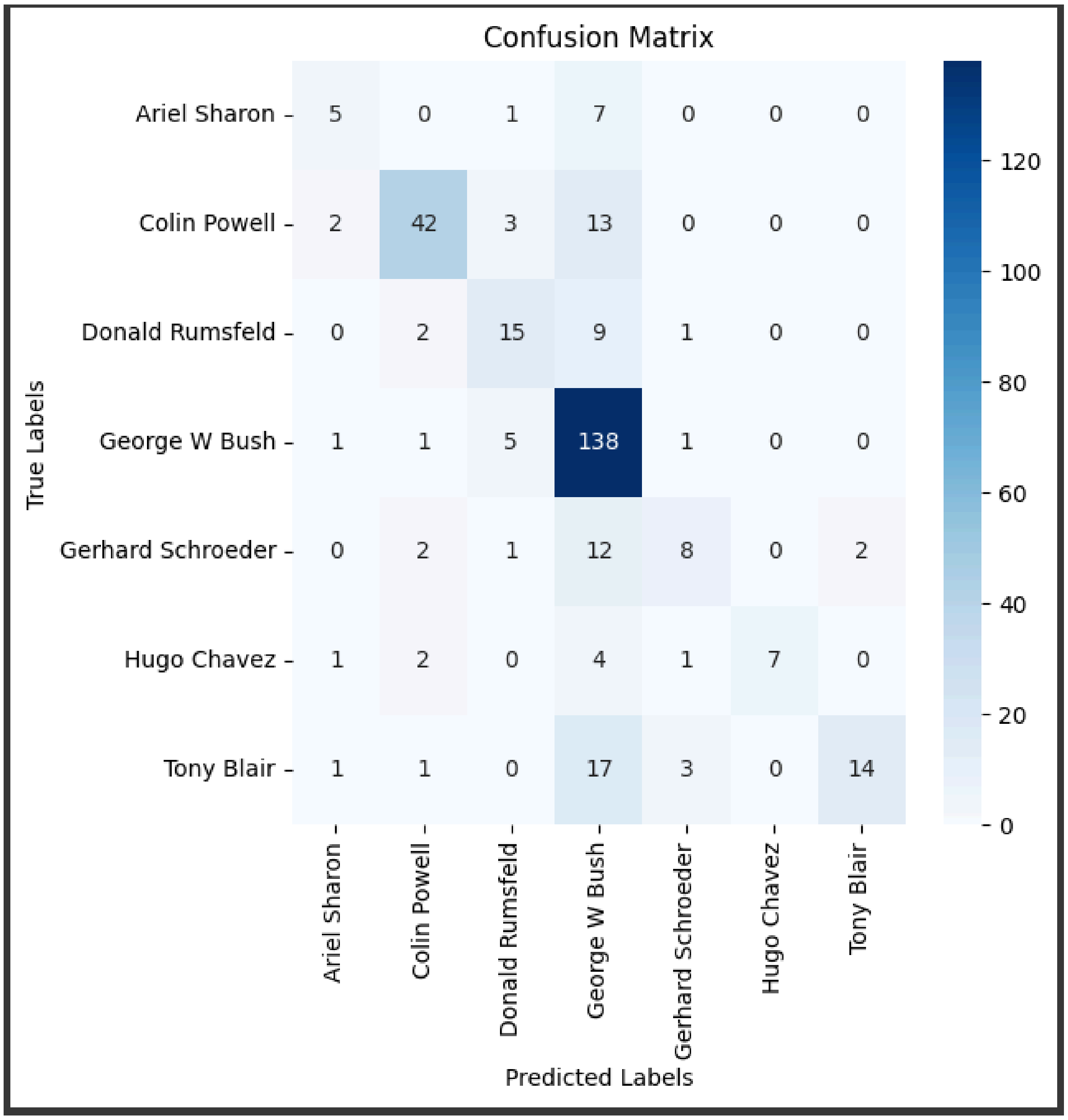
Link to the code:
eigenfaces.ipynb

Matrix for n_components = 150



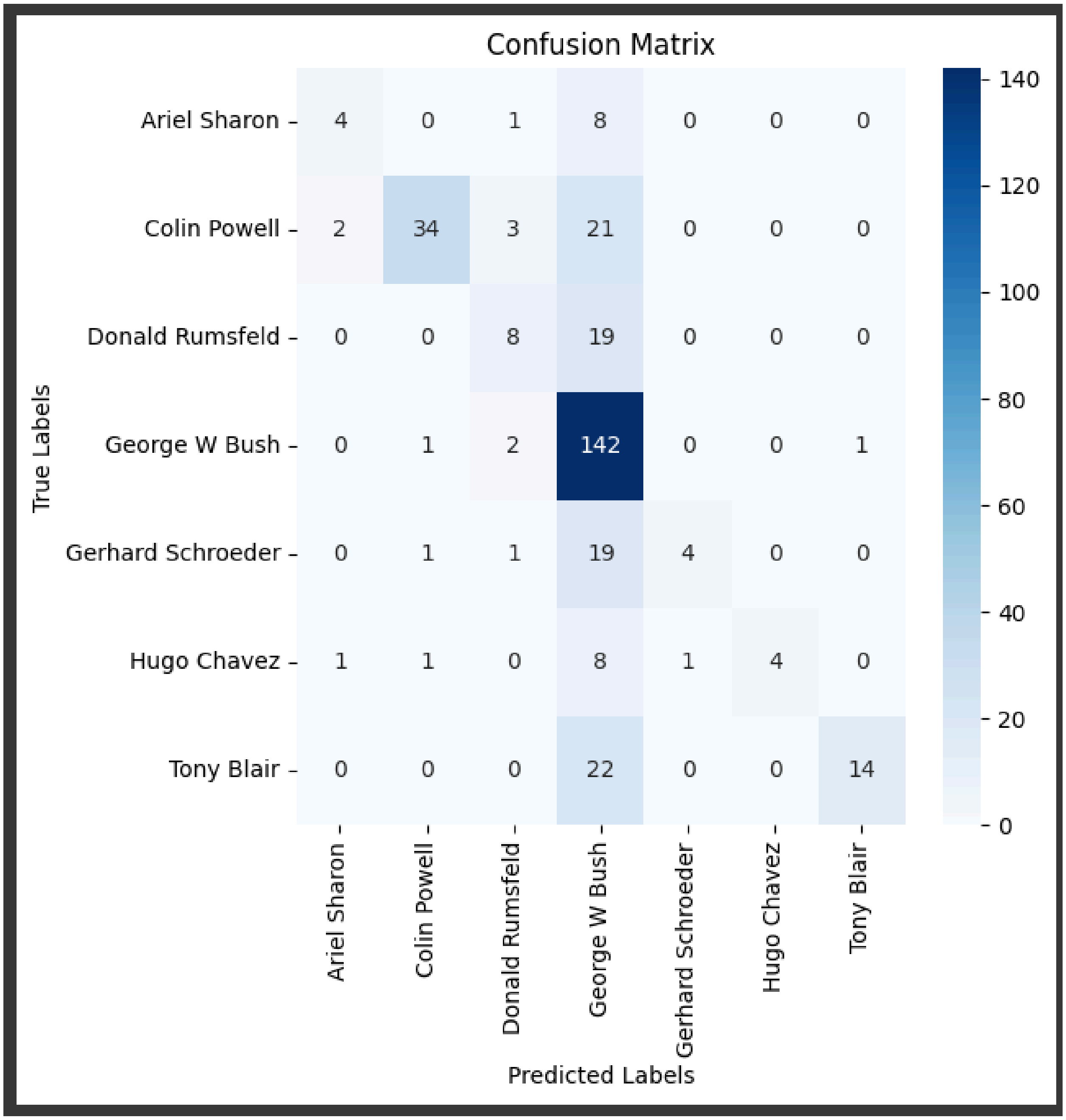
	Ariel Sharon	Colin Powell	Donald Rumsfeld	George W Bush	Gerhard Schroeder	Hugo Chavez	Tony Blair
Ariel Sharon	4	0	1	8	0	0	0
Colin Powell	1	37	1	20	0	0	1
Donald Rumsfeld	0	1	15	11	0	0	0
George W Bush	0	0	1	144	0	0	1
Gerhard Schroeder	1	0	2	14	7	0	1
Hugo Chavez	0	1	1	7	1	5	0
Tony Blair	0	0	0	22	1	0	13

N_components = 100



True Labels \ Predicted Labels	Ariel Sharon	Colin Powell	Donald Rumsfeld	George W Bush	Gerhard Schroeder	Hugo Chavez	Tony Blair
Ariel Sharon	5	0	1	7	0	0	0
Colin Powell	2	42	3	13	0	0	0
Donald Rumsfeld	0	2	15	9	1	0	0
George W Bush	1	1	5	138	1	0	0
Gerhard Schroeder	0	2	1	12	8	0	2
Hugo Chavez	1	2	0	4	1	7	0
Tony Blair	1	1	0	17	3	0	14

N_components = 200



True \ Predicted	Ariel Sharon	Colin Powell	Donald Rumsfeld	George W Bush	Gerhard Schroeder	Hugo Chavez	Tony Blair
Ariel Sharon	4	0	1	8	0	0	0
Colin Powell	2	34	3	21	0	0	0
Donald Rumsfeld	0	0	8	19	0	0	0
George W Bush	0	1	2	142	0	0	1
Gerhard Schroeder	0	1	1	19	4	0	0
Hugo Chavez	1	1	0	8	1	4	0
Tony Blair	0	0	0	22	0	0	14