Assignment 1 - Face Recognition

Nour Hesham Shaheen - 7150

Eyad Salama - 7128

1. **Download Dataset and Understand Format**
2. **Generate the Data Matrix & Label Vector**

   Using the **OpenCV** library, we read each image and reshape it into a 10304 (92x112) vector. We add corresponding labels into a “y” array.

   At the end we have a data matrix of shape (400, 10304) and a labels array of shape (400,).
3. **Split Dataset into Training & Testing sets**

To ensure a 50-50 split, and that all classes are balanced in the training and testing set, we divide the data matrix D into two matrices, **d_train** that contains all odd instances and **d_test** that contains all even instances, so that in both the training and testing set, we have 5 instances of each face.

Two corresponding labels vectors **y_train** and **y_test** is generated in the same manner.

Now, **d_train** and **d_test** have the shape (200, 10304) and **y_train** and **y_test** have the shape (200,).

4. PCA1

1. **Compute mean:** We first compute the mean (per column) of each feature and generate a matrix of dimensions (1, 10304) → a mean for each column.
2. **Center the data:** We generate a Z matrix containing centered data of shape (200, 10304)**.**
3. **Compute covariance matrix**: using the NumPy library.
4. **Compute eigenvalues and eigenvectors**: using the NumPy library. Then, we reverse sort eigenvalues and corresponding eigenvectors.
5. For each alpha, we calculate the number of eigenvalues that make the explained variance almost equal to alpha. Then we slice the eigenvector matrix to produce a **projection matrix.**
6. For each projection matrix, we produce a new projected data matrix.
7. We run a **KNN classifier** with KNN=1 for each alpha (for each projection matrix).
8. Accuracies recorded:

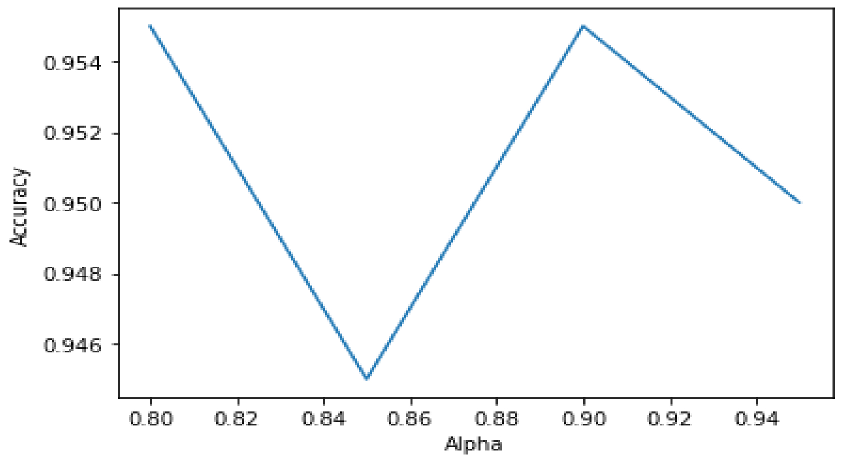
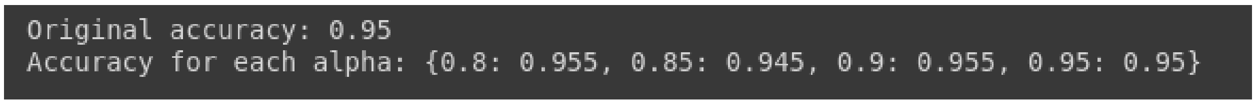


Figure 1: PCA Accuracies



```
Original accuracy: 0.95
Accuracy for each alpha: {0.8: 0.955, 0.85: 0.945, 0.9: 0.955, 0.95: 0.95}
```

Figure 2: PCA Accuracies dictionary

EXTRA: We added extra alphas to plot a more accurate relationship:

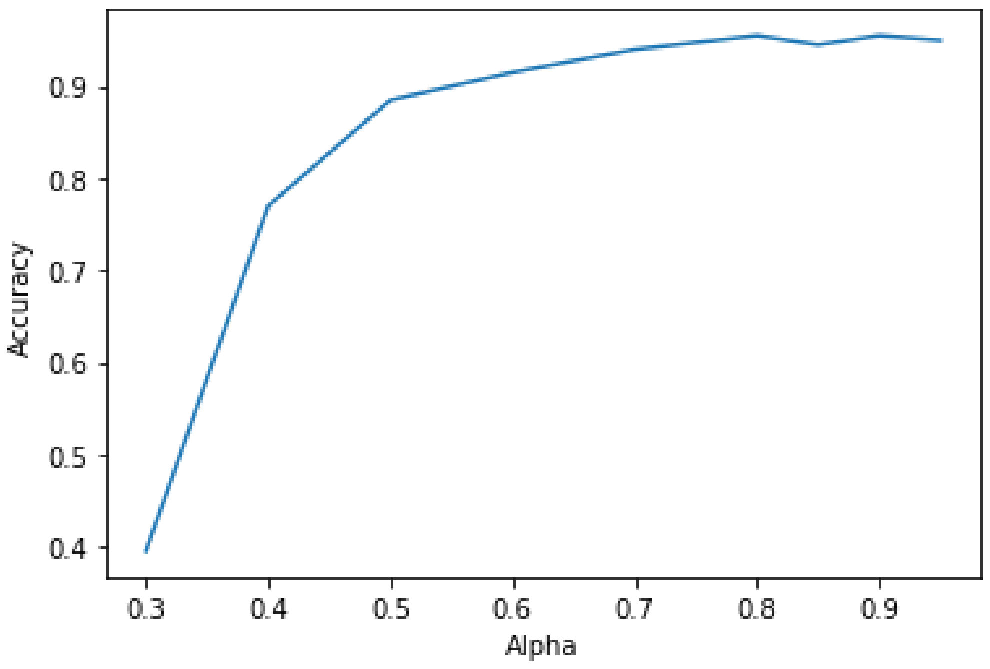


Figure 3: Accuracy for Each Alpha PCA

5. LDA

1. **Calculate class means:** for each of the 40 classes, this produces a matrix of shape (40, 10304).
2. **Calculate Sb the between class scatter matrix:** using the previously calculated class means matrix and the overall mean.
3. **Calculate the center class matrices Zi:** center class matrices by removing the mean from them.
4. **Calculate the class scatter matrices Si.**
5. **Calculate the within-class scatter matrix S:** by summing the 40 class scatter matrices.
6. **Compute dominant eigenvectors:** by taking the first 39 eigenvectors.

Final accuracy =0.96

```
accuracy: 0.96
```

6. KNN Tuning

We reran the classifier on the maximum accuracy alpha each time in the case of PCA algorithm and on the LDA algorithm.

The following plot is the plot of accuracy vs. number of nearest neighbors.

Tie breaking:

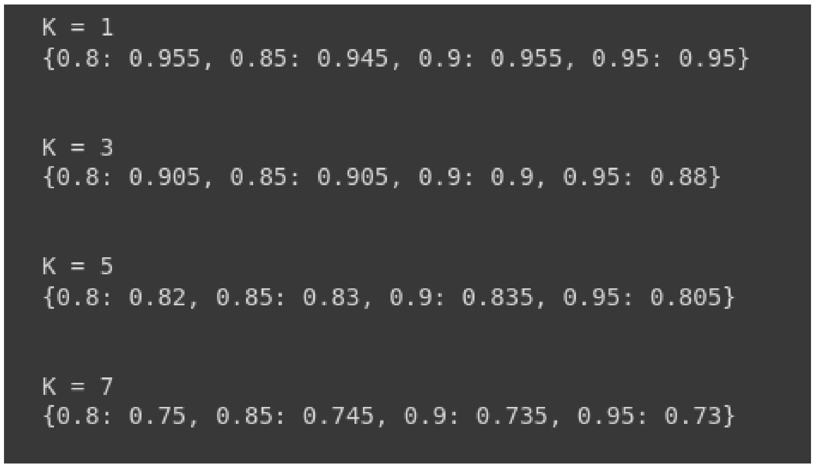
We left tie breaking to the default sklearn tie breaking strategy of KNeighborsClassifier

From the documentation for KNeighborsClassifier:

"Warning: Regarding the Nearest Neighbors algorithms, if it is found that two neighbors, neighbor k+1 and k, have identical distances but different labels, the results will depend on the ordering of the training data."

So, in the case of ties, the answer will be the class that happens to appear first in the set of neighbors.

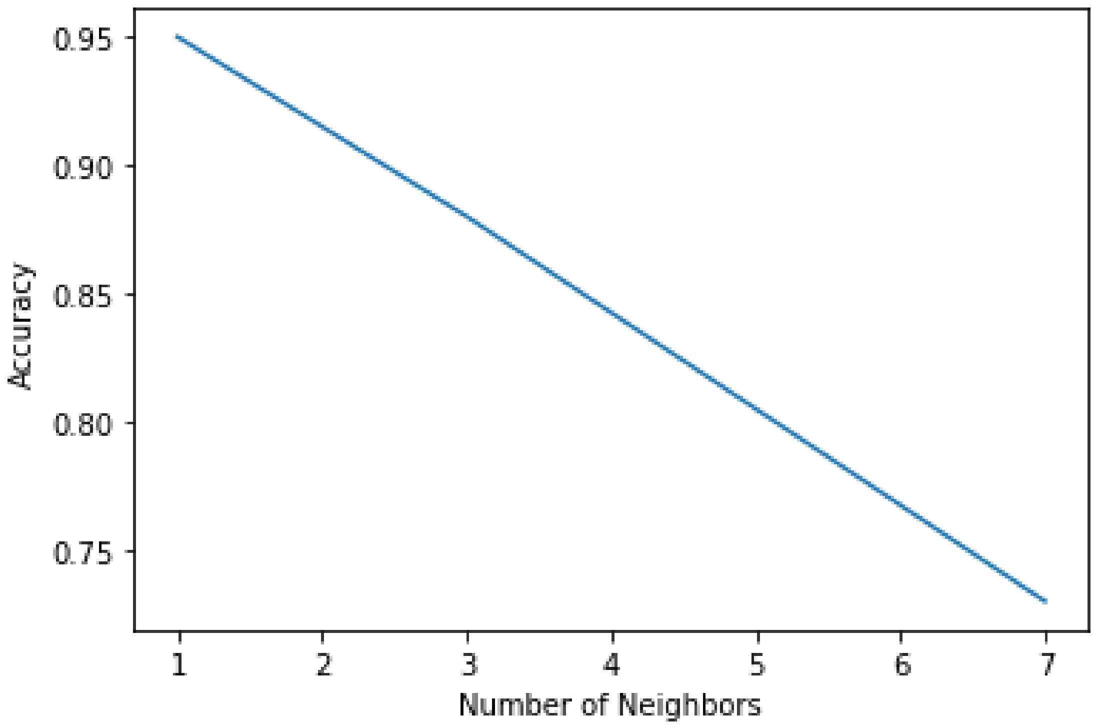
PCA:



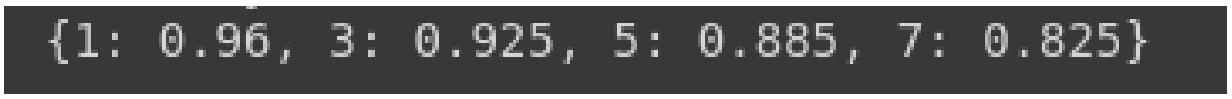
```
K = 1
{0.8: 0.955, 0.85: 0.945, 0.9: 0.955, 0.95: 0.95}


K = 3
{0.8: 0.905, 0.85: 0.905, 0.9: 0.9, 0.95: 0.88}


K = 5
{0.8: 0.82, 0.85: 0.83, 0.9: 0.835, 0.95: 0.805}


K = 7
{0.8: 0.75, 0.85: 0.745, 0.9: 0.735, 0.95: 0.73}
```

Figure 4: PCA per K Accuracy

We took the maximum accuracy for each K (best alpha) and plotted the relationship between K and accuracy.



LDA:



```
{1: 0.96, 3: 0.925, 5: 0.885, 7: 0.825}
```

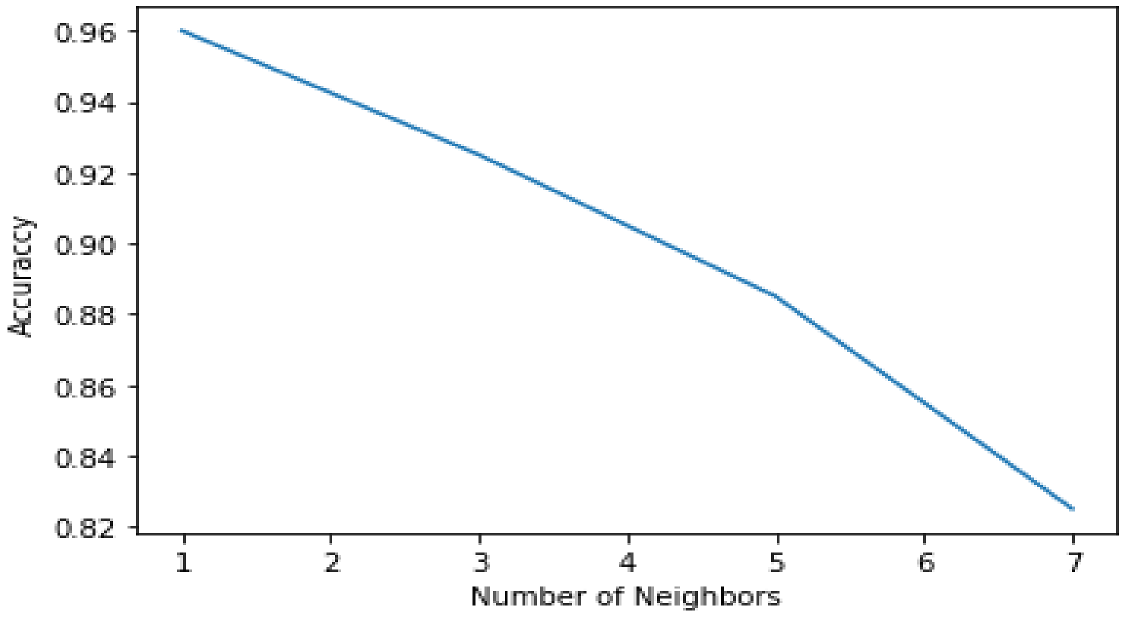


Figure 5: LDA per K Accuracy

7. Faces vs. Non-Faces

We load non-faces and crop them to the size 92x112.

Non-face images are of cats, dogs, and other animals as well as scenery.

In the same manner, we read them using the OpenCV library, and reshape them into a 10304-dimensional vector.

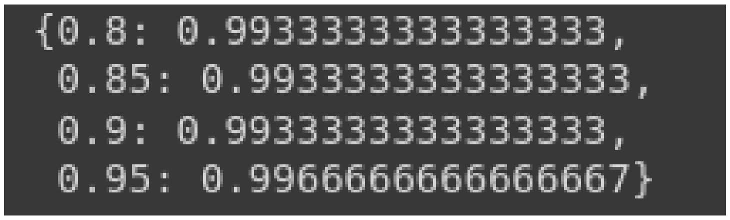
Firstly, to keep the 50-50 train/test split as well as balance face data and non-face data, we load 400 non-face images.

Our training set is now composed of 200 faces and 200 non-faces, and our testing set is the same.

Our labels are now 0 for non-face and 1 for face.

We run PCA for a couple of alphas and LDA as well on the classification task of face vs. non-faces.

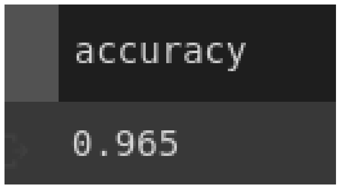
Below are the recorded accuracies for PCA with different alphas:



```
{0.8: 0.9933333333333333,
 0.85: 0.9933333333333333,
 0.9: 0.9933333333333333,
 0.95: 0.9966666666666667}
```

Figure 6: PCA Alpha Accuracy

The accuracy for LDA:



```
accuracy
0.965
```

Figure 7: LDA Accuracy

Increase number of faces in training set

For this, we loaded multiple numbers of non-faces and distributed them equally on the training and testing set.

The following numbers are used: [200, 300, 400, 500, 600, 700]

This means we divide 200 into 100 non-faces for training and 100 non-faces for testing, and so on.

The resulting plot is the accuracy vs. the number of non-faces in our data matrix.

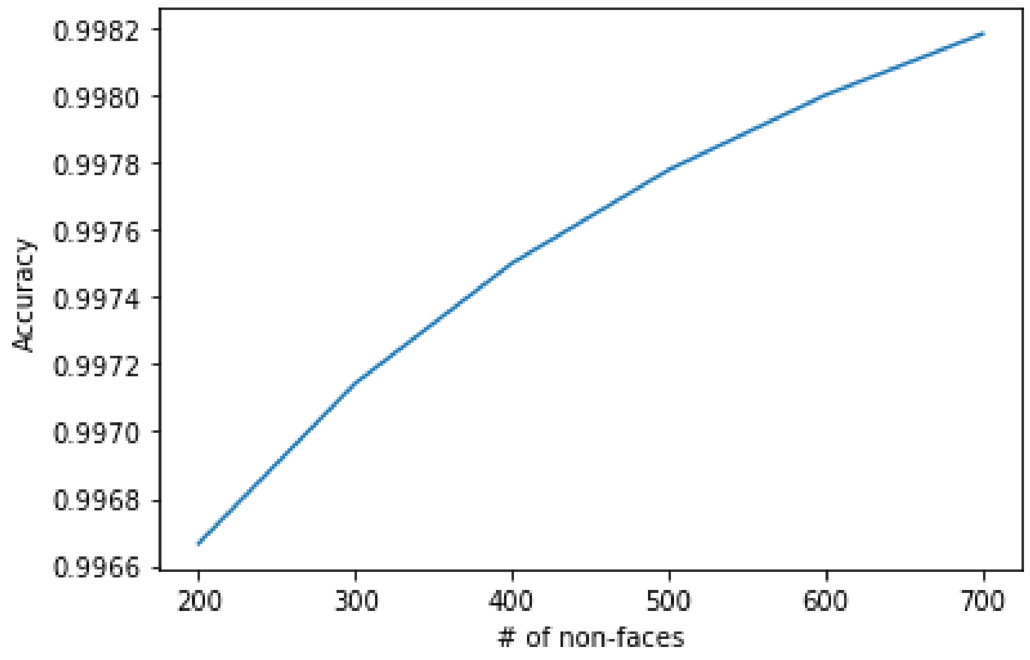


Figure 8: PCA

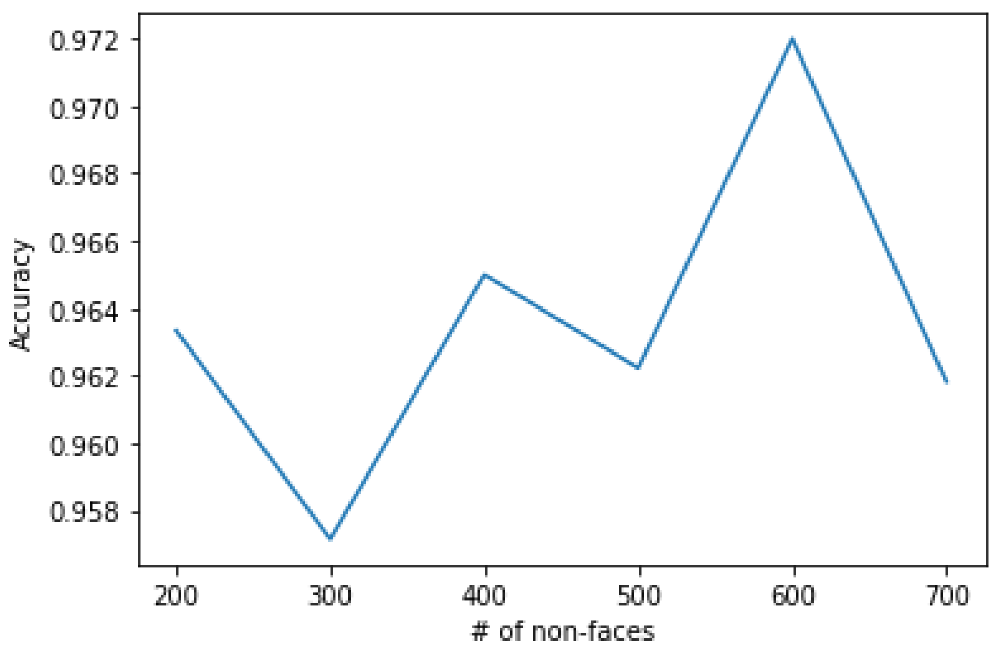


Figure 9: LDA

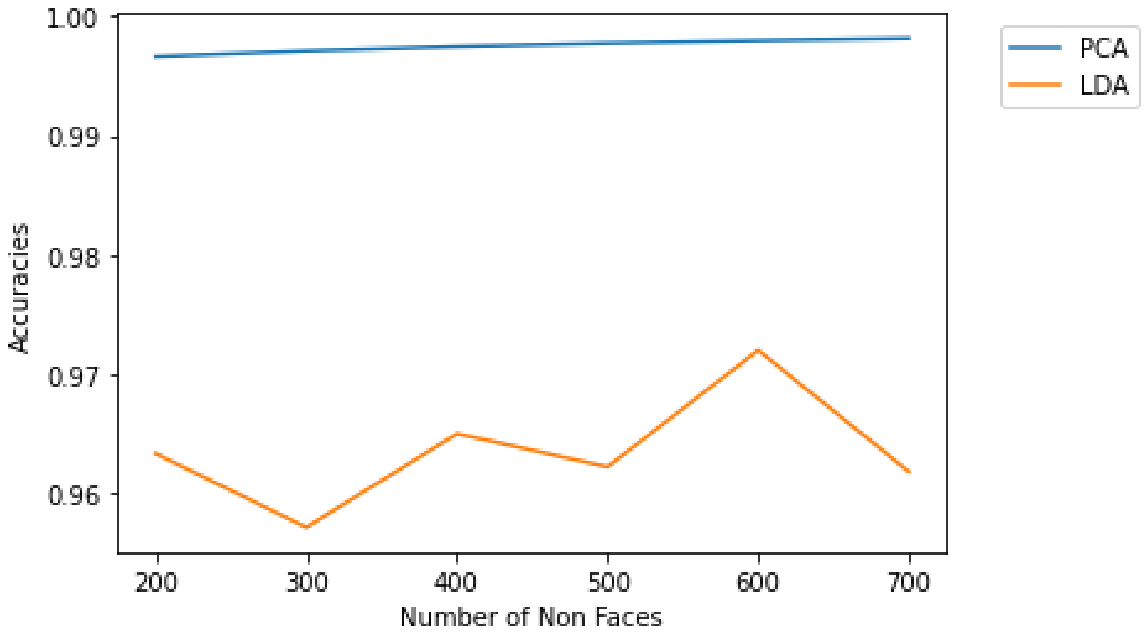


Figure 10: PCA vs LDA

Observations

PCA performs better in the case where the number of samples per class is less. Whereas LDA works better with large datasets having multiple classes; class separability is an important factor while reducing dimensionality.

This means, in the first task, LDA performed slightly better since there were 40 classes, but in the second task, PCA was visibly better.

8. Bonus

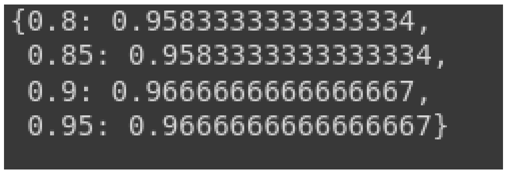
We tried different training and testing splits for the subject classification task (first task).

We have 400 faces. We take the first 7 instances of each class for training and the last 3 for testing.

We now have a new training matrix of shape (280, 10304) and a new testing matrix of shape (120, 10304).

PCA:

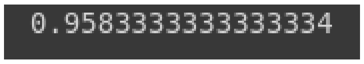
Resulting accuracies for different alphas:



```
{0.8: 0.9583333333333334,
 0.85: 0.9583333333333334,
 0.9: 0.9666666666666667,
 0.95: 0.9666666666666667}
```

(The 70-30 split is much better than 50-50 split)

LDA:



```
0.9583333333333334
```

(Both splits resulted in almost similar accuracies)

****