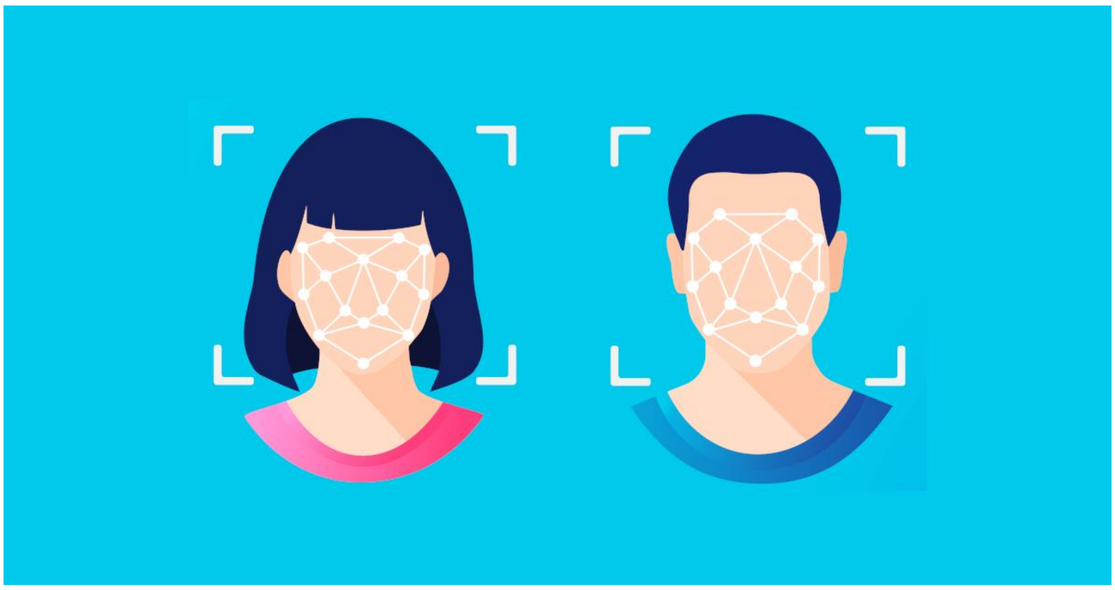


Assignment 1

Facial recognition

Hazem Mohammed Abdullah	6723
Ahmed Ashraf Abdelkarim Hussein	6940
Amr Abdel Samee Yousef	7126

Loading the data

The data is uploaded on google drive and then called into the notebook and loaded into numpy arrays

Data is then split 50% testing and 50% training and reshaped inside the numpy array

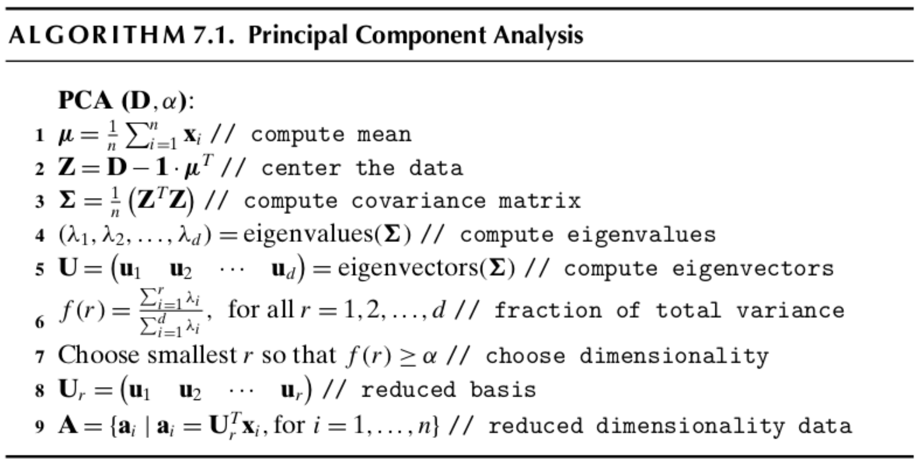
The same method is repeated for the non faces data which used the Cifar-10 dataset that contains 10k images of cars, planes, birds, animals, etc.…

A label is appended in another array to determine the class of each image

Data examples:

Classification using PCA

Principal component analysis - (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components and refrred 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. A nearest neighbour classifier works afterwards by comparing compare a face’s position in Eigenfaces subspace with the position of known individuals and the distances between them.



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 $\boldsymbol{\Sigma} = \frac{1}{n}(\mathbf{Z}^T\mathbf{Z})$ // 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.

We used `alphas` with these values `[0.8, 0.85, 0.9, 0.95]` to get the projected dimensions as following

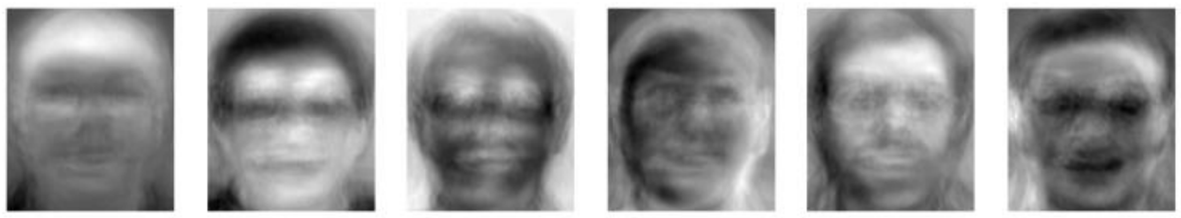
```
alphas_dim : [36, 51, 76, 115]

proj_mat0: (10304, 36) ,  proj_mat2: (10304, 76)

proj_mat1: (10304, 51) ,  proj_mat3: (10304, 115)
```

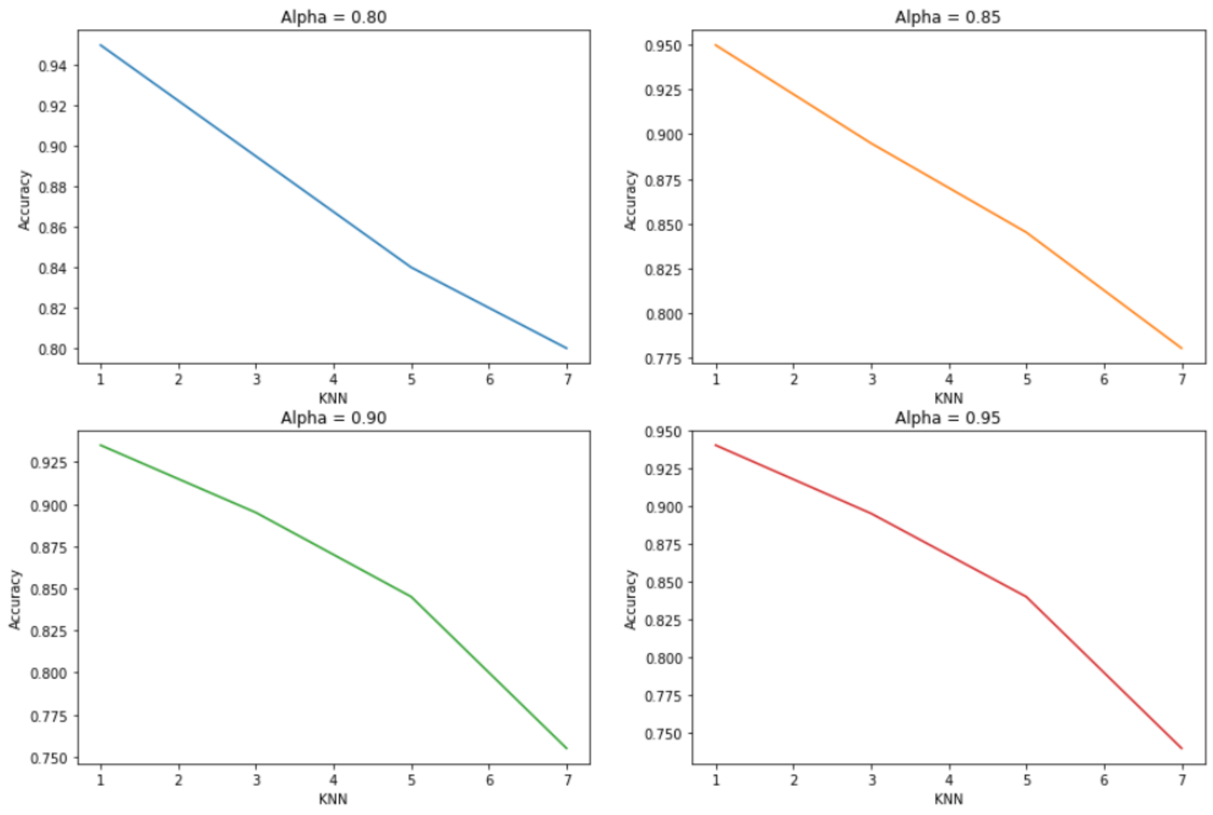
Visualization of Eigenfaces

Visualization of the biggest 6 eigen vectors



KNN

We used KNN to classify the data and the results are:



Faces Vs Nonfaces

We used CIFAR-10 dataset to get non faces images. We used 4000 image and splited them as following:

200 image: for testing (fixed upon all testing)

3000 image: for training

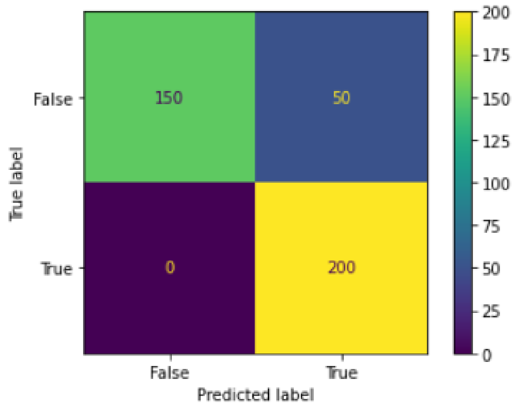
We started with 50 non face image and 200 face image we didnot change the number of faces images duuring the training. We changed the non faces images staring from 50 to 3000. [50, 200, 500, 1000, 3000]

Using this training data with the fixed test data (200 images) we got the following results:

50 non face image:

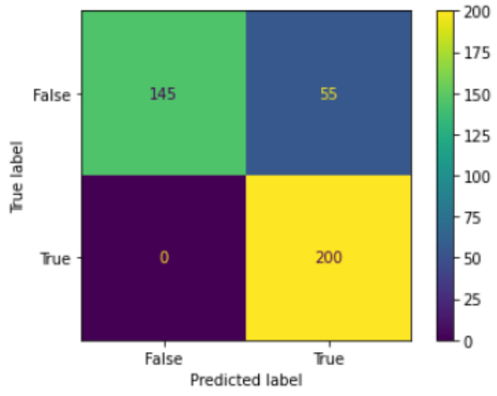
1) With alpha = 0.8

```
0.875
```

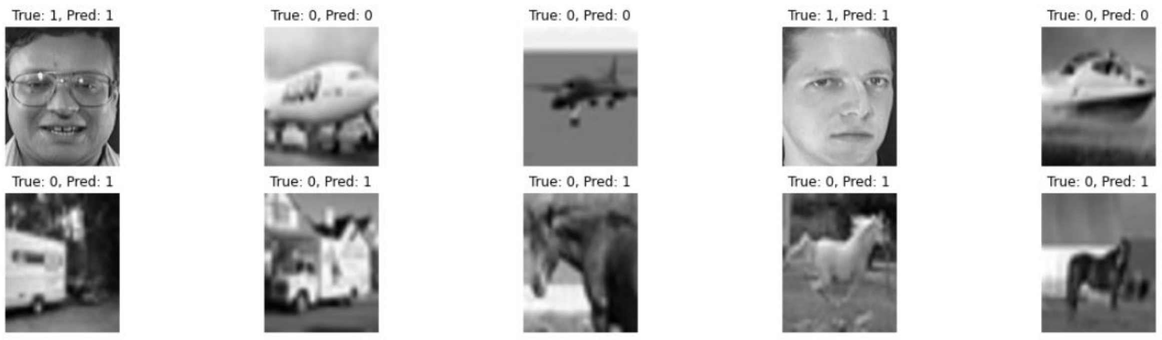


2) With alpha = 0.95

```
0.8625
```



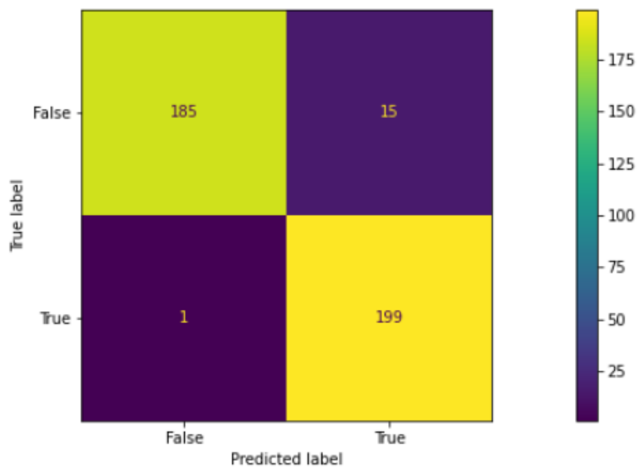
Test cases:



500 non face image:

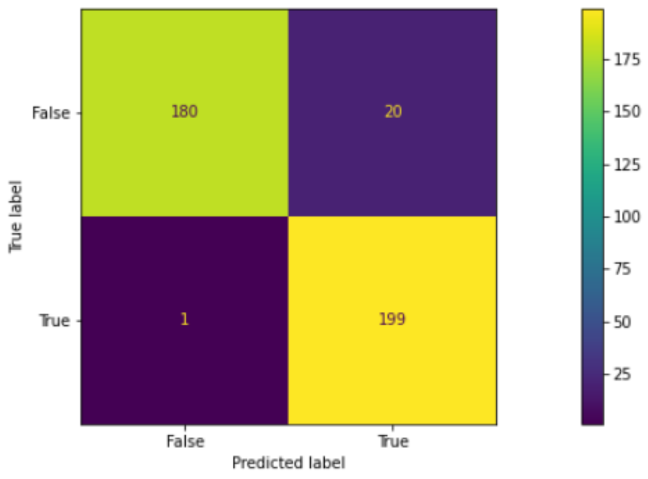
1) With alpha = 0.8 and 50 non face image

```
0.96
```

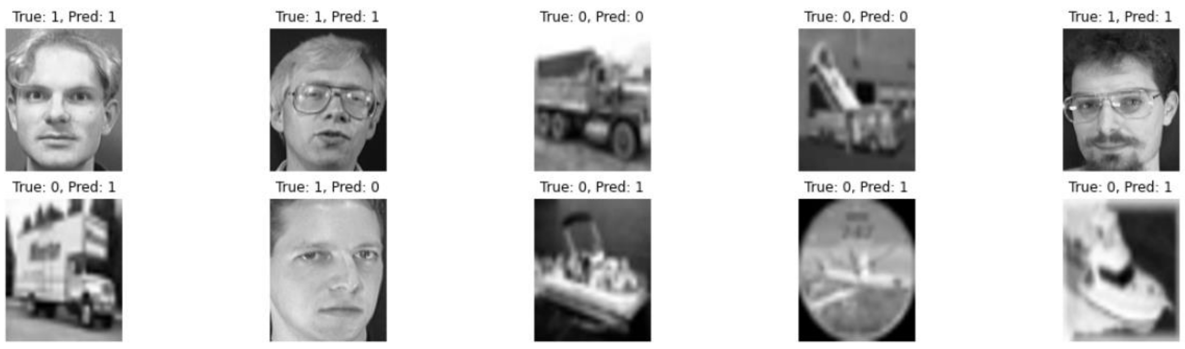


2) With alpha = 0.95 and 50 non face image

```
0.9475
```



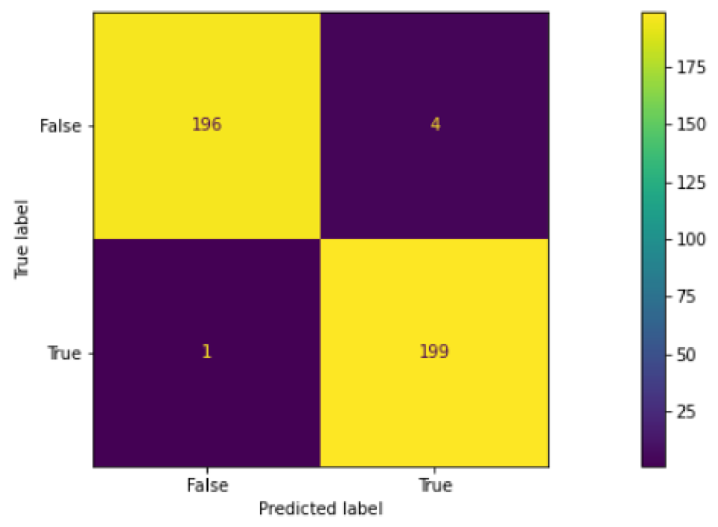
Test cases:



3000 non face image:

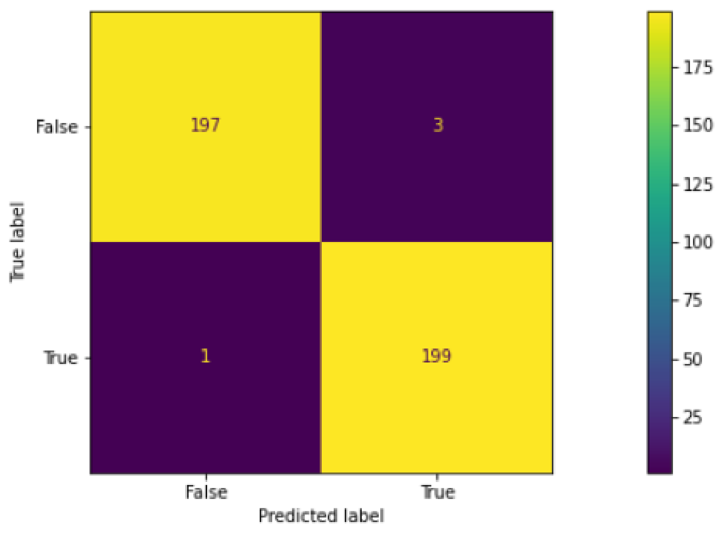
1) With alpha = 0.8 and 50 non face image

```
0.9875
```

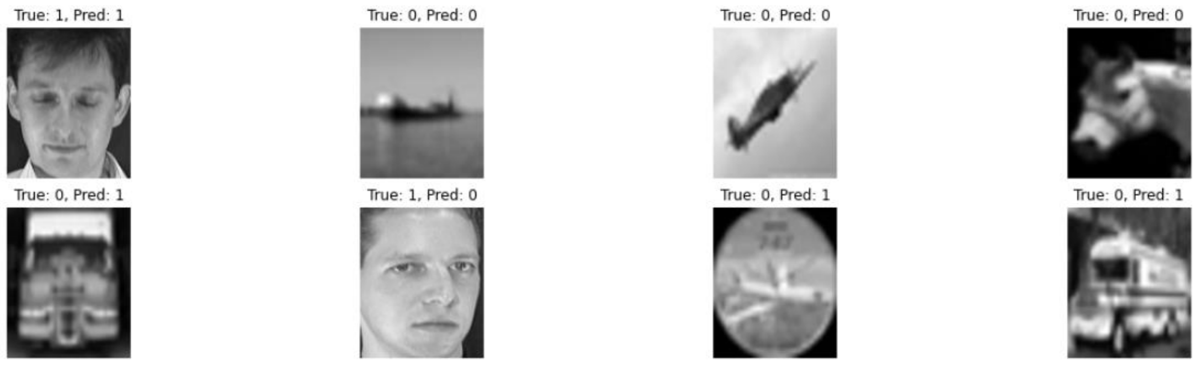


2) With alpha = 0.95 and 50 non face image

```
0.99
```

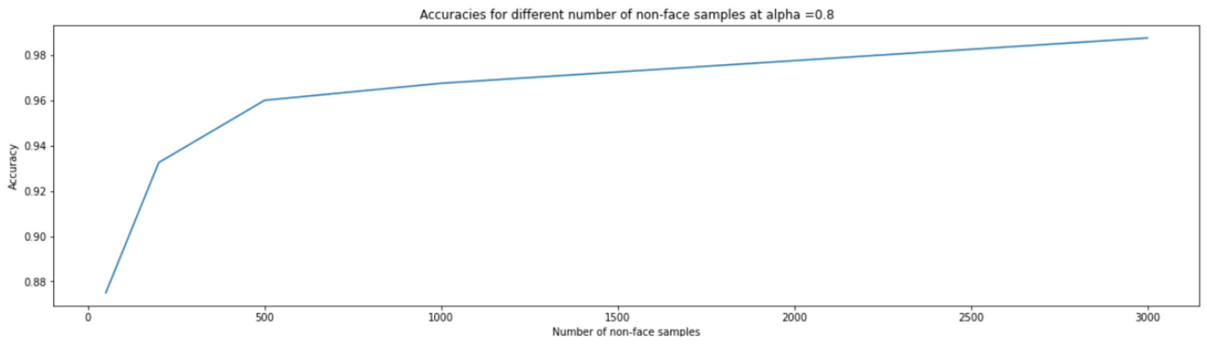


Test cases:

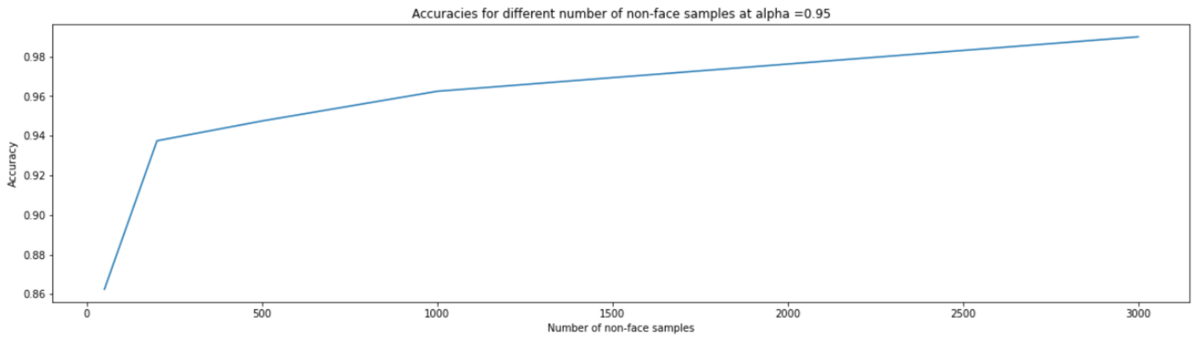


Comparison between all data splits:

1) Alpha = 0.8



2) Alpha = 0.95



LDA

Linear discriminant analysis (LDA), normal discriminant analysis (NDA), or discriminant function analysis is a generalisation of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterises or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.

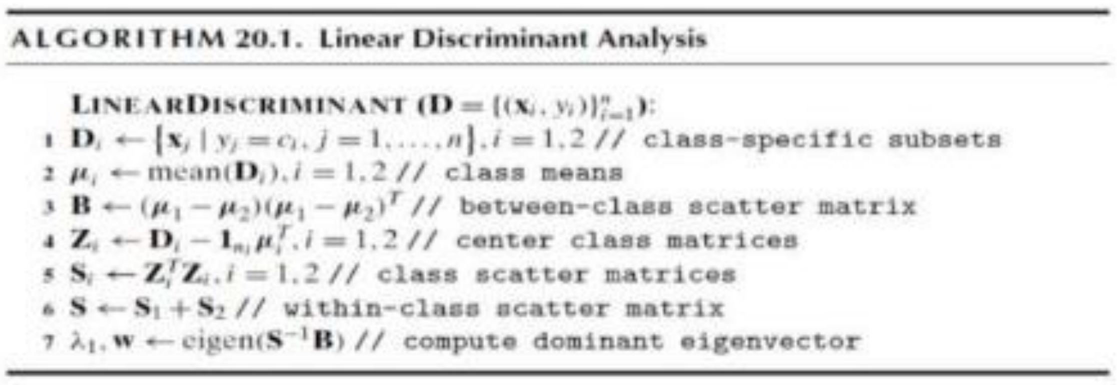
LDA is closely related to analysis of variance (ANOVA) and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements. However, ANOVA uses categorical independent variables and a continuous dependent variable, whereas discriminant analysis has continuous independent variables and a categorical dependent variable (*i.e.* the class label).Logistic regression and probit regression are more similar to LDA than ANOVA is, as they also explain a categorical variable by the values of continuous independent variables. These other methods are preferable in applications where it is not reasonable to assume that the independent variables are normally distributed, which is a fundamental assumption of the LDA method.

LDA is also closely related to principal component analysis (PCA) and factor analysis in that they both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data. PCA, in contrast, does not take into account any difference in class, and factor analysis builds the feature combinations based on differences rather than similarities. Discriminant analysis is also different from factor analysis in that it is not an interdependence technique: a distinction between independent variables and dependent variables (also called criterion variables) must be made.

LDA works when the measurements made on independent variables for each observation are continuous quantities. When dealing with categorical independent variables, the equivalent technique is discriminant correspondence analysis.

Discriminant analysis is used when groups are known a priori (unlike in cluster analysis). Each case must have a score on one or more quantitative predictor measures, and a score on a group measure. In simple terms, discriminant function analysis is classification - the act of distributing things into groups, classes or categories of the same type.

LDA that was implemented in this assignment for multiclass using this algorithms but with different variations



ALGORITHM 20.1. Linear Discriminant Analysis

```
LINEARDISCRIMINANT ($\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$):
1 $\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n\}, i = 1,2$ // class-specific subsets
2 $\boldsymbol{\mu}_i \leftarrow \text{mean}(\mathbf{D}_i), i = 1,2$ // class means
3 $\mathbf{B} \leftarrow (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ // between-class scatter matrix
4 $\mathbf{Z}_i \leftarrow \mathbf{D}_i - \mathbf{1}_{n_i}\boldsymbol{\mu}_i^T, i = 1,2$ // center class matrices
5 $\mathbf{S}_i \leftarrow \mathbf{Z}_i^T \mathbf{Z}_i, i = 1,2$ // class scatter matrices
6 $\mathbf{S} \leftarrow \mathbf{S}_1 + \mathbf{S}_2$ // within-class scatter matrix
7 $\lambda_1, \mathbf{w} \leftarrow \text{eigen}(\mathbf{S}^{-1}\mathbf{B})$ // compute dominant eigenvector
```

These variations are :

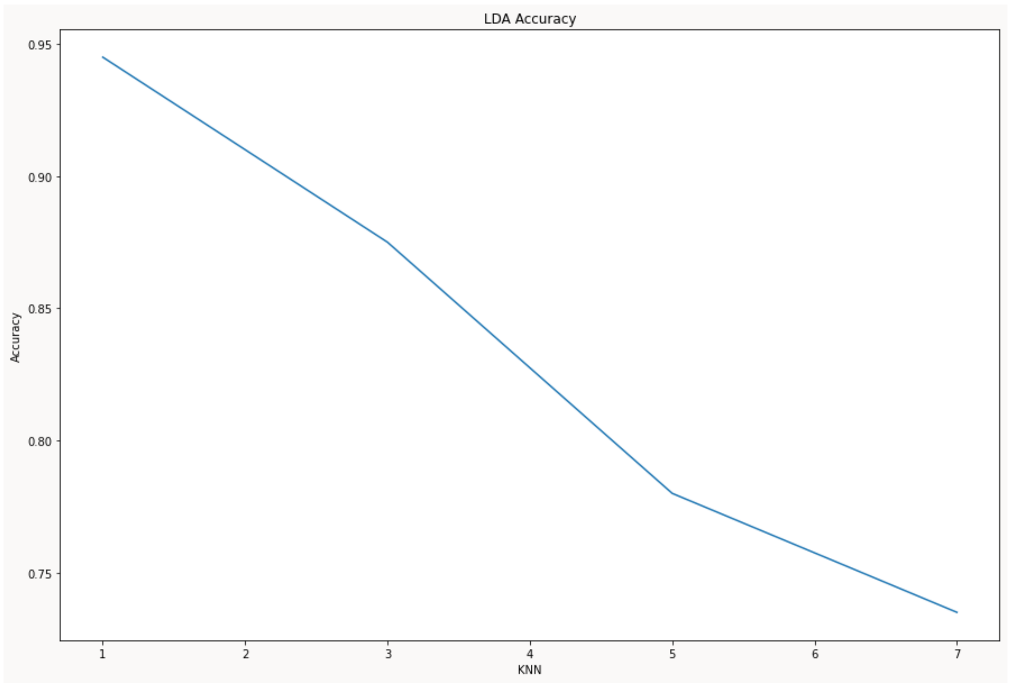
- Calculate the mean vector for every class Mu1, Mu2, ..., Mu40.
- Replace B matrix by S

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

- S matrix sums S1, S2, S3, ...S40.
- Taking the eigenvectors of the largest 39 eigenvalues and concatenating them together and then using np.dot() function to multiply the resulting eigenvectors with the data to get the new data

KNN will then run for different values of neigbours on the test data and the accuracy is then measured.

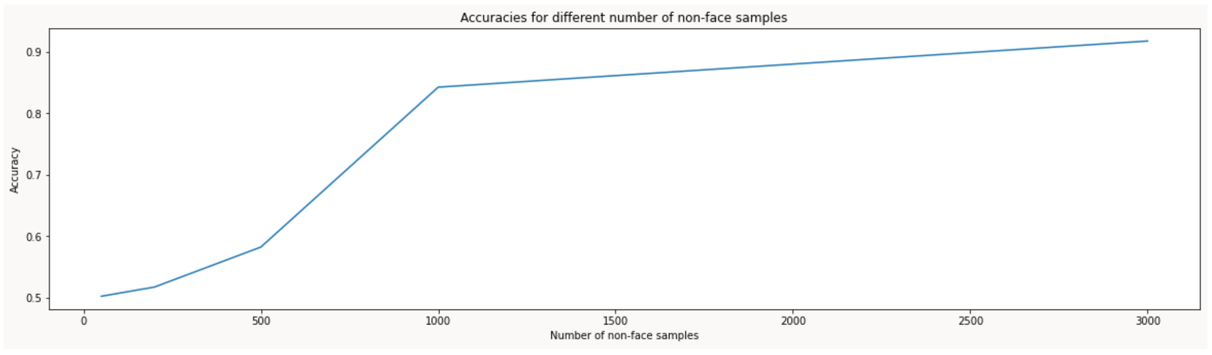
The results are:



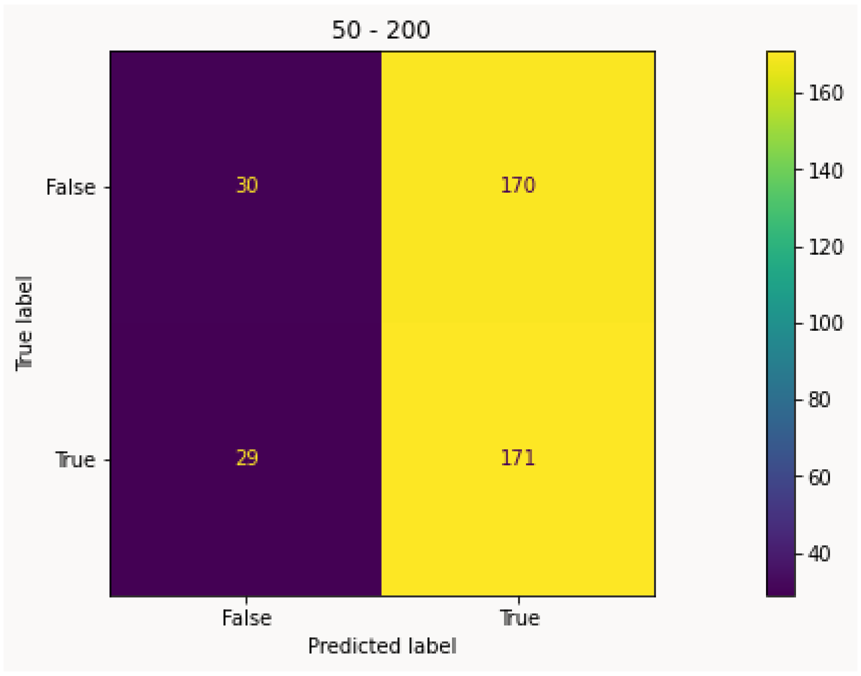
LDA in faces vs non faces

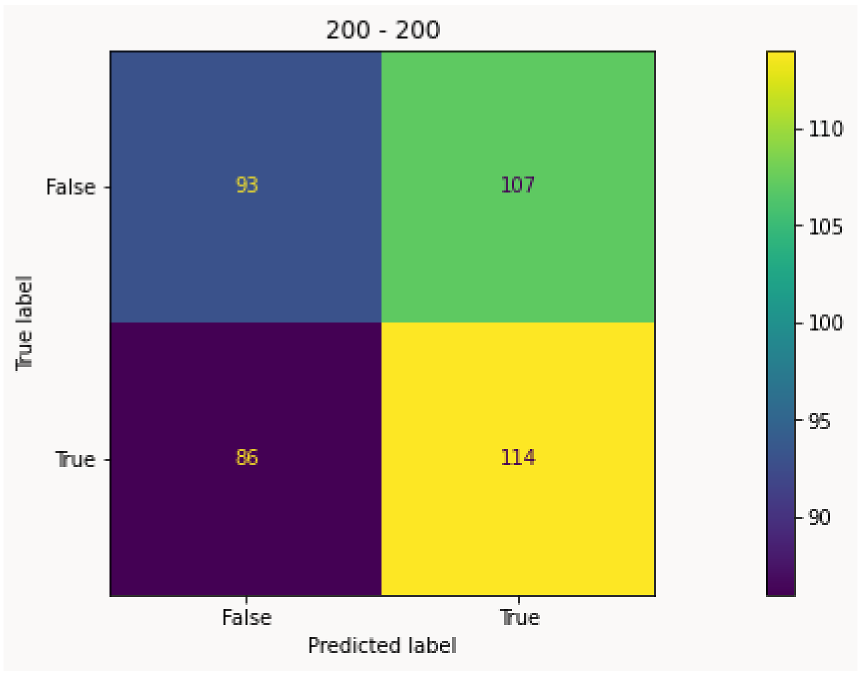
It will follow the same algorithm as in the picture exactly with only 1 eigenvector which corresponds to the highest eigenvalue

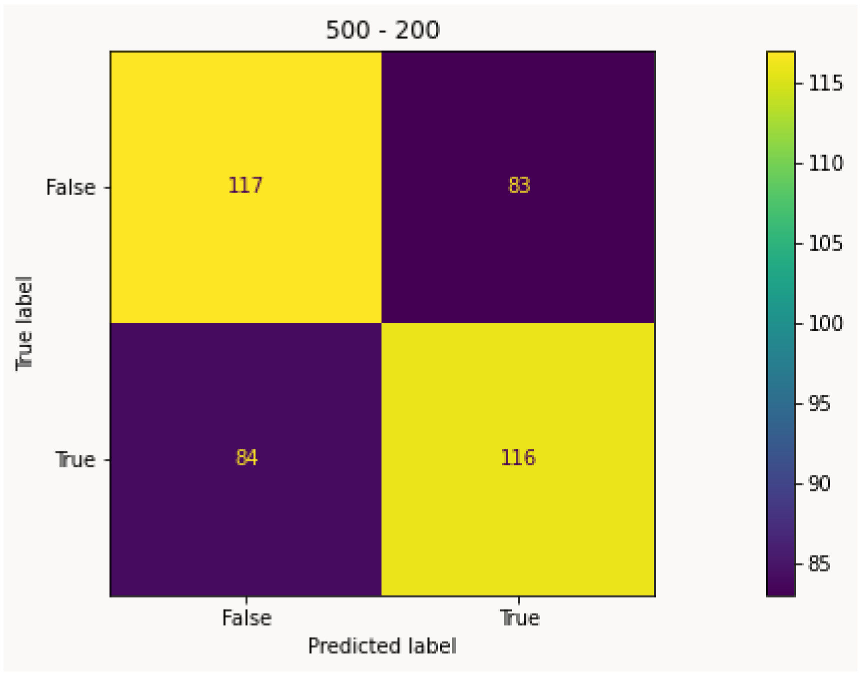
We tested using different number of non-faces samples starting from 50 to 3000 and then the accuracy is measured for each one of them while fixing the KNN value

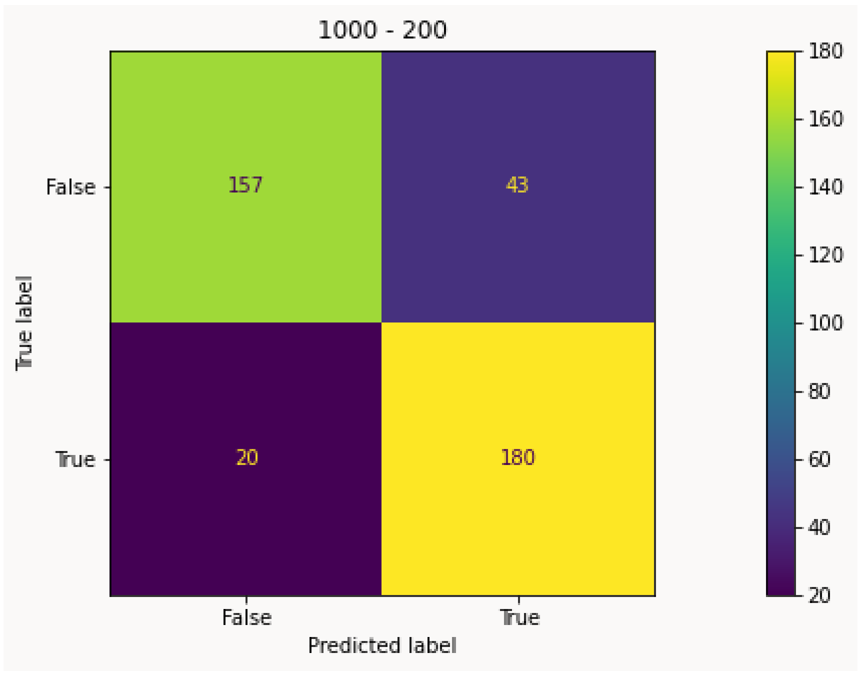


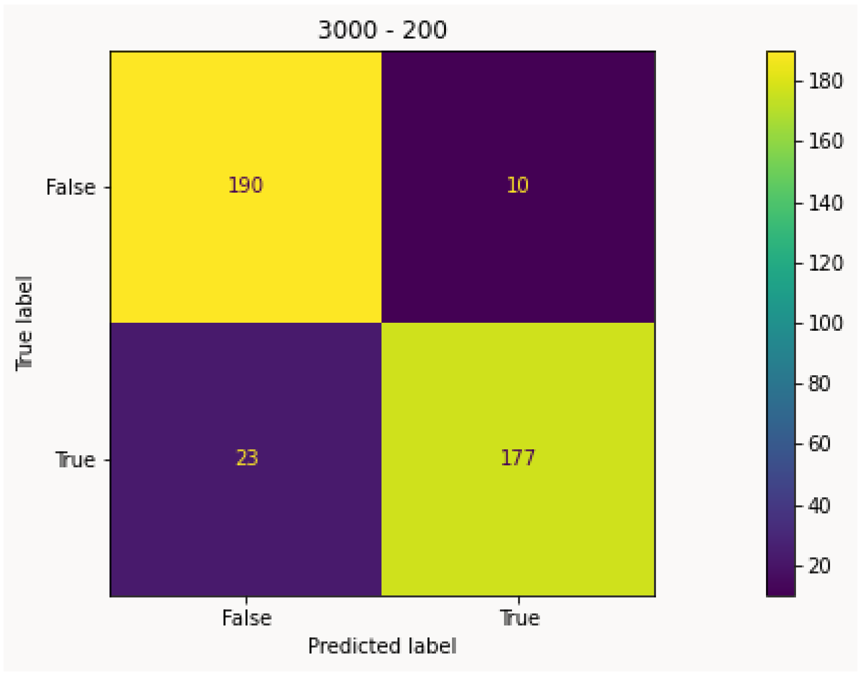
And below is the confusion matrix of the LDA for different values of non-faces


50 - 200
False
True
30
170
29
171
True label
Predicted label
160
140
120
100
80
60
40


200 - 200
False
True
93
107
86
114
True label
Predicted label
110
105
100
95
90

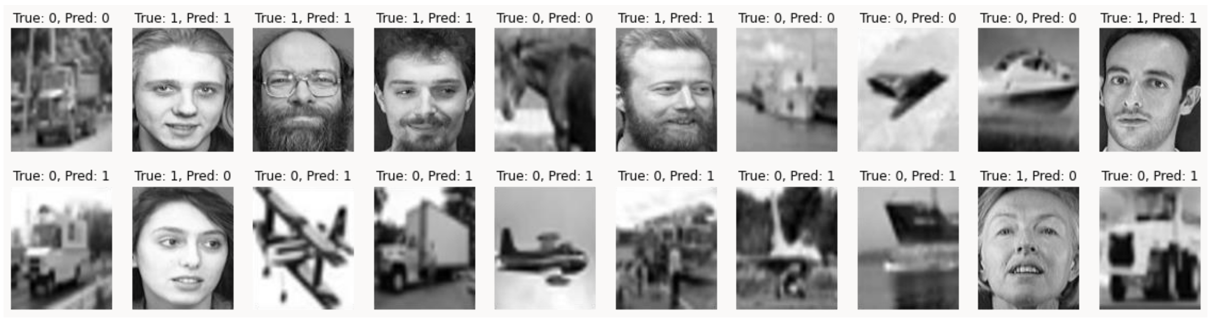

500 - 200
True label
Predicted label
False
True
False
True
117
83
84
116
115
110
105
100
95
90
85


1000 - 200
True label
Predicted label
False
True
False
True
157
43
20
180
180
160
140
120
100
80
60
40
20



Also there are some samples of data shown which are classified correctly or incorrectly

Example for this data:

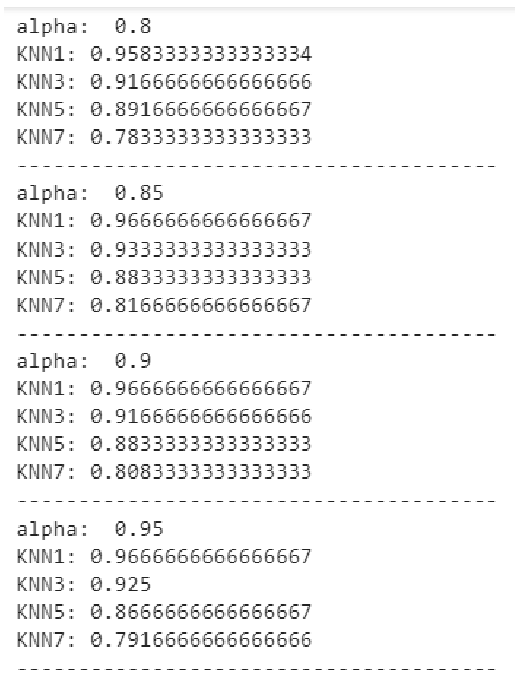


BONUS:

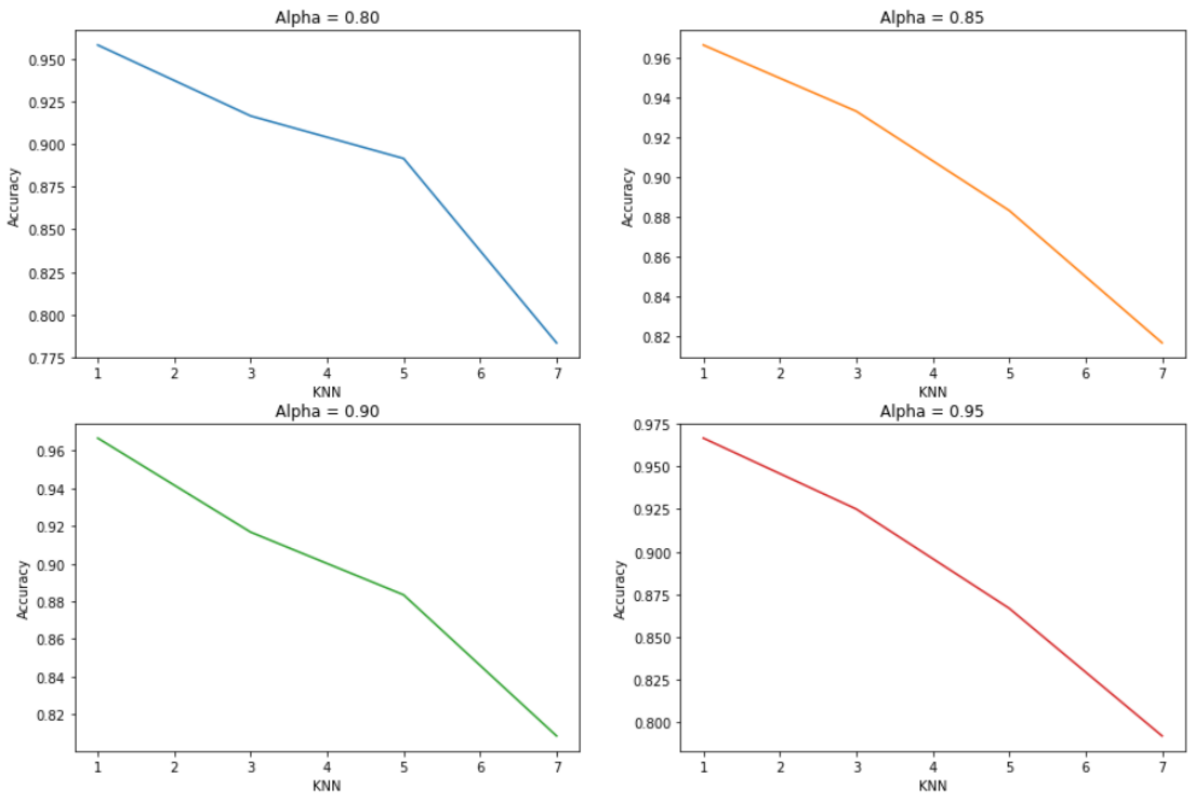
1. Split 30 - 70

We splited the data into 30% test and 70% train and applied the same PCA, LDA and KNN that we used in the 50 - 50 split we got the following results:

PCA:

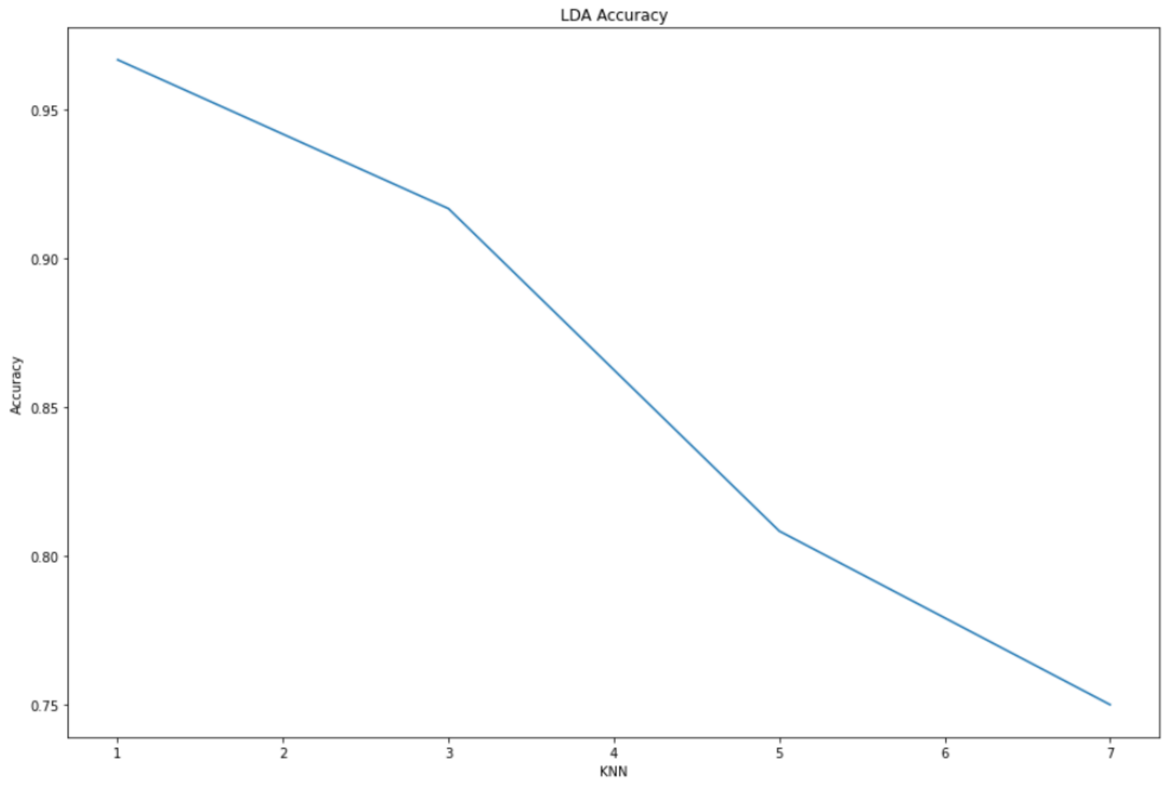


```
alpha:  0.8
KNN1: 0.9583333333333334
KNN3: 0.9166666666666666
KNN5: 0.8916666666666667
KNN7: 0.7833333333333333
---------------------------------------
alpha:  0.85
KNN1: 0.9666666666666667
KNN3: 0.9333333333333333
KNN5: 0.8833333333333333
KNN7: 0.8166666666666667
---------------------------------------
alpha:  0.9
KNN1: 0.9666666666666667
KNN3: 0.9166666666666666
KNN5: 0.8833333333333333
KNN7: 0.8083333333333333
---------------------------------------
alpha:  0.95
KNN1: 0.9666666666666667
KNN3: 0.925
KNN5: 0.8666666666666667
KNN7: 0.7916666666666666
---------------------------------------
```



LDA:

```
KNN:  1 Accuracy:  0.9666666666666667
KNN:  3 Accuracy:  0.9166666666666666
KNN:  5 Accuracy:  0.8083333333333333
KNN:  7 Accuracy:  0.75
```



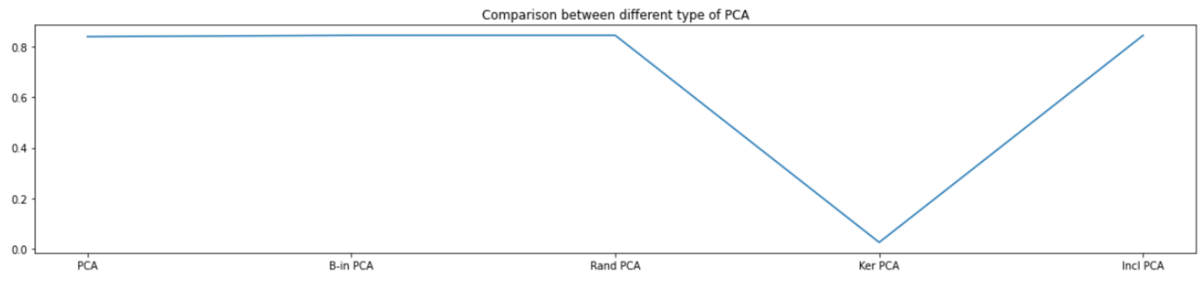
2. Variations

2.1. PCA variations

In this part we using 3 different variations for the PCA.

- Built-in PCA
- Randomized PCA
- Kernel PCA
- Incremental PCA

All of them got the same accuracy when tested on the same data used in our PCA which is ~ 84% except the kernel PCA we faced some erro in getting a Getting a high accuracy due to unknown reason the following is the plot of the accuracies of all of them.



2.2. LDA variations

In this part we tried alot of variations such as

- DA - Linear Discriminant Analysis
- FDA - Fisher's Discriminant Analysis
- QDA - Quadratic Discriminant Analysis
- KDA - Kernel Discriminant Analysis
- DLA - Diagonal Discriminant Analysis

But we failed to use them but the KDA which use the kernel trick in

Its algorithm.