جامعة الإسكندرية
ALEXANDRIA
UNIVERSITY

Pattern Recognition
Assignment 1: Face Recognition

Ali Hassan Mekky 6850
Sohayla Khaled Abou Zeid 6851

Download the Dataset and understand the Format:

Downloading 10 images for each of 40 people, where each person in considered a class.

Generate the Data Matrix and label Vector:

- Using OpenCV library to read the images.
- Each image is reshaped from 92 x 112 to 1 x 10304 (flattened).
- There are 10 consecutive elements in the Y array having the value of the current person ranging from 1 to 40.
- Now there are data matrix with dimensions 400 x 10304 and Y matrix with dimensions 400 x 1.

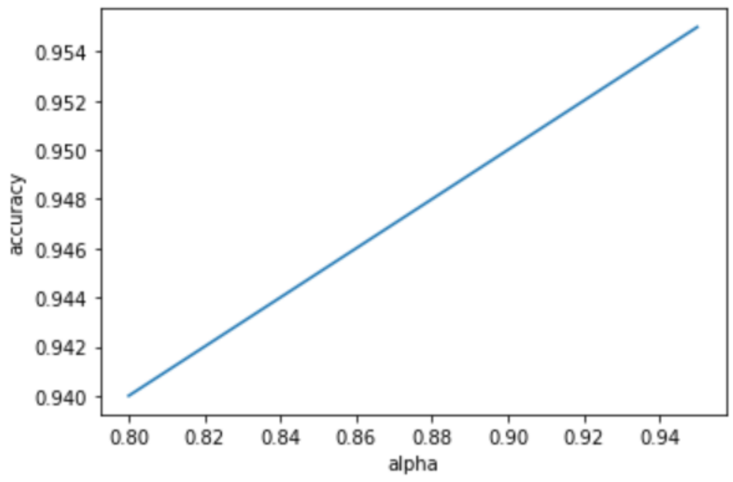
Splitting the dataset into training and test sets:

- From the data matrix and label matrix, the odd rows are kept for training set and the even rows for testing, therefore, there are 5 instances for each person in the training set and 5 instances of the same person in the test set.

Classification using PCA:

1) Computing the total mean of the whole dataset, mean per each column, we get a (1 x 10304) result.
2) Generating the centred data matrix "Z" we get a (200 x 10304) result.
3) Computing the covariance matrix we get a (10304 x 10304) result.
4) Getting the eigenvalues and vectors.
5) Reverse sorting the eigenvectors according to their corresponding eigenvalues.
6) Calculating the number of eigenvalues taken for each alpha by comparing the explained variance to each alpha.
7) Producing a projection matrix for each alpha by slicing the eigenvectors matrix, 4 projection matrices are produced.
8) Projecting the data on each one of the projection matrices.
9) Use the projected training data to train KNN classifier with 1 neighbour.

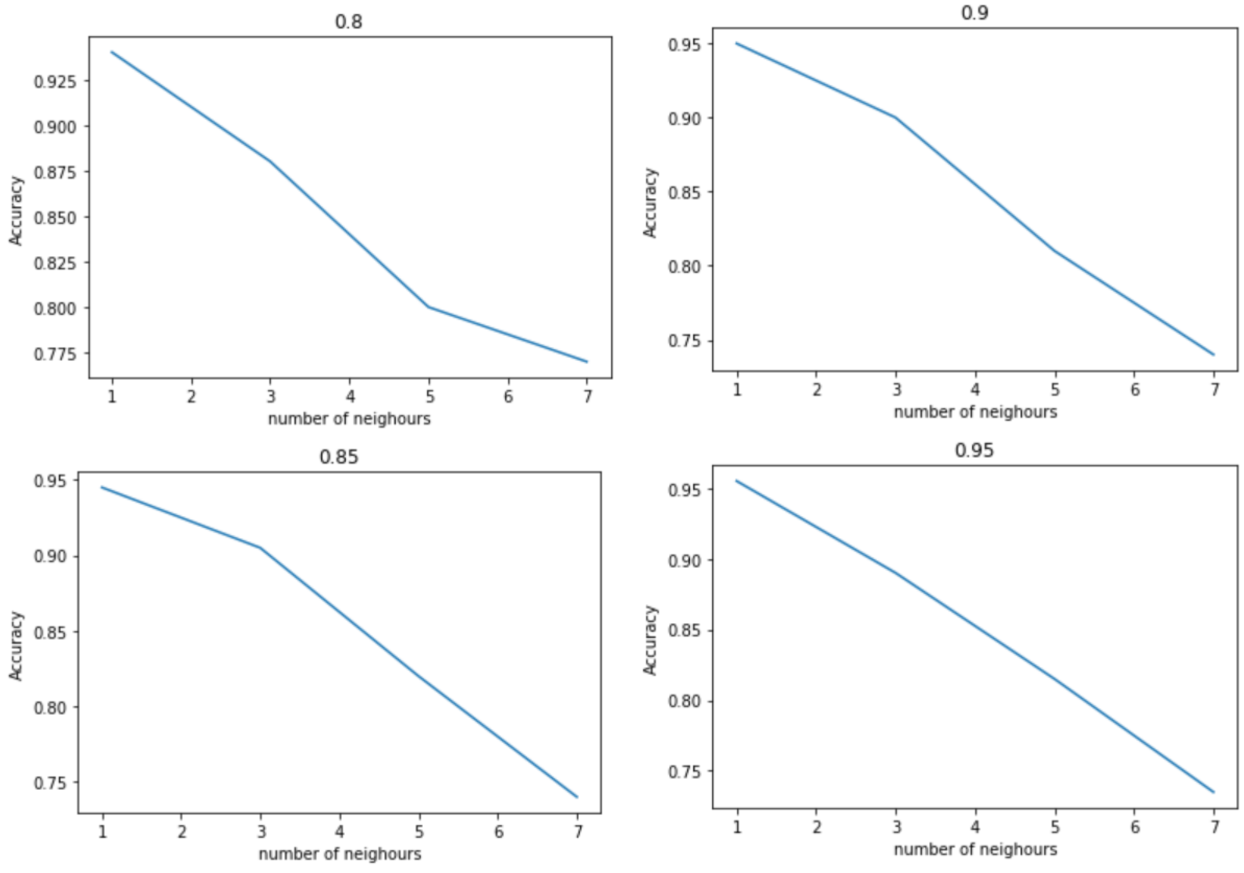
10) Recording accuracies:



PCA Tuning:

For each alpha:

- Training various versions of KNN models with different numbers of neighbours to identify the best k
- Number of neighbours to tune: 1, 3, 5, 7



- PCA Variant:

Accuracy after randomized PCA = 0.94 (for 35 components).

Randomized PCA: This is an extension to PCA which uses approximated Singular Value Decomposition(SVD) of data. Conventional PCA works in O(n*p2) + O(p3) where n is the number of data points and p is the number of features whereas randomized version works in O(n*d*2) + O(d3) where d is the number of principal components. Thus, it is blazing fast when d is much smaller than n. sklearn provides a method randomized_svd in sklearn.utils.extmath which can be used to do randomized PCA. This method returns three matrices: U which is an m x m matrix, Sigma is an m x n diagonal matrix, and V^T is the transpose of an n x n matrix where T is a superscript. Another way to use sklearn.decomposition.PCA and change the svd_solver hyperparameter from ‘auto’ to ‘randomized’ or ‘full’. However, Scikit-learn automatically uses randomized PCA if either p or n exceeds 500 or the number of principal components is less than 80% of p and n.

Randomized PCA is a variation of Principal Component Analysis (PCA) that is designed to approximate the first k principal components of a large dataset efficiently. Instead of computing the eigenvectors of the covariance matrix of the data, as is done in traditional PCA, randomized PCA uses a random projection matrix to map the data to a lower-dimensional subspace. The first k principal components of the data can then be approximated by computing the eigenvectors of the covariance matrix of the projected data.

Randomized PCA has several advantages over traditional PCA:

Scalability: Randomized PCA can handle large datasets that are not possible to fit into memory using traditional PCA. Speed: Randomized PCA is much faster than traditional PCA for large datasets, making it more suitable for real-time applications. Sparsity: Randomized PCA is able to handle sparse datasets, which traditional PCA is not able to handle well. Low-rank approximation: Randomized PCA can be used to obtain a low-rank approximation of a large dataset, which can then be used for further analysis or visualization.

By comparing to the accuracy of the traditional PCA by using 35 eigenvectors:
the accuracy of both = 0.94, Randomized PCA managed to get the same accuracy as the traditional PCA, this means it was able to approximate the first 35 components correctly (same accuracy but faster, traditional PCA takes 6 mins on average and randomized PCA takes few seconds)

Classification using LDA:

11) Computing the total mean of the whole dataset, mean per each column, we get 1 x 10304 result.
12) Computing the mean of all features per class, we get 1 x 10304 mean for each class.
13) Computing between class scatter matrix (Sb) using the following formula: 10304 x 10304 matrix

$$S_b = \sum_{k=1}^{m} n_k(\mu_k - \mu)(\mu_k - \mu)^T$$

14) Separating the data of each class and centering it, Z = X_c - μ_c
15) Computing within-class scatter matrix (Ss) using the following formula: 10304 x 10304 matrix

$$\mathbf{S}_i \leftarrow \mathbf{Z}_i^T \mathbf{Z}_i \,.$$

16) Getting the eigenvalues and vectors for the matrix $S_s^{-1}S_b$.
17) Choosing the first dominant 39 vectors to determine the new dimensions. Note: number of eigenvectors used is bounded by the number of classes - 1.
18) Project the training data and the new data on the new 39 dimensions, to get new training and testing data with dimensions (200,39)
19) Use the projected training data to train KNN model with 1 neighbor.
20) Accuracy of the model = 0.9

Comparing to PCA results: despite it being a classification problem, the accuracy using PCA was higher than that using LDA, this is because the face recognition problem is a special case where PCA works well because by the nature of the human being the variance between the features of each person is very small while the variance between different people faces is large.
Accuracy using PCA = 0.96
Accuracy using LDA = 0.9

LDA Tuning:

- Training various versions of KNN models with different numbers of neighbors to identify the best k
- Number of neighbors to tune: 1, 3, 5, 7

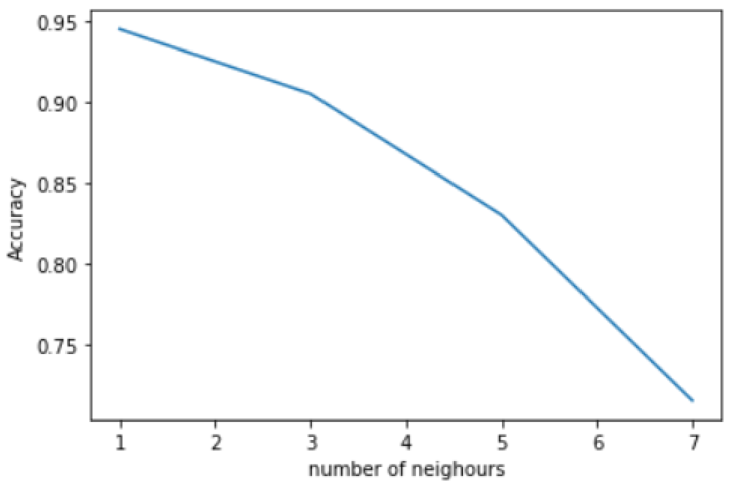


Fig 8.1

- **Observation:** accuracy decreases by increasing the number of neighbors because a large number of neighbors causes underfitting with this data as the number of samples per class = 5.

LDA Variant: The Shrinkage LDA

- Accuracy after shrinkage LDA = 0.97
- Idea: Shrinkage linear discriminant analysis (LDA) is a variant of LDA that uses a shrinkage estimator to regularize the covariance matrices of the classes. In normal LDA, the covariance matrices are estimated using the sample covariance of the entire dataset, which can be unstable and lead to overfitting. Shrinkage LDA addresses this issue by using a shrinkage estimator, such as the Ledoit-Wolf estimator, to regularize the covariance matrices and improve their stability, It is observed that there are many features and few numbers of samples in this problem, Therefore, Regularization will improve the performance.

Shrinkage Linear Discriminant Analysis (LDA) improves performance by addressing the issue of high-dimensional data and the curse of dimensionality. In high-dimensional data, the number of variables (features) is often much larger than the number of observations, making it difficult to estimate the parameters of a statistical model accurately.

Traditional LDA assumes that the covariance matrix of each class is equal, which may not hold true in high-dimensional data. Shrinkage LDA overcomes this limitation by introducing a shrinkage parameter that shrinks the sample covariance matrix toward a common covariance matrix. This common covariance matrix is a weighted average of the individual covariance matrices of each class.

By shrinking the covariance matrix towards a common covariance matrix, Shrinkage LDA reduces the estimation error and improves the accuracy of the classification. Moreover, it also prevents overfitting by regularizing the model and stabilizes the covariance matrix estimate. Shrinkage LDA has been shown to outperform traditional LDA in various classification tasks, especially in high-dimensional data with small sample sizes.

Breaking the Tie in KNN model:

- using sklearn: breaks the tie by choosing the element that comes first in the original data, this is the one used.
- choosing randomly between tied classes (not preferable)
- decreasing K until the tie breaks
- increasing K until the tie breaks, which may lead to underfitting if k exceeds the number of samples of the tied classes (not preferable)

Faces Vs Non-Faces:

1. Reading non-faces data:
    - Using the Same Library OpenCV to read the non-face data either from a google drive folder or a local folder, Reading a maximum of 600 non-face images, but will use only 400 images in training the model to balance the data between the 2 classes.
    - The images were reshaped to be 92x112 then flattened
    - Then, concatenating both face and non-face data to form the whole dataset.

- Initializing the labels of face data as 1 and non-face data as 0.
- Splitting the data into training and testing sets with 50% training and 50% testing.

2. Applying PCA on the new data:

   Using the same Algorithm discussed in the PCA section earlier, The data is projected into a new space whose dimensions are determined by alpha, Then, Training a KNN model with 1 neighbor with the new projected data.
   Accuracies: {alpha = 0.5: 0.9875, alpha = 0.75: 0.995, alpha = 0.8: 0.995, alpha = 0.95: 0.9975}
   Tuning the KNN model with different numbers of neighbors to get the best k with each alpha:

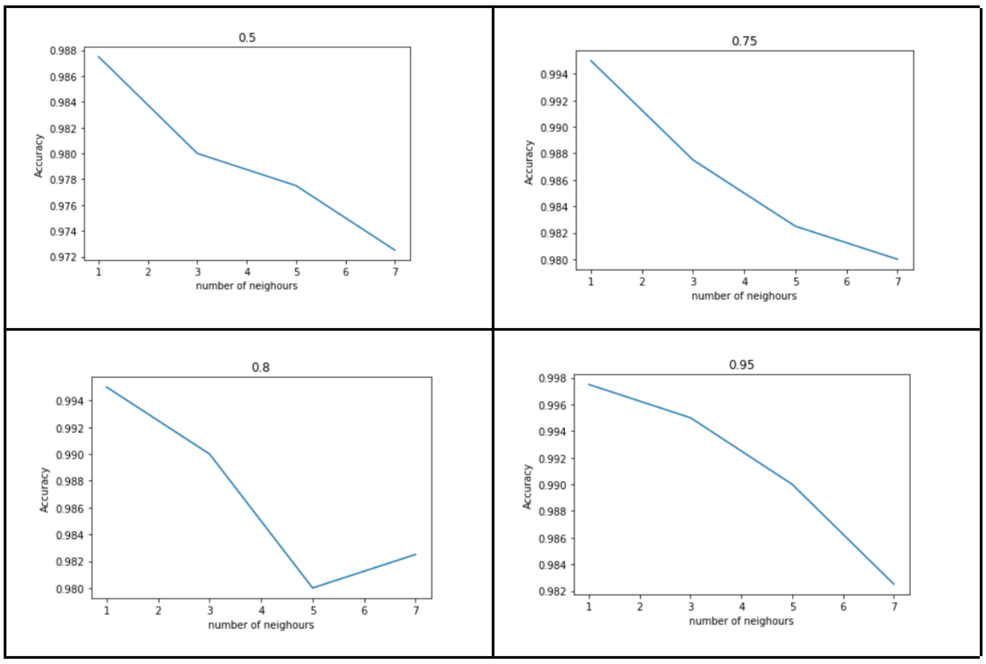


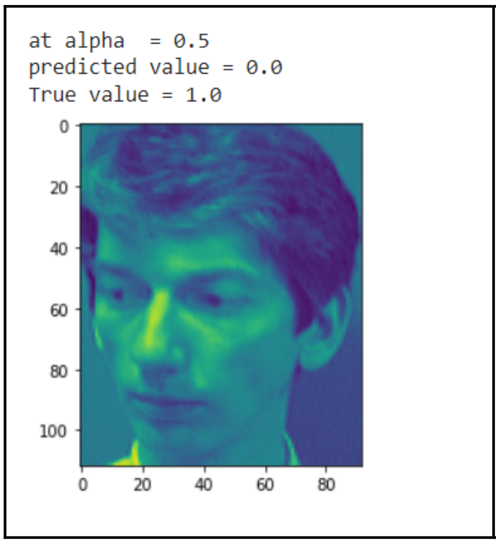
Fig 11.1

Observation: K = 1 is the best value, therefore, we continue with k = 1.

3. Failure Cases using PCA:

   As the data are not shuffled and was stacked together, we can track the failure test case to get its index in the original data to plot and analyze it. First, we compare the predicted output to Y_test to get the failure cases, then, we track them by applying inverse calculations to get the index and plot them, examples are shown in the following figure:

```
at alpha  = 0.5
predicted value = 0.0
True value = 1.0
```



```
at alpha  = 0.95
predicted value = 1.0
True value = 0.0
```

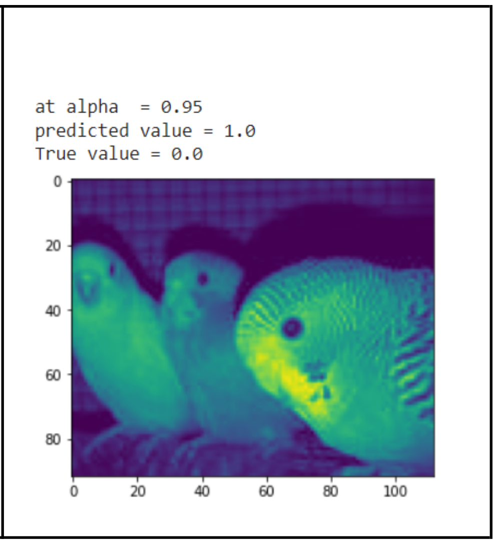


Fig 11.2

A Confusion matrix was used to analyze the failure test cases:

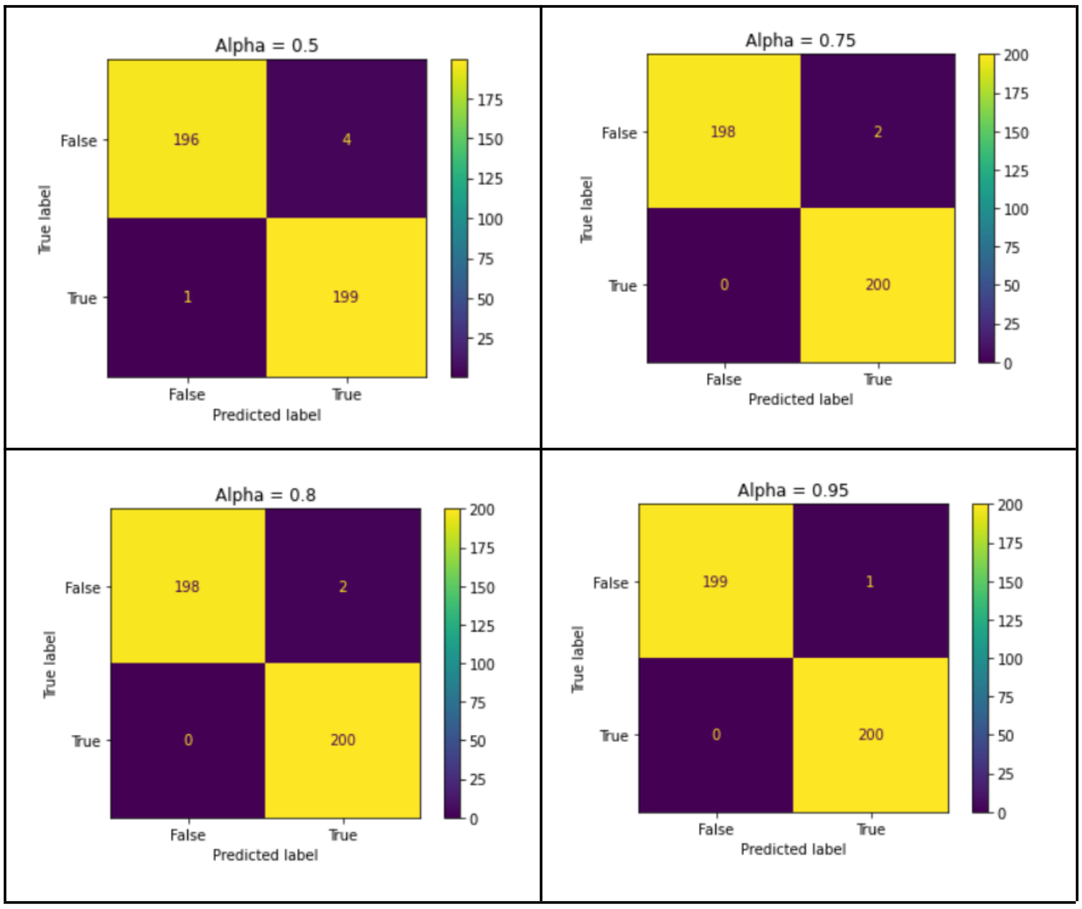


Fig 11.3

4. Applying LDA on the new data:

   Using the same algorithm discussed earlier to project the data into a new space using only one eigenvector, the reason for this is that LDA seeks to project the data onto a lower-dimensional space while preserving the class-discriminatory information as much as possible. The number of components determines the dimensionality of the new space onto which the data is projected. If the number of components is greater than the number of classes minus one, then the projection will be over-determined and may not provide useful discriminative information.

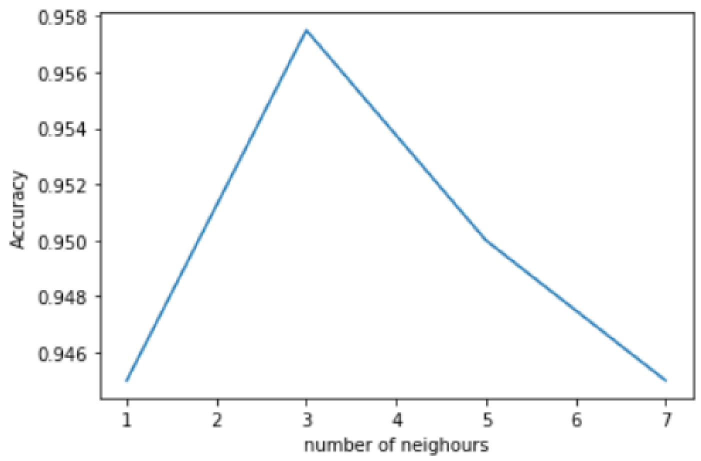
   On the other hand, if the number of components is less than the number of classes minus one, then the projection may not be able to capture all of the discriminative information, and some information about the classes may be lost.

   Therefore, the number of components in LDA is chosen based on the balance between the dimensionality of the new space and the amount of discriminative information that needs to be preserved, **and the number of eigenvectors chosen is bounded by the number of classes minus 1 (in binary classification: only 1 eigenvector used)**.

   Training a KNN model with 1 neighbor with the projected data:

   Accuracy = 0.945

   Then, Tuning the model to get the best number of neighbors:



```
[0.945, 0.9575, 0.95, 0.945]
```

Fig 11.4

5. Failure Cases using LDA:

   By tracking down the failure test cases to get their index in the original data, we plot and analyze them, an example in shown in the following figure:

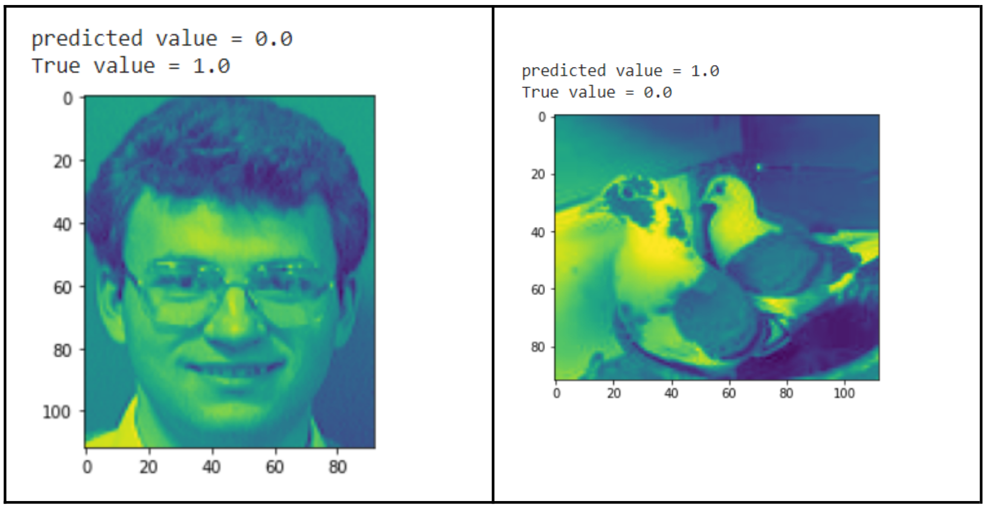


Fig 11.5

A confusion matrix is used to analyze failure test cases:

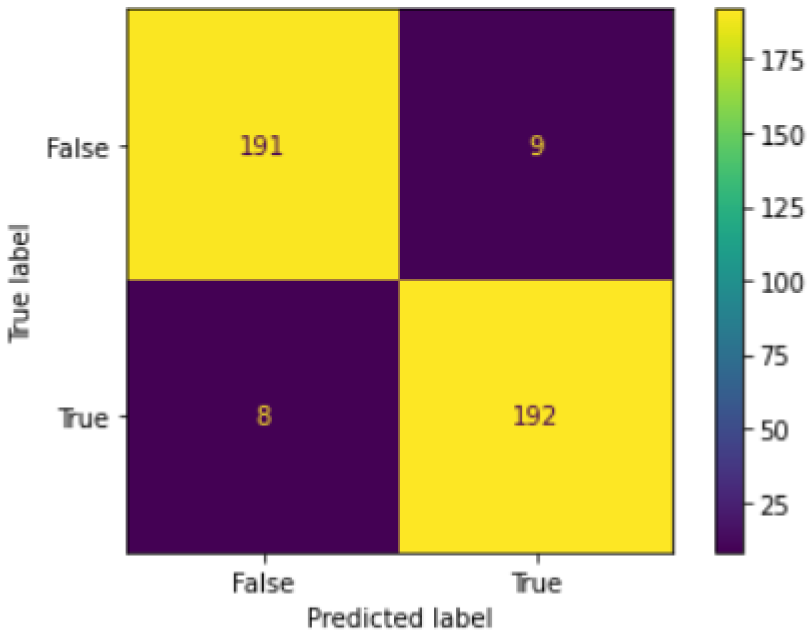
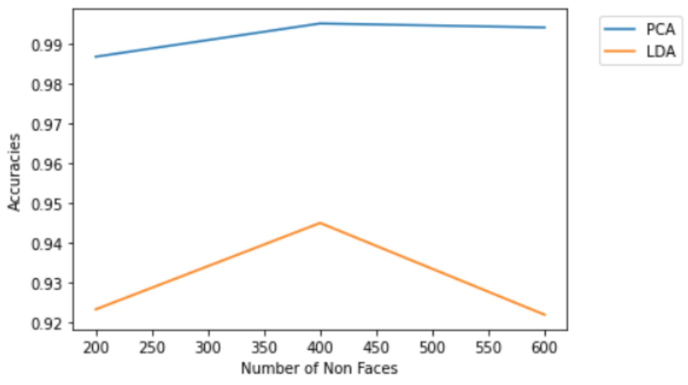


Fig 11.6

6. Increasing number of non-face data with keeping the number of face data constant:

   We will use different numbers of non-face samples from the 600 read images.
   The numbers of non-face samples used are
   - 200: the majority from faces data.
   - 400: balancing between the two classes.
   - 600: the majority from non-face data.

By plotting the obtained accuracies



```
PCA accuricies[0.9866666666666667, 0.995, 0.994]
LDA accuricies[0.9233333333333333, 0.945, 0.922]
```

Fig 11.7

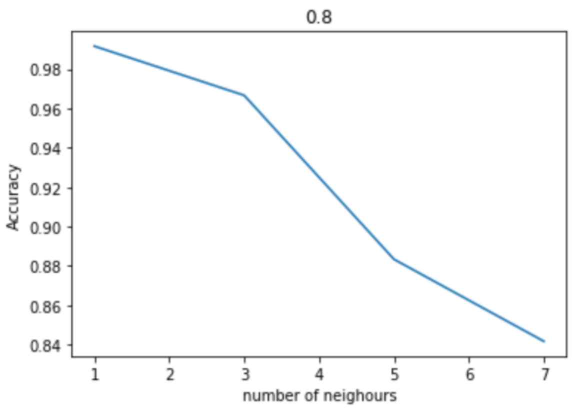
Observation: increasing the number of non-faces images while fixing the number of faces images, leads to imbalances between the number of samples in each class, and this leads the algorithm to be biased towards the majority class, which will reduce its performance.

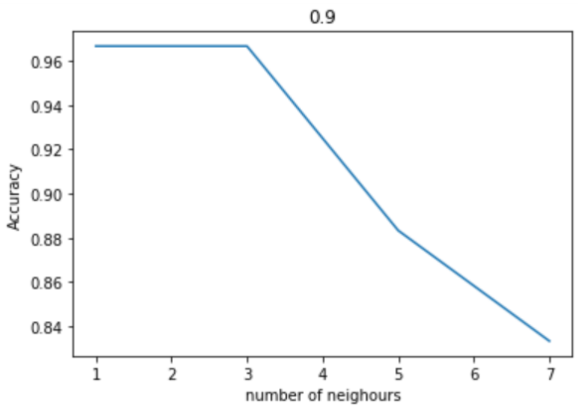
Bonus:

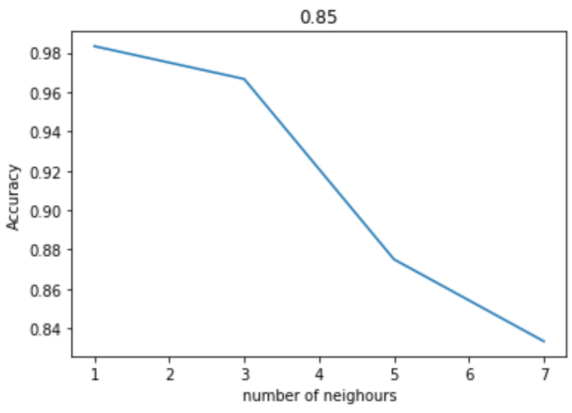
1. Splitting the data:
   - From the data matrix and label matrix, 7 instances from each class are kept for training and the 3 for testing, therefore, there are 7 instances for each person in the training set and 3 instances of the same person in the test set.
   - A training matrix of shape (280 x 10304) is generated.
   - A testing matrix of shape (120 x 10304)

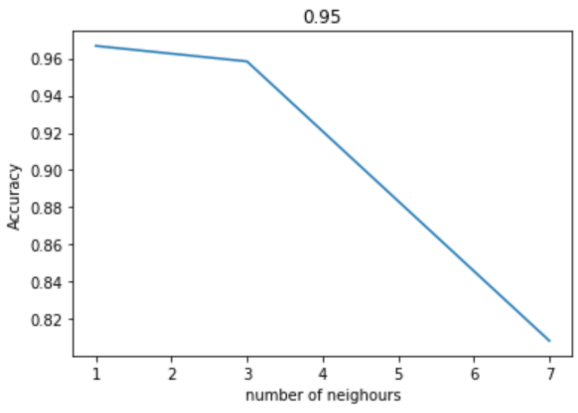
2. Applying PCA on the new data:
   -Using the same Algorithm discussed in the PCA section earlier, The data is projected into a new space whose dimensions are determined by alpha, Then, Training a KNN model with 1 neighbor with the new projected data.
   Accuracies: {alpha = 0.8: 0.9916, alpha = 0.85: 0.983, alpha = 0.9: 0.9667, alpha = 0.95: 0.9667}

-Tuning the KNN model with different numbers of neighbors to get the best k with each alpha:









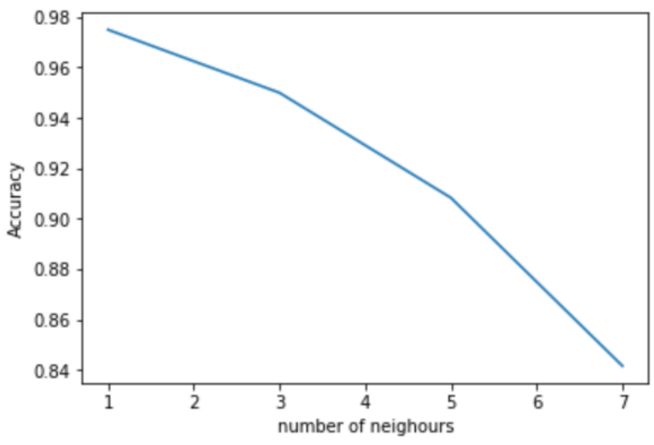
3. Applying LDA on the new data:

Using the same algorithm discussed earlier to project the data into a new space using 39 eigenvectors.

Accuracy = 0.975

4. LDA tuning:

Tuning the model to get the best number of neighbors:



```
[0.975, 0.95, 0.9083333333333333, 0.8416666666666667]
```

Google colab Notebook:

To View the assignment Notebook:
https://colab.research.google.com/drive/1Xe3b_iO7gSmYj9Gf2rmgU9T26ROxQ1X1?usp=sharing