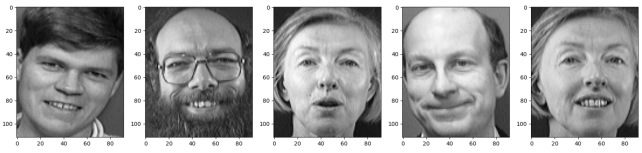
Introduction to Machine Learning
Assignment 3 Report

Nour Mohamed Mahmoud 7591 Ahmed Moghazy 7397
Khalid Osama 7693

1 Introduction

Our Dataset: There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). A preview image of the Database of Faces is available.

- Here is 5 random samples from our dataset:



2 Classification Procedure

In this project we intend to perform face recognition. Face recognition means that for a given image you can tell the subject id.

Principal component analysis :

PCA is also known as Karhunen Loeve projection. Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components and referred to as Eigenfaces. In PCA, the main idea to re-express the available dataset to extract the relevant information by reducing the redundancy and minimize the noise. We used a nearest neighbor classifier works afterwards by comparing compare a face's position in Eigenfaces subspace with the position of known individuals and the distances between them.

Pseudo code algorithm:

PCA algorithm steps and procedure is provided in Figure below:

ALGORITHM 7.1. Principal Component Analysis

PCA $(\mathbf{D}, \alpha)$:

1 $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ // compute mean

2 $\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \boldsymbol{\mu}^T$ // center the data

3 $\boldsymbol{\Sigma} = \frac{1}{n}\left(\mathbf{Z}^T\mathbf{Z}\right)$ // compute covariance matrix

4 $(\lambda_1, \lambda_2, \ldots, \lambda_d) = \text{eigenvalues}(\boldsymbol{\Sigma})$ // compute eigenvalues

5 $\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \end{pmatrix} = \text{eigenvectors}(\boldsymbol{\Sigma})$ // compute eigenvectors

6 $f(r) = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$, for all $r = 1, 2, \ldots, d$ // fraction of total variance

7 Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality

8 $\mathbf{U}_r = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r \end{pmatrix}$ // reduced basis

9 $\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{U}_r^T \mathbf{x}_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

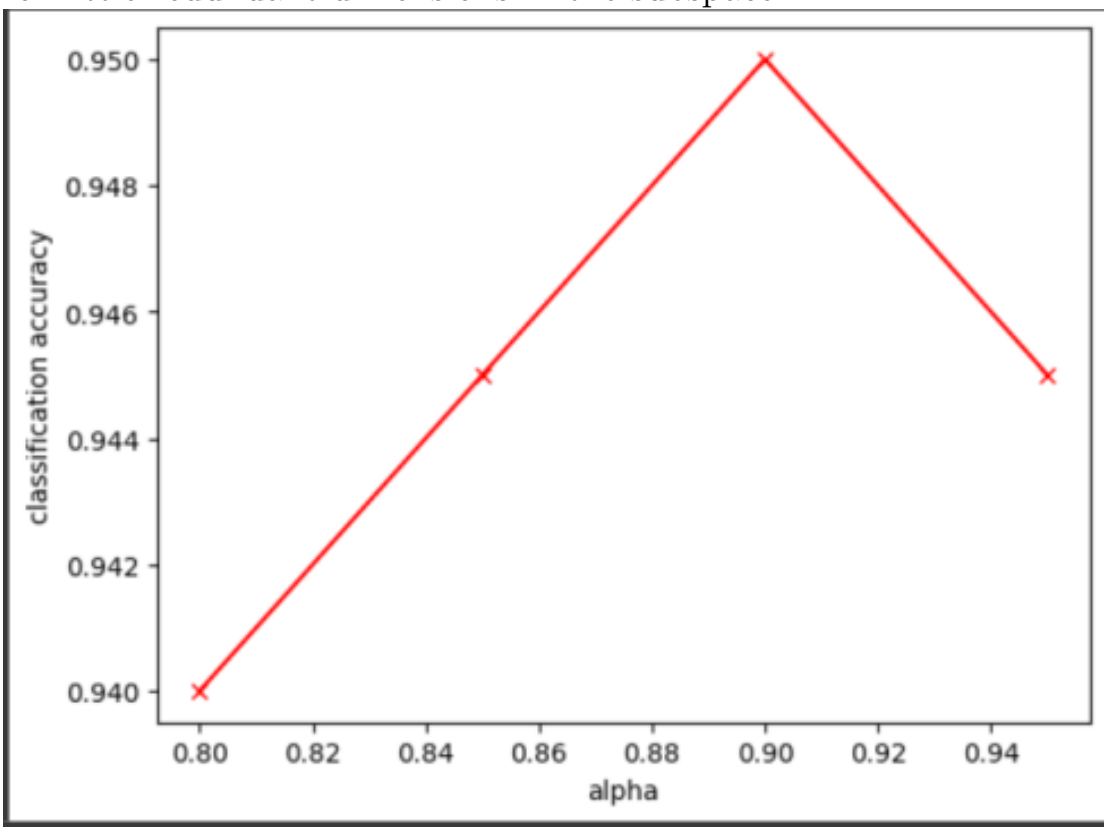
PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. PCA calculates the Eigenvectors of the co-variance matrix, and projects the original data onto a lower dimensional feature subspace, which is defined by Eigenvectors with large Eigenvalues. The coordinates of the data points in the lower subspace dimension is computed and fed to the classifier after this algorithm job ends.

3 Implementation and Results

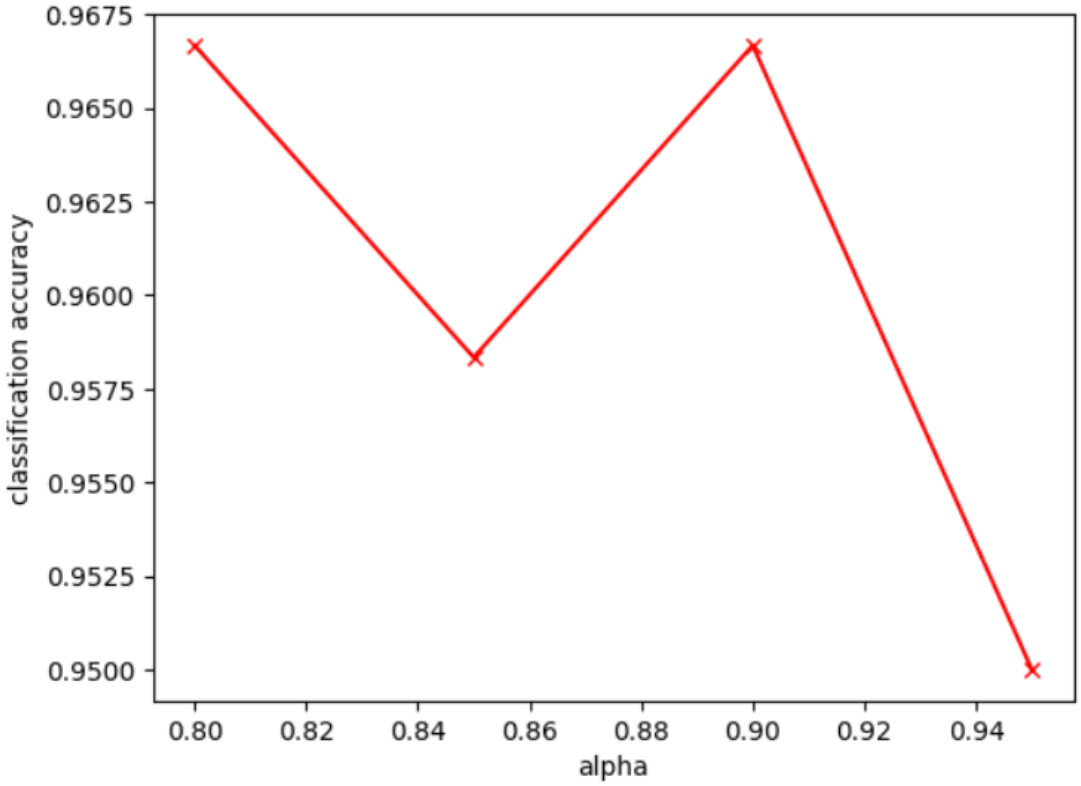
We implemented the above algorithm using Python and NumPy arrays. The results shown in this section are related to the 50-50 data split test, results on the 70-30 data split test can be obtained from the source code attached and referred to later. Computing the mean vectors of each dimension, normalizing the data, and computing the co-variance matrix, Eigenvalues and Eigenvectors. After Projection of data, we obtain the coordinates A matrix and use finally the simple k-nearest neighbors' classifier to determine the class labels of testing data. We established four trials when choosing the dimension (r) based on four different values of alpha: 0.8 , 0.85, 0.9, 0.95 which yielded different values of (r) and recomputed the accuracy of the classifier for each trial. Figures below show a plot of the relation between(alpha)

and accuracy of the KNN classifier.

From the figures below we can deduce that as alpha increases, the number of reduced dimensions required to achieve that alpha value increases. It is then obvious from figure 1, that after a certain alpha value the accuracy actually starts decreasing, which is the case when it is set to 0.95. So, we can deduce the best alpha value on Fig. 1 is 0.9 which achieves very good accuracy with no or little redundant dimensions in the subspace.



50-50 split



70-30 split

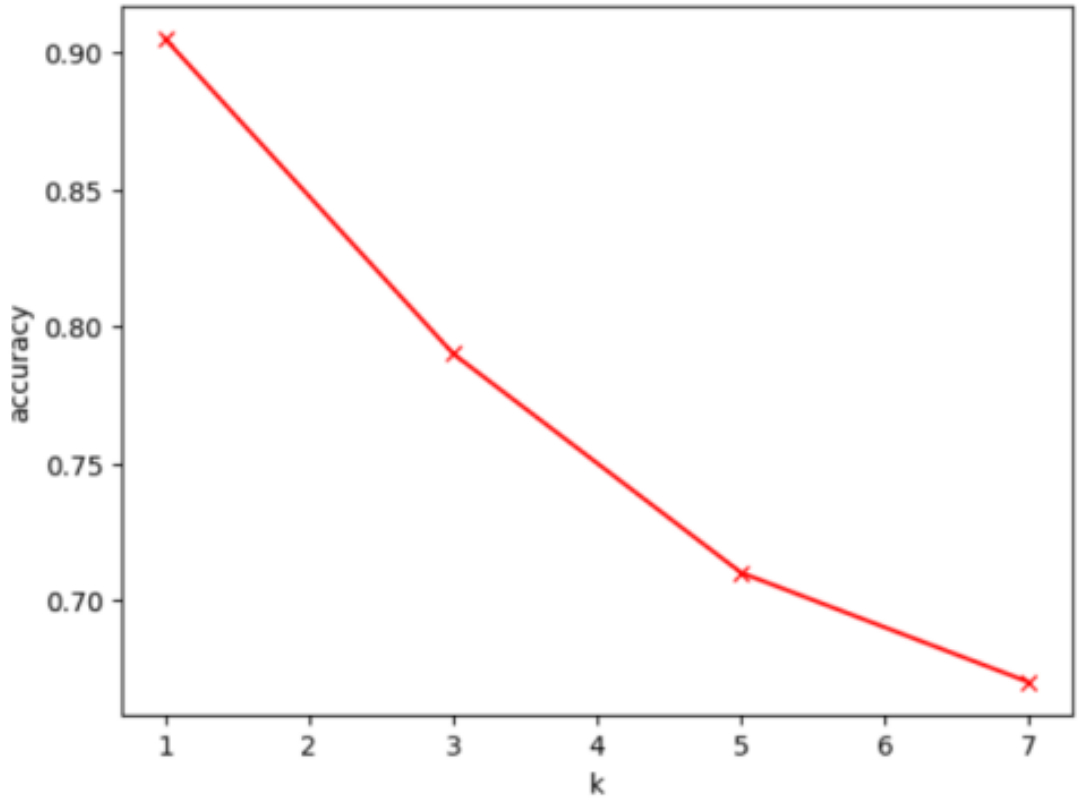
4 K-Nearest Neighbors (KNN) Classification

We used simple nearest neighbor classifiers to determine the class labels of testing data.

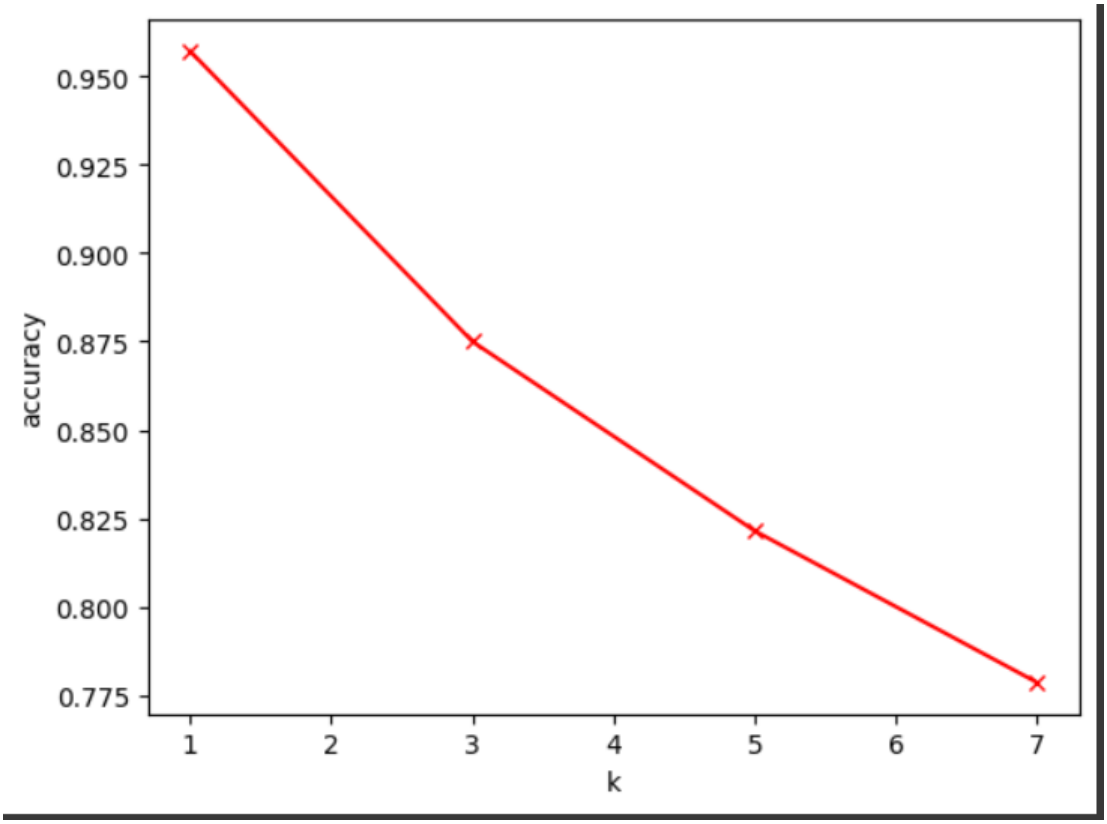
On the training stage we managed to iterate over all projections and get the accuracy for every alpha on the

K nearest neighbors (1, 3, 5, 7). Then we tested our model with the projected test matrix to predict the class labels and print out the accuracy for the best K. Here is some figures about the accuracies :

From the figure below we can see that the accuracy decreases as the k value increases.



the performance measure (accuracy) against the K value in 50-50.



the performance measure (accuracy) against the K value in 70-30.

```
alpha: 0.8
   param_n_neighbors  mean_test_score  mean_train_score
0                  1            0.910           1.00000
1                  3            0.810           0.92000
2                  5            0.740           0.85375
3                  7            0.705           0.75250


alpha: 0.85
   param_n_neighbors  mean_test_score  mean_train_score
0                  1            0.910           1.00000
1                  3            0.825           0.93000
2                  5            0.740           0.83375
3                  7            0.710           0.74750


alpha: 0.9
   param_n_neighbors  mean_test_score  mean_train_score
0                  1            0.905           1.00000
1                  3            0.800           0.91875
2                  5            0.710           0.82250
3                  7            0.680           0.74000


alpha: 0.95
   param_n_neighbors  mean_test_score  mean_train_score
0                  1            0.905           1.00000
1                  3            0.790           0.91625
2                  5            0.710           0.80250
3                  7            0.670           0.72750
```

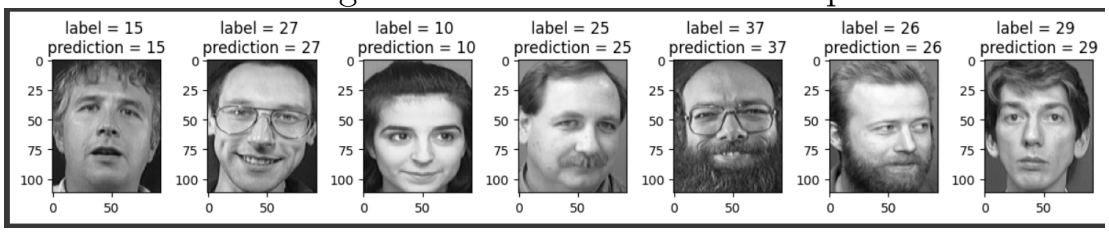
50-50 split accuracies

```
alpha: 0.8
   param_n_neighbors  mean_test_score  mean_train_score
0                  1         0.950000          1.000000
1                  3         0.907143          0.964286
2                  5         0.857143          0.922321
3                  7         0.789286          0.875000


alpha: 0.85
   param_n_neighbors  mean_test_score  mean_train_score
0                  1         0.950000          1.000000
1                  3         0.903571          0.968750
2                  5         0.853571          0.913393
3                  7         0.778571          0.869643


alpha: 0.9
   param_n_neighbors  mean_test_score  mean_train_score
0                  1         0.960714          1.000000
1                  3         0.885714          0.964286
2                  5         0.846429          0.905357
3                  7         0.771429          0.859821


alpha: 0.95
   param_n_neighbors  mean_test_score  mean_train_score
0                  1         0.957143          1.000000
1                  3         0.875000          0.961607
2                  5         0.821429          0.903571
3                  7         0.778571          0.852679
```

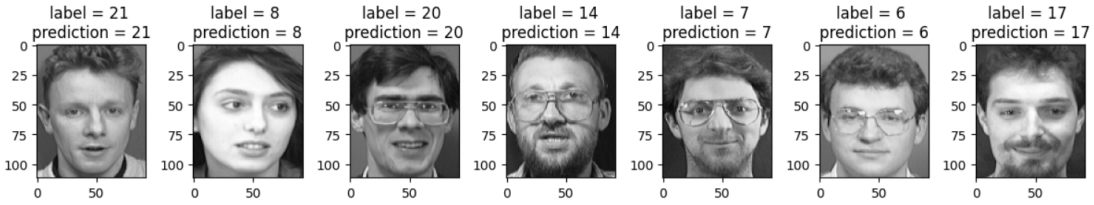
70-30 split accuracies

5 Test Stage

On the test stage we printed out some random images to test our model with prediction label vs the true one , here is some of these images in both 50-50 and 70-30 splits :



50-50 split test images



70-30 split test images

6 Conclusion

So, to summarize our work in the first approach, Eigenfaces along with a simple KNN Classifier did a very good job in faces classification as well as face reconstruction experiment conducted on a sample test face.

7 Report Link

Here you can view our latex report :
latex report link