Assignment 1
Pattern Recognition
Report
Face Recognition

Esraa Wael El-Hawash 3959

Guehad Mohamed Ahmed 3861

Assignment 1 Face Recognition

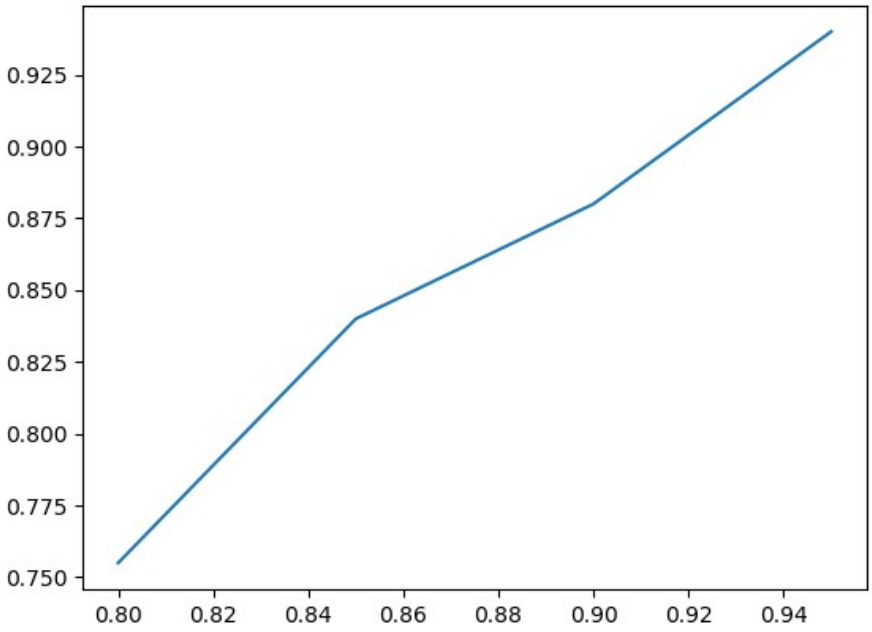
We were asked to perform face recognition. Face recognition means that for a given image you can tell the subject id.
Our database of subject is very simple. It has 40 subjects.
The database is a folder of 40 folders each of which contains 10 different pictures of the same person giving a total of 400 pictures all together
Every image is a grayscale image of size 92x112.

First step:
the first thing we did was load the data into out program, and we did so by writing in the directory where the folder containing all pictures is at.
Next we made a loop that goes to the directory location that looks for the 40 folders and loads them in the program, then another loop that opens the chosen file from the previous loop's iteration looks for the images inside said folder and opens all image inside the folder by using Image.open() function once that's done the immediate step afterwards is to turn that image into a vector using the Numpy library which proved very useful in this assignment 'np.reshape()'.
Following this the vector is appended into an array 'A' that will store all image vectors in one place.
Then, we used Pandas library to create a dataframe that contains our traininng data and our test data using texts and labels , meaning it will contain every image vector and the label means the class or in this case 'person' this image belongs to.

Second step:
we created 4 arrays text_train=[] text_test=[] label_train=[] label_test=[]
to hold training data's image vector and classification , same goes for the test data.
Which leads us to the part of splitting our given database into a training set to train our model on and a test set to test our model with and check out the results and the accuracies.
We did the split using a loop and did as we were asked to do which was to take all even rows of our data matrix as training set and all odd ones to join the test data giving us a 50-50 split on our database.
We did so by using remainder on rows of our created data matrix.

Third step:
comes the classification part we were asked to implement from scratch two classifiers : PCA & LDA
let's start with
PCA:
following the pseudocode the first thing we did was calculate the mean of the training data using np.mean() , Next we subtracted the mean from the original data matrix (training data).
After that we calculate the covariance matrix using np.cov() function following this we use np.linalg.eigh() function to calculate the eigen values and eigen vectors.
The sum of eigen values is calculated and we iterate on all our eigen values and we calculate the fraction of total variance by dividing the number of values by the sum previously calculated and inside this loop we check if the division is bigger than or equal to the alpha value which is sent to the pca function and if this condition becomes true at any point in the program the loops breaks and the iteration in which this occurred is saved in a variable.
Next is the projection matrix which is formed from the eigen vectors, from the first eigen vector column till the number of iteration saved in the loop mentioned above.
We then calculate our projected data by multiplying the training data by the projection matrix using np.dot() and this marks the end of our pca function steps.
Next thing we do is the testing which happens by multiplying the test data by the projection matrix same as we did for our training data.
And we use the KNN classifier to classify our test data and figure out out accuracy
we call the KNN function using the sci-kit learn library *from sklearn.neighbors import KneighborsClassifier.* We then use the knn.fit() to fit the training data using the knn classifier and knn.score() function to find the accuracy of our model so we use it on the test data and we find out how accurate it was in classifiying the images given to it.

Now all this is done within a loop that sends to the PCA function different values of alpa alpha =[0.8,0.85,0.9,0.95].
Each alpha value gives out a different accuracy and we made a plot to show exactly the relation between those two values.



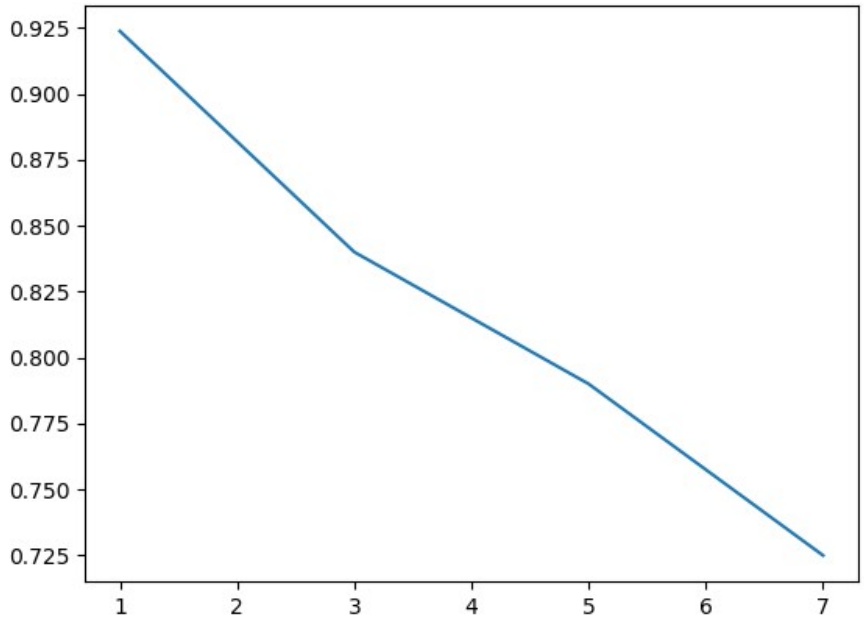
Notice that the accuracy increases considerably with increasing the alpha
to give you a more numeric sense :
when alpha =0.8 accuracy= 0.755
0.85 0.84
0.9 0.88
0.95 0.94
this concludes the PCA function.

<u>LDA:</u>
now for the LDA function heres what we did.
Consider every folder a class and we take 5 samples of that class as our training data as explained before.
We go in a loop that takes the first 5 samples or in this case vector images and append it to a list of class matrices.
And in that same loop after getting the samples of the first class we find the mean of it using np.mean() and then we append it to a list of class means which will hold all 40 means of all classes/folders.
Next we calculate the total mean which as the pdf explained would be calculated as the sum of number of samples of one class divided by all classes times the mean of the specific class
we did that using a for loop and we have that sum as our total mean which is then used in calculating Sb (between class scatter matrix)
in a loop we calculate the class mean of every class minus the total mean which gives us a value lets call it P, and then we calculate Sb by summing the values of the multiplication of samples by P and P transpose.

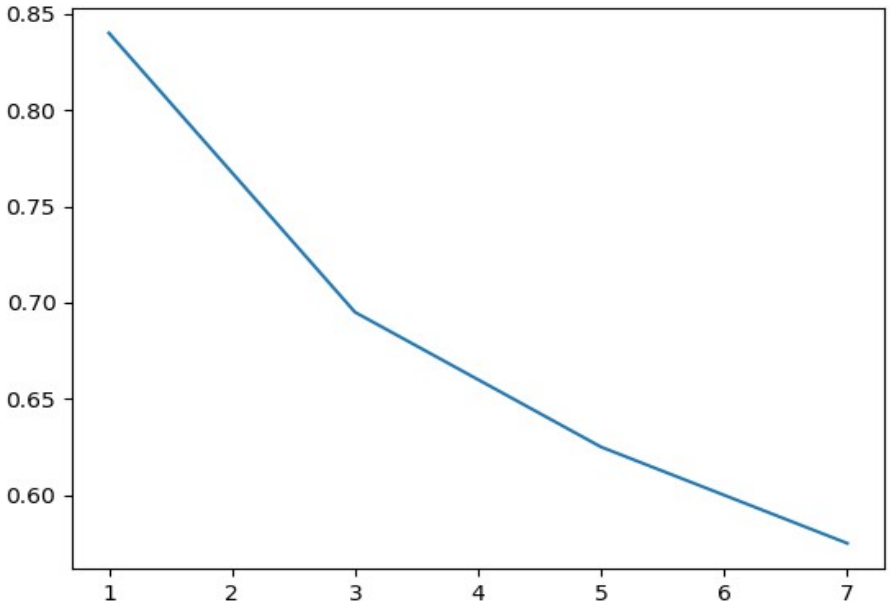
Now that we have Sb we sit on it for a while till we calculate Si (class scatter matrices)
starting by centering the data which is done subtracting every class matrix by its mean, afterwards we multiply every matrix by it's transpose and we append that into an array for all classes.
The S (within class scatter matrix) is calculating by summing all values in the Si array.
We then get the inverse of S using np.linalg.pinv(S).
we reach the part of calculating the eigen values and vectors of the product of Sb and Sinverse using np.linalg.eigh() .
The projection matrix is formed using 39 dominant eigen vectors, we did it using this line *projectionMatrix = eigen_vectors[:, eigen_vectors.shape[1]-39:eigen_vectors.shape[1]]* essentially what this does is choose the first 39 columns of the eigen vector matrix to form the projection matrix with it.
Projected data is then simply the dot product of our training data matrix and the projection matrix
repeating the steps previously mentioned in pca we classify using KNN classifier and we use the same fit() and score() functions to fit our data and test them and find out how accurate our model turned out to be.

Step Four:
Tuning:
tuning for a KNN classifier is quite simple we repeat the classification process described in both PCA and LDA for a set of different K values [1,3,5,7,9…]
we plotted the performance measure (accuracy) against the K value for both classifiers and here are the results we found .

PCA:



LDA:



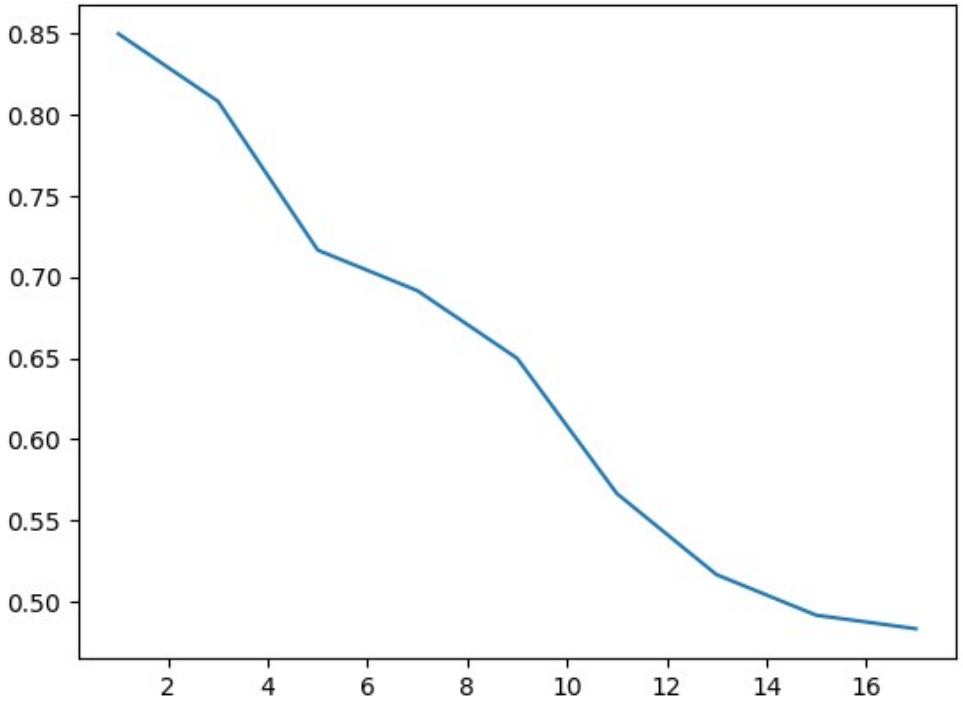
Notice that the accuracy keeps dropping the higher the k we choose.

Bonus

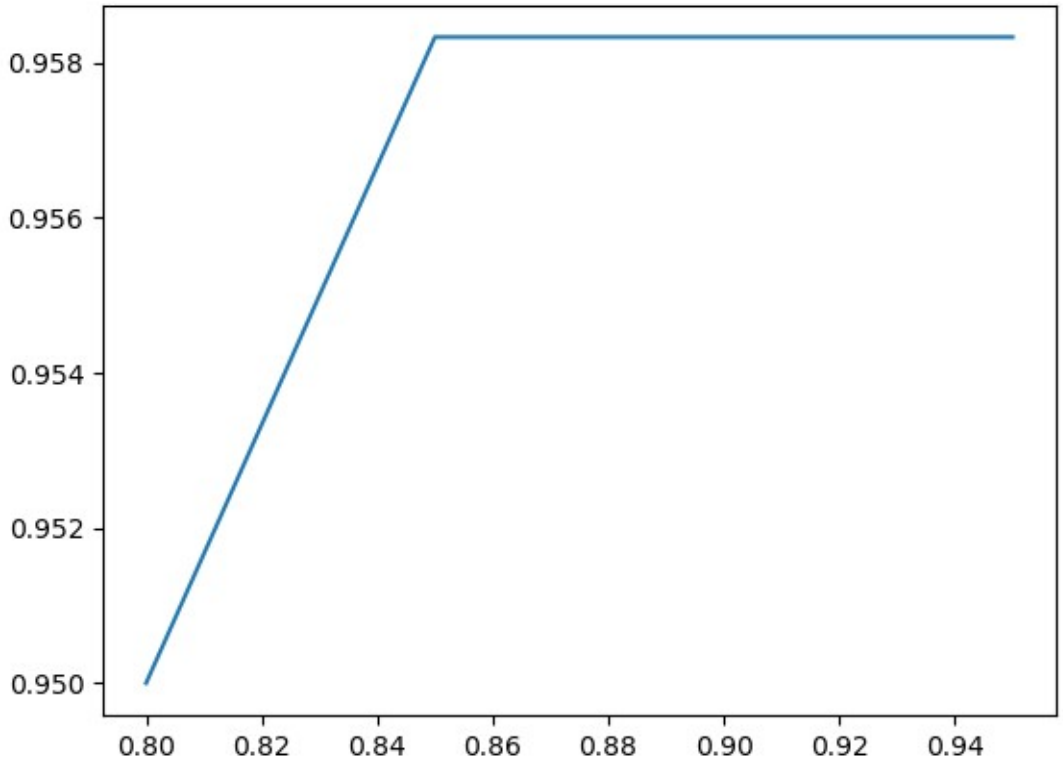
for this part we tried to split our training and test data sets differently
instead of a 50-50 split we tried a 70-30 split with 70% of data in the training set and the others in the test set.
We did this by using a function called split70() and in this function we iterate over all rows in the data matrix and check if their remainder by 10 is bigger than 2 then we append them to our training data if not they go to the test set and with this strategy they're split exactly 70-30 with about 280 vectors in training and 120 in testing.
We ran the same classifiers on this new split.

And we found the following results :

For the LDA:
we notice a ***slight*** increase in the accuracy for the LDA
with the 50-50 split the accuracies were 0.84, 0.69, 0.625 for K = 1,3,5 respectively.
And for the 70-30 split they came at 0.85, 0.808, 0.716.
this is due to the larger fragment of database that the model got to train on.
However we notice that the accuracies keep dropping with increasing the K regardless of how we split as shown in the plot below.



For the PCA:
we notice an increase in accuracy the accuracies increased to 0.95, 0.9083, 0.8583, 0.825 for K=1,3,5,7 respectively.
However we notice that the accuracies keep dropping with increasing the K regardless of how we split as shown in the plot below.



Conclusion:
it was expected for the accuracies to increase because the model was trained on a much larger dataset which lead to better results for our program.