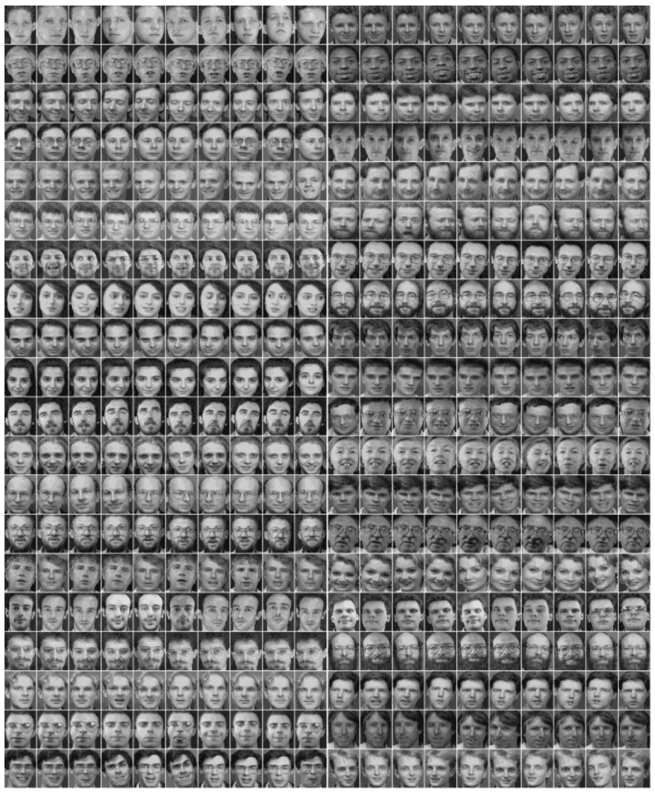
Pattern Recognition
Assignment 1

Face Recognition
using
PCA & LDA



Contributors:

Adham Amr Mohamed (6713)
Hossam Eldin Ahmed Shaaban (6721)
Saeed Elsayed Abokhatwa (6829)

1. Download the dataset & understand the format:

The AT&T Database Of Faces, (formerly 'The ORL Database of Faces'), contains a set of face images taken between April 1992 and April 1994.
There are ten different images of each of 40 distinct subjects.

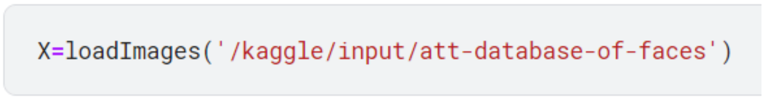
For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

The files are in PGM format. The size of each image is 92x112 pixels, with 256 grey levels per pixel. The images are organised in 40 directories (one for each subject), which have names of the form sX, where X indicates the subject number (between 1 and 40). In each of these directories, there are ten different images of that subject, which have names of the form Y.pgm, where Y is the image number for that subject (between 1 and 10).

The dataset was imported as input data to our Kaggle notebook to perform the required face recognition tasks.

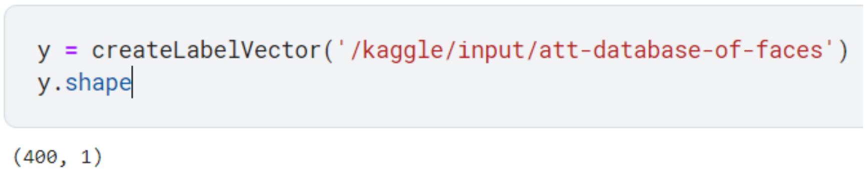
2. Generate the data matrix & the label vector:

a. To be able to operate on the data, the loadImages function was implemented. loadImages() converts every image in the dataset into a row vector of 10304 values corresponding to the image size, stacks the 400 row vectors into a single data matrix and returns the data matrix as a numpy array.



```
X=loadImages('/kaggle/input/att-database-of-faces')
```

b. To generate the label vector y, the createLabelVector() fn. was implemented. y consists of 400 rows: each row is a numpy array containing one integer value from 1:40; each value representing the subject id of an image. **This way, the integer in the *i*th row in y is the label of the *i*th row vector in the data matrix X.**

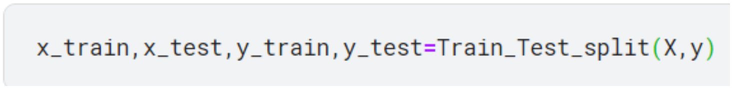


```
y = createLabelVector('/kaggle/input/att-database-of-faces')
y.shape
```

```
(400, 1)
```

3. Split the dataset into training & test sets:

The Train_Test_Split() function iterates over the data matrix X and the label vector y, assigns their even rows to training sets, their odd rows to test sets and returns the resulting matrices. This way, 50% of the data is used for training, and the other 50% is kept for testing.



```
x_train,x_test,y_train,y_test=Train_Test_split(X,y)
```

4. Classification Using PCA :

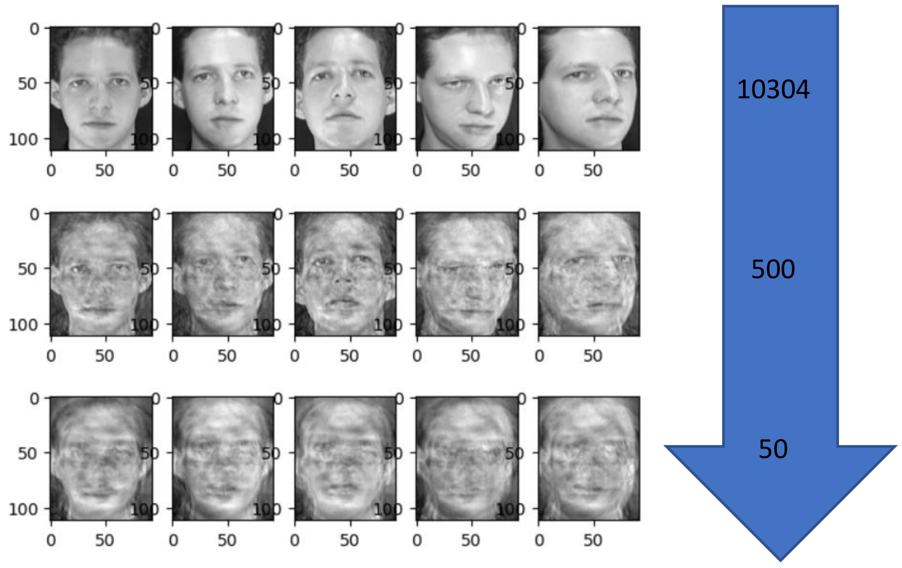
Points (a) through (e) :

Alpha	Number of eig vectors to consider	Accuracy (FNN)
0.8	36	0.97
0.85	52	0.975
0.9	76	0.97
0.95	116	0.965

Table 4.1

e) when alpha increases , the number of eig vector to consider will also increase however … considering more eig vectors will cause our model to overfit on the training data .. which makes the accuracy drop in the testing data so it’s wise to consider accuracy measure on testing data as a indicator of which value of alpha to consider.

4-2 demonstration of the data: eigen vectors to consider are on axis of the arrow



in figure 4-2.1 the less eigen vectors we consider the less visual the face becomes

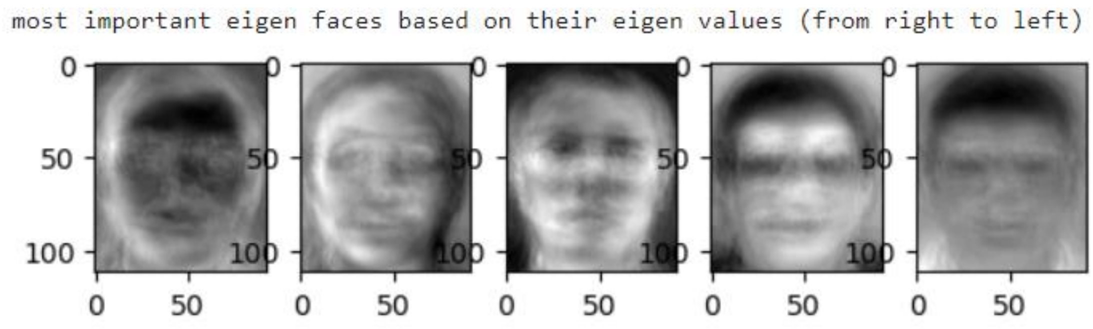

most important eigen faces based on their eigen values (from right to left)
0
50
100
0
50

Figure 4-2.1

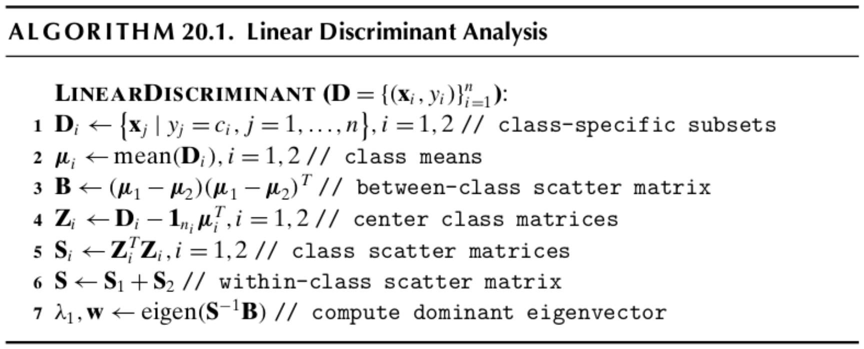
In figure 4-2.2 we can see the eigen faces where they contain the most important features of a face and we use them for PCA.

the projection and reconstruction of images is done by the code below :

```
def project(self,X):
    return X @ self.Ur
        #(200,10304) x (10304, r) = (200,r)
def reconstruct_images(self,x_reduced):
    return x_reduced @ self.Ur.T
        #(200,r) x (r, 10304) = (200,10304) to be able to view it
```

5. Classification using LDA:

5-a. Linear Discriminant Analysis (LDA) is a popular supervised learning technique used in statistics, pattern recognition, and machine learning. It is used to classify data into distinct categories based on a set of features. LDA seeks to find a linear combination of features that maximizes the separation between different categories while minimizing the variation within each category.
LDA is often used as a dimensionality reduction technique as well. By projecting the data onto a lower-dimensional subspace while maintaining the separability of the categories, LDA can help reduce the complexity of a classification problem and improve classification accuracy.
LDA assumes that the data follows a normal distribution and that the covariance matrix of the features is equal across all categories. LDA has been successfully applied to a wide range of applications, including face recognition, text classification, and bioinformatics.



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
```

Figure 5.1 LDA pseudocode

We will replace B matrix in Figure 5.1 line 3 by Sb given by the following formula:

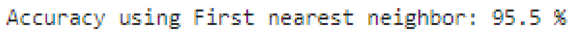
$$S_b = \sum_{k=1}^{m} n_k(\mu_k - \mu)(\mu_k - \mu)^T$$

Where n is the number of samples in each class, µk is the mean of class K and µ is the overall mean.

b. After computing W, we will use the dominant 39 eigenvectors.

c. After selecting the dominant 39 vector we will project our data on the 39 vectors to
   reduce our data from 200x10304 to 200x39.

d. Using a simple classifier like KNN we will get a 95.5% Accuracy using 1KNN.

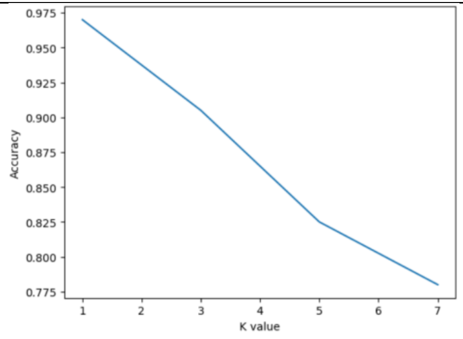
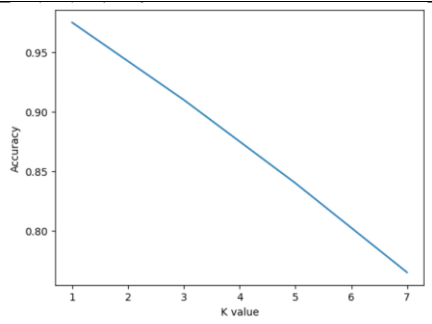
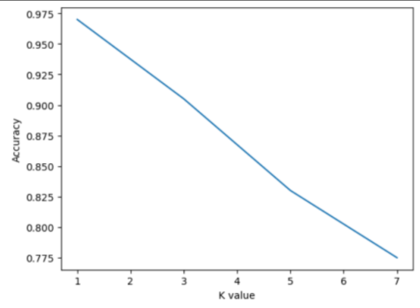
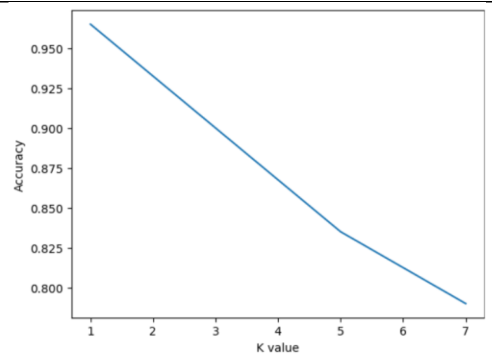


```
Accuracy using First nearest neighbor: 95.5 %
```

e. Comparing the results to PCA using 1st NN:
   As shown in table 4.1, using different alphas yields a higher accuracy than LDA.

6-Classifier Tuning:

6-1 PCA:

Alpha	Graph
0.8 [0.97, 0.905, 0.825, 0.78]	
0.85 [0.975, 0.91, 0.84, 0.765]	
0.9 [0.97, 0.905, 0.83, 0.775]	
0.95 [0.965, 0.9, 0.835, 0.79]	

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

6-2. LDA:

In this part we Set the number of neighbors in the K-NN classifier to 1,3,5,7.
Sklearn the K-NN classifier was used to classify and handle the tie breaking.
Figure 6.1 shows that accuracy drops while increasing K value.

.

```
[0.955, 0.885, 0.835, 0.795]
```

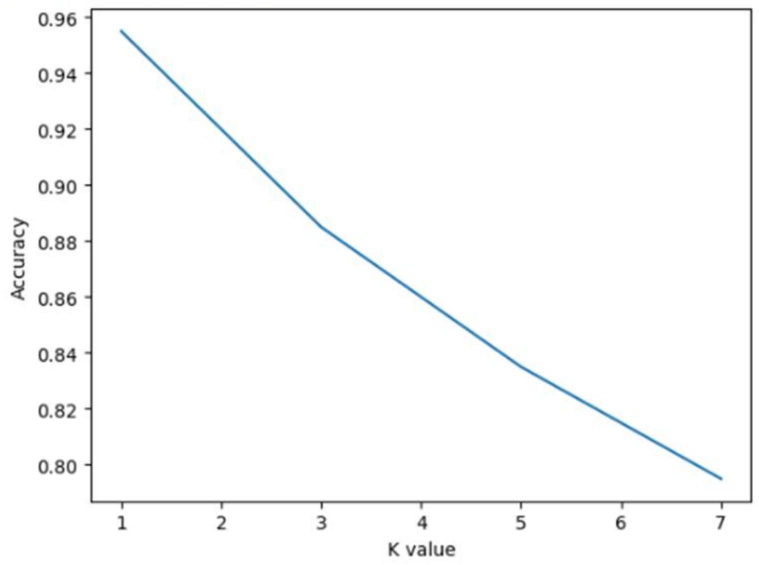


Figure 6.1: graph between the number of neighbors on x-axis and accuracy on y-axis.

As shown in figure 6.1, The accuracy drops while increasing the value of K.

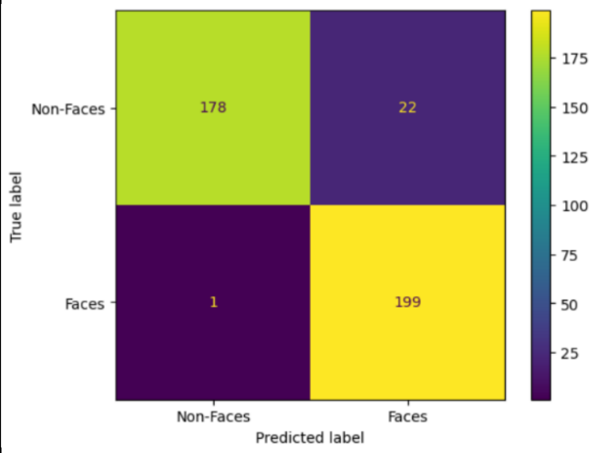
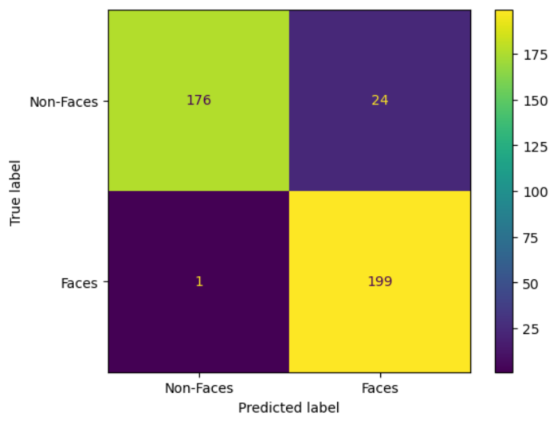
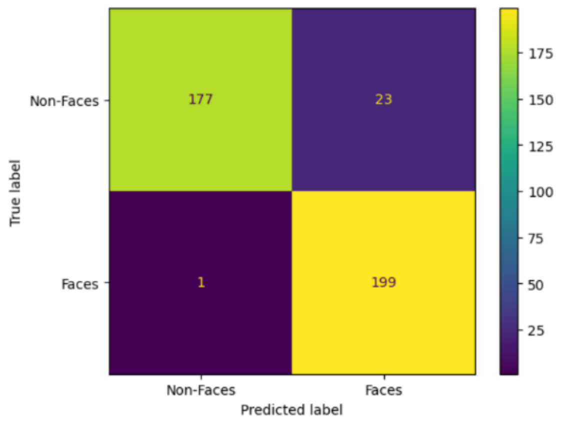
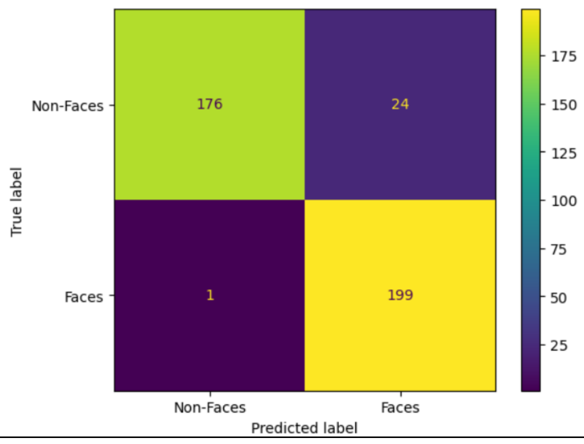
7. Compare faces vs. non-faces images:

7-1. PCA:

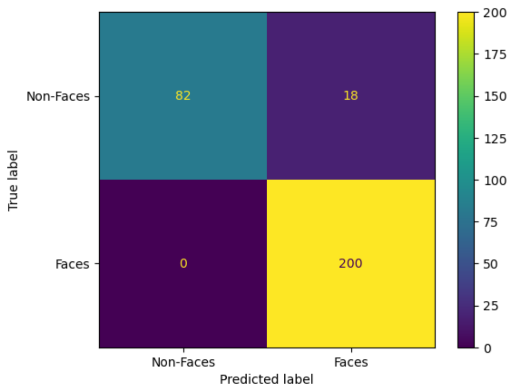
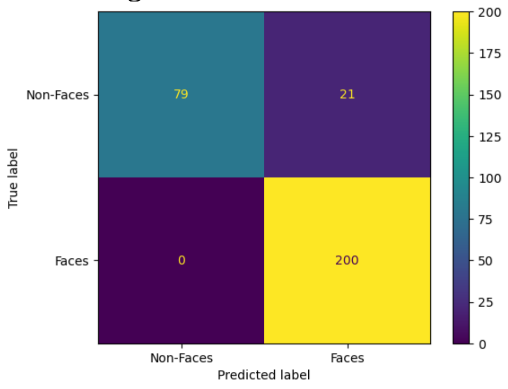
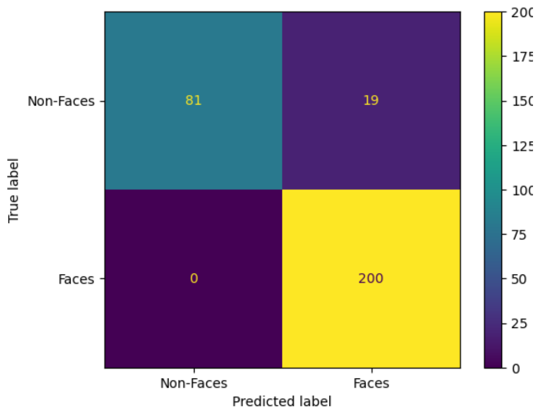
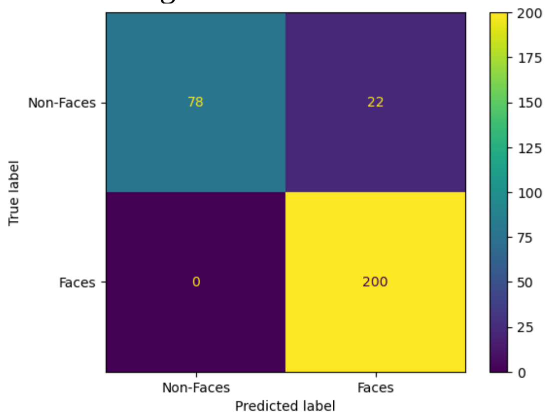
In addition to the 400 faces images, we added another 400 images consisting of shoes, furniture, pepsi and cocacola cans as non-faces images. Our new data matrix has 800 row vectors, each row consists of 10304 values corresponding to the image size. A new label vector was created, with 1 as the label for faces, and 0 as the label for non-faces. Following are the results for PCA using the 50% split:

Failure & success cases:

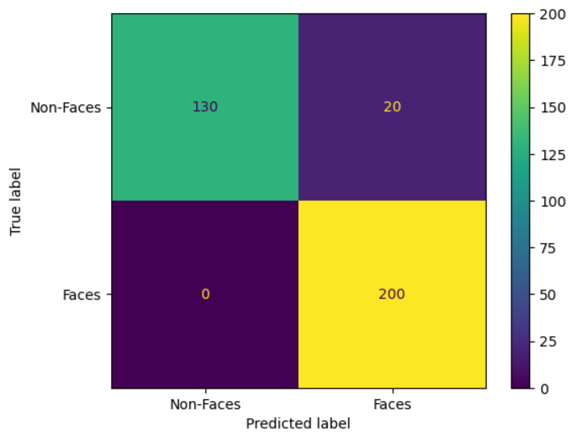
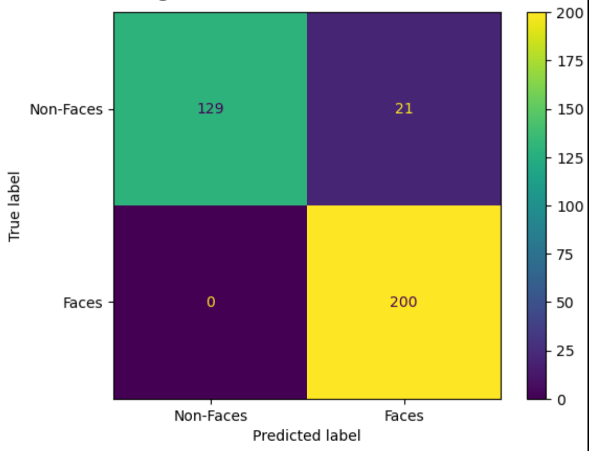
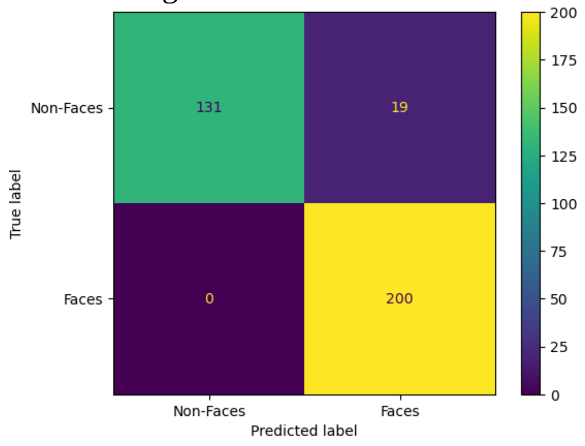
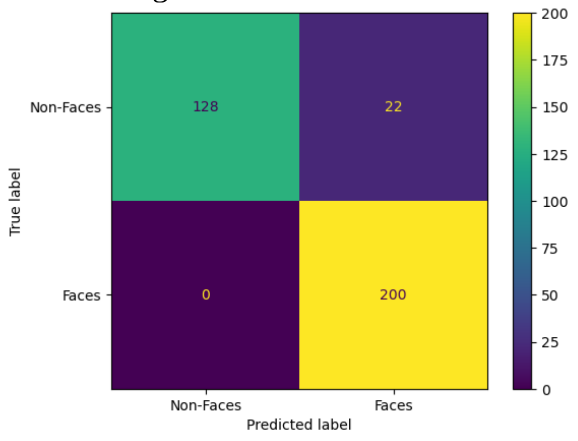
200 faces vs 200 non-faces:

Alpha = 0.8 # eigenvectors to consider = 18 	Alpha = 0.9 # eigenvectors to consider = 32 
Alpha = 0.85 # eigenvectors to consider = 58 	Alpha = 0.95 # eigenvectors to consider = 120 

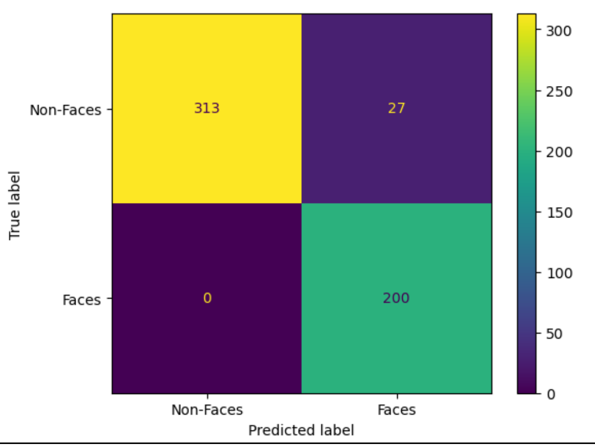
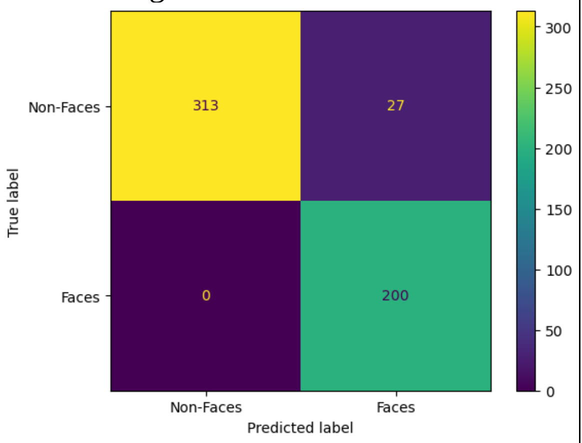
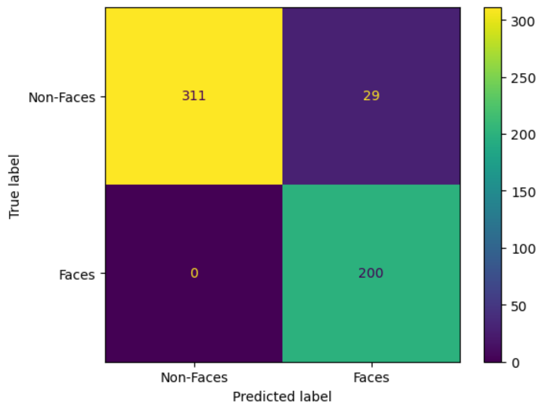
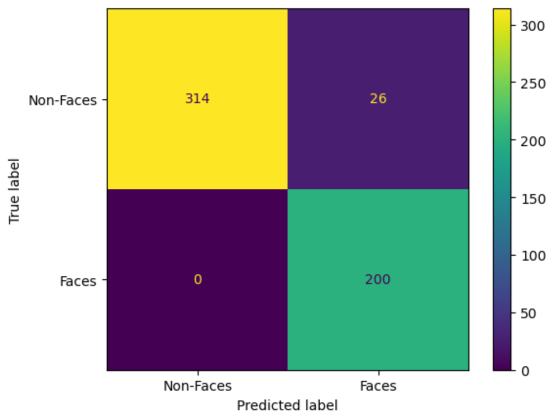
200 faces vs 100 non-faces:

Alpha = 0.8 14 eigenvectors to consider = 	Alpha = 0.9 45 eigenvectors to consider = 
Alpha = 0.85 25 eigenvectors to consider = 	Alpha = 0.95 93 eigenvectors to consider = 

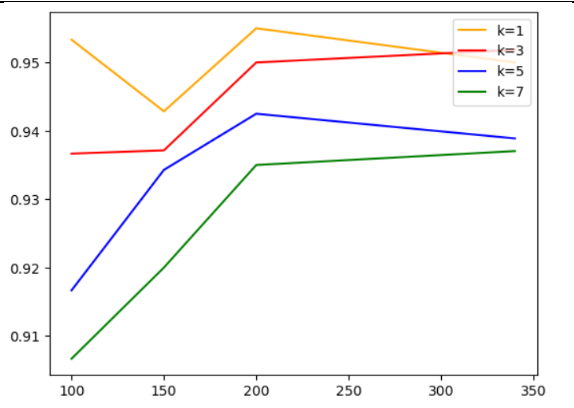
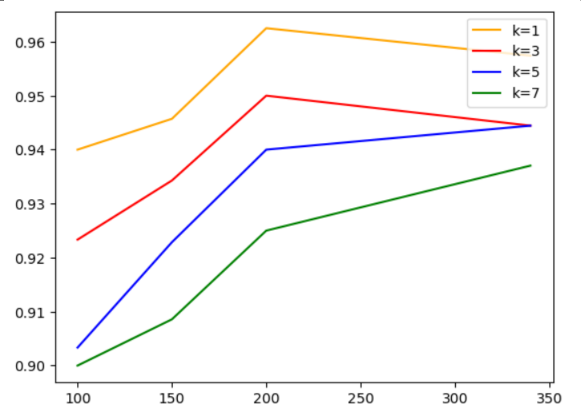
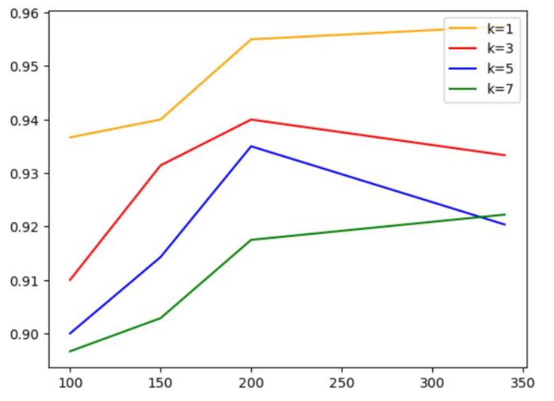
200 faces vs 150 non-faces:

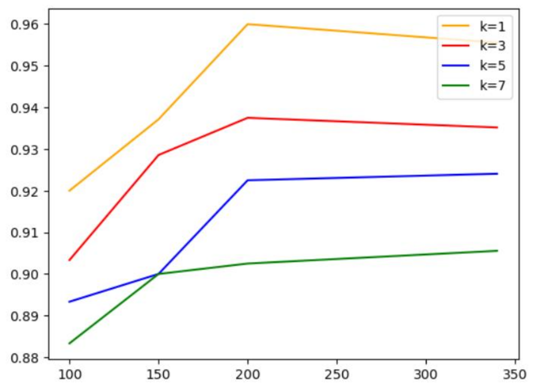
Alpha = 0.8 # eigenvectors to consider = 	Alpha = 0.9 # eigenvectors to consider = 
Alpha = 0.85 # eigenvectors to consider = 	Alpha = 0.95 # eigenvectors to consider = 

200 faces vs 340 non-faces:

Alpha = 0.8 # eigenvectors to consider = 	Alpha = 0.9 # eigenvectors to consider = 
Alpha = 0.85 # eigenvectors to consider = 	Alpha = 0.95 # eigenvectors to consider = 

Accuracy measures: best results are obtained when the data is balanced (200 F vs 200 NF)

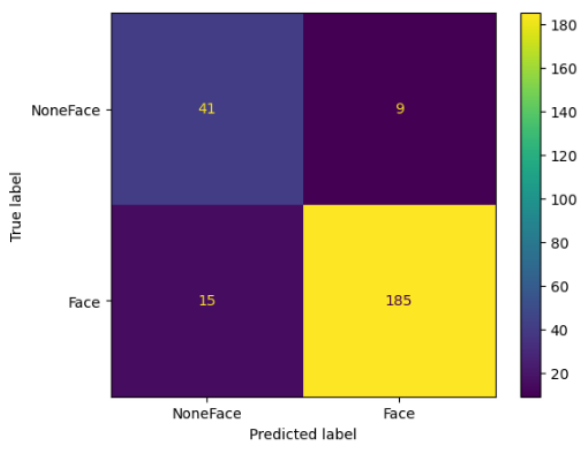
PCA alpha = 0.8	
PCA alpha = 0.85	
PCA alpha = 0.9	

PCA alpha = 0.95 Best results is when the data is balanced (200 faces vs 200 non-faces) Increasing or decreasing CNN	

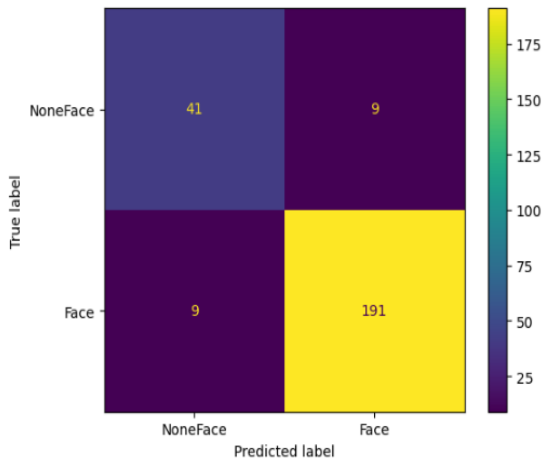
7-2. LDA:

Failure & success cases:

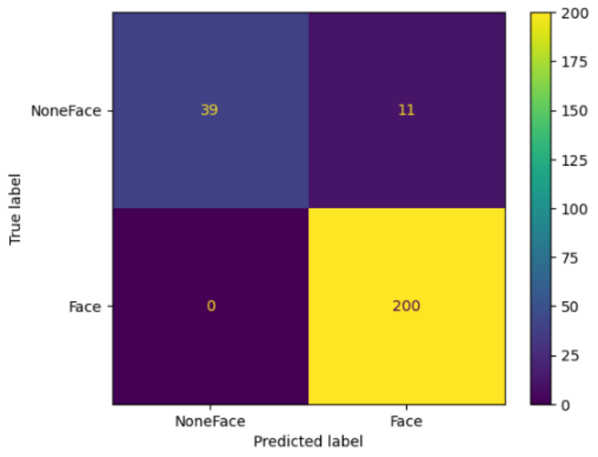
```
Using 400 Face and 100 None face
Accuracy for  1 NN =  0.904
Total number of samples: 250
Number of success cases: 226
Number of failure cases: 24
```



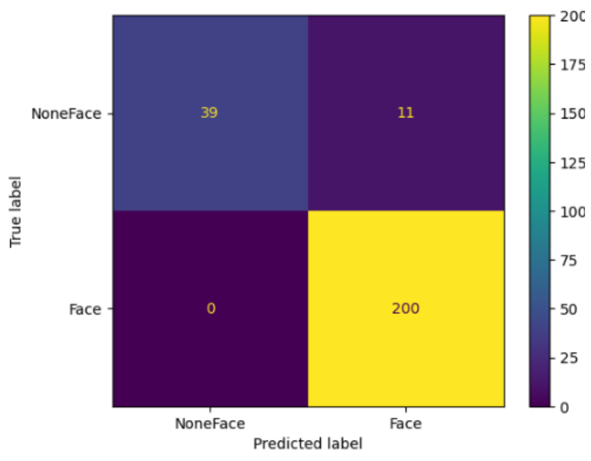
```
Accuracy for  3 NN =  0.928
Total number of samples: 250
Number of success cases: 232
Number of failure cases: 18
```



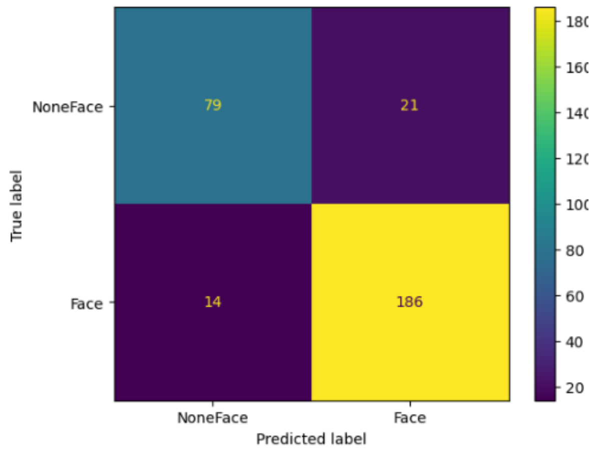
```
Accuracy for  5 NN =  0.956
Total number of samples: 250
Number of success cases: 239
Number of failure cases: 11
```



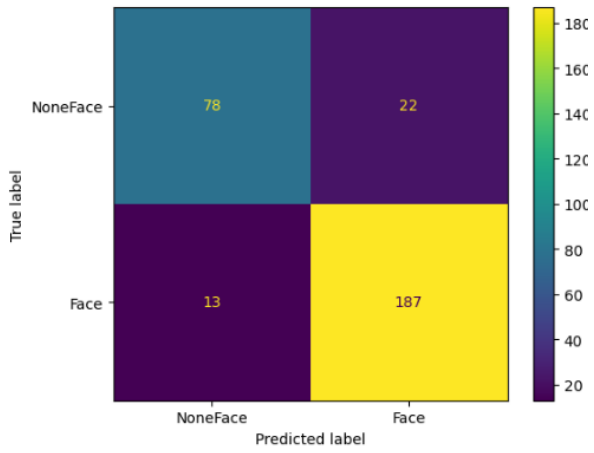
```
Accuracy for  7 NN =  0.956
Total number of samples: 250
Number of success cases: 239
Number of failure cases: 11
```



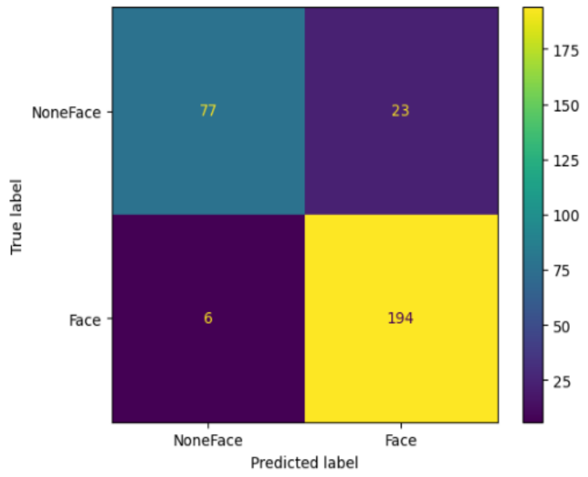
```
Using 400 Face and 200 None face
Accuracy for  1 NN =  0.8833333333333333
Total number of samples: 300
Number of success cases: 265
Number of failure cases: 35
```



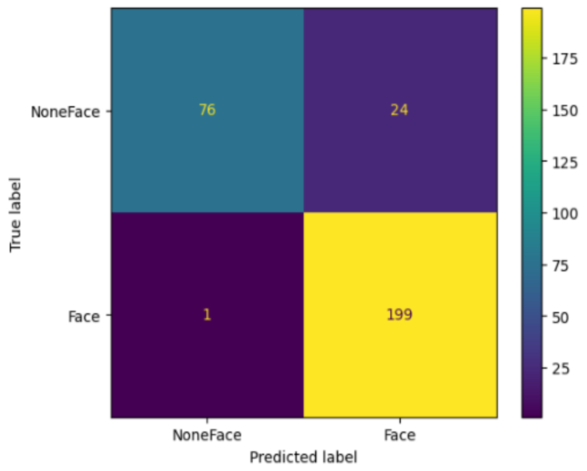
```
Accuracy for  3 NN =  0.8833333333333333
Total number of samples: 300
Number of success cases: 265
Number of failure cases: 35
```



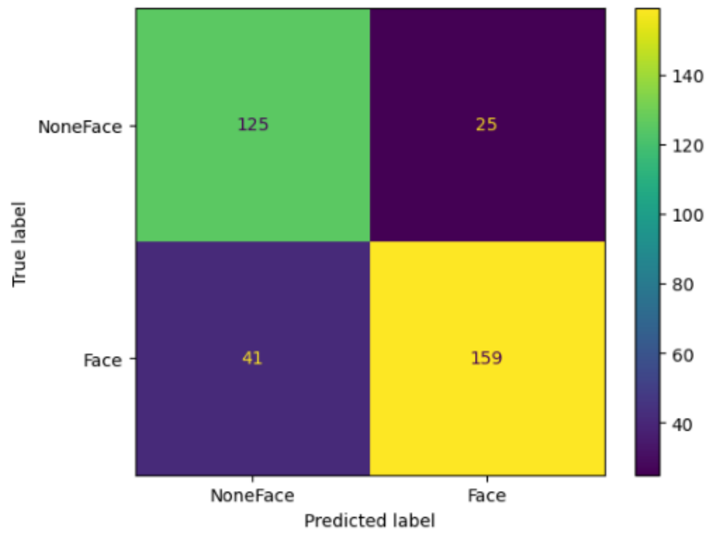
```
Accuracy for  5 NN =  0.9033333333333333
Total number of samples: 300
Number of success cases: 271
Number of failure cases: 29
```



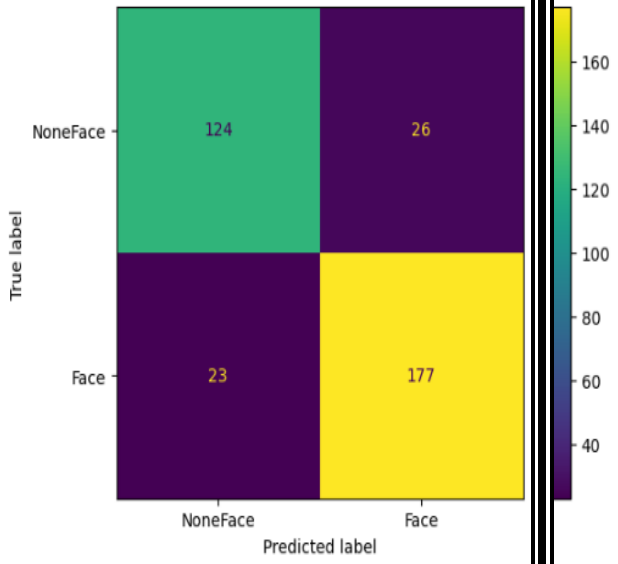
```
Accuracy for  7 NN =  0.9166666666666666
Total number of samples: 300
Number of success cases: 275
Number of failure cases: 25
```



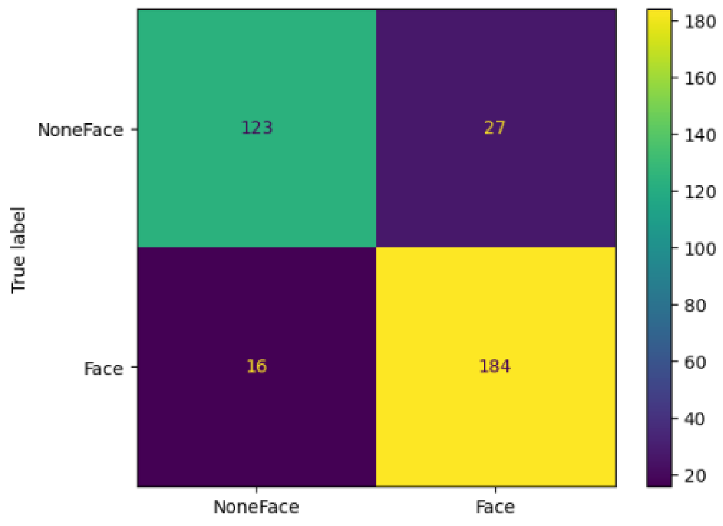
```
Using 400 Face and 300 None face
Accuracy for  1 NN =  0.8114285714285714
Total number of samples: 350
Number of success cases: 284
Number of failure cases: 66
```



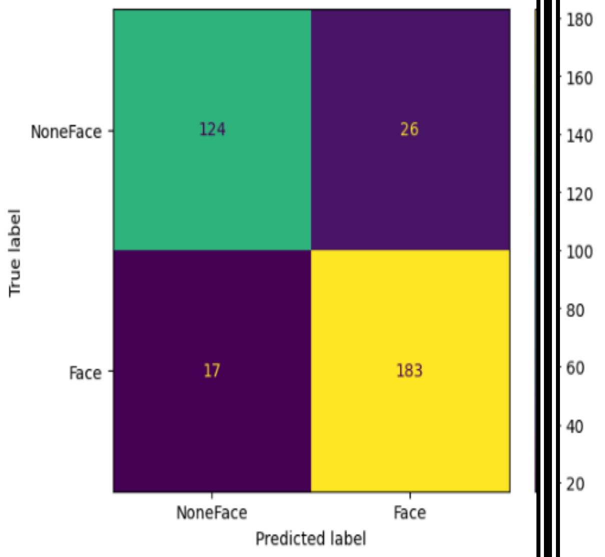
```
Accuracy for  3 NN =  0.86
Total number of samples: 350
Number of success cases: 301
Number of failure cases: 49
```

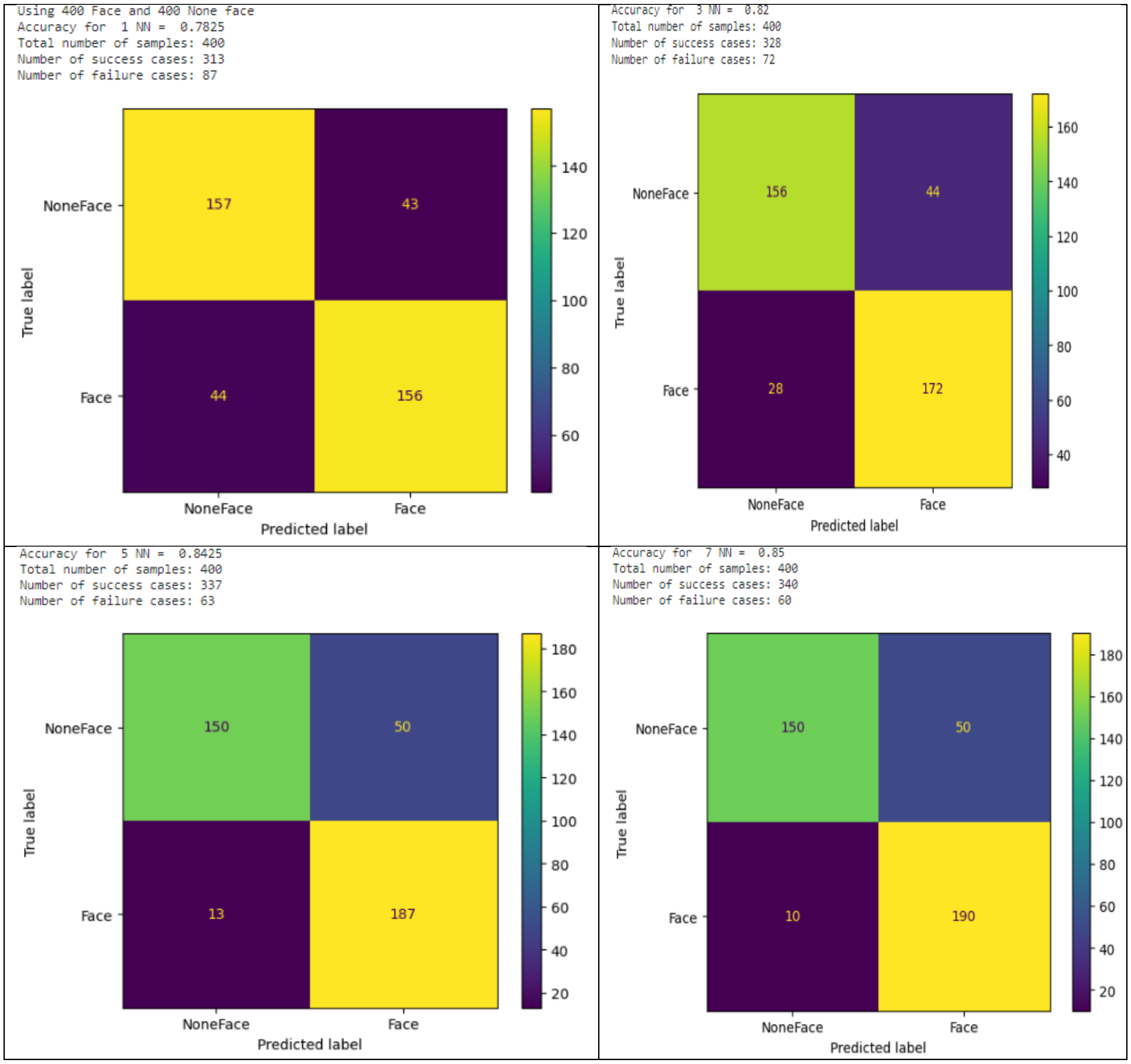


```
Accuracy for  5 NN =  0.8771428571428571
Total number of samples: 350
Number of success cases: 307
Number of failure cases: 43
```



```
Accuracy for  7 NN =  0.8771428571428571
Total number of samples: 350
Number of success cases: 307
Number of failure cases: 43
```





```
Using 400 Face and 400 None face
Accuracy for  1 NN =  0.7825
Total number of samples: 400
Number of success cases: 313
Number of failure cases: 87
```

```
Accuracy for  3 NN =  0.82
Total number of samples: 400
Number of success cases: 328
Number of failure cases: 72
```

```
Accuracy for  5 NN =  0.8425
Total number of samples: 400
Number of success cases: 337
Number of failure cases: 63
```

```
Accuracy for  7 NN =  0.85
Total number of samples: 400
Number of success cases: 340
Number of failure cases: 60
```

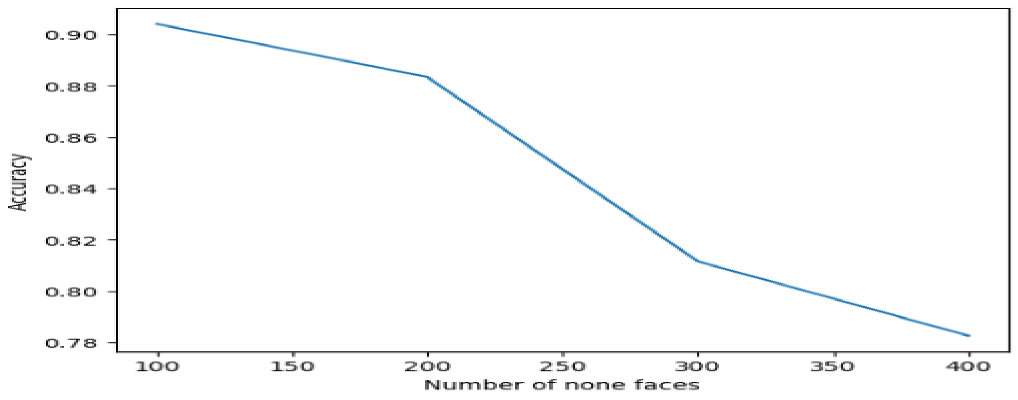


Figure 7.1

As shown in Figure 7.1, The accuracy drops while increasing the number of none faces in the training set.

We chose 1 dominant vector because number of components (eigenvectors) must be less than or equal to min(numberofclass-1,numberOfFeatures).

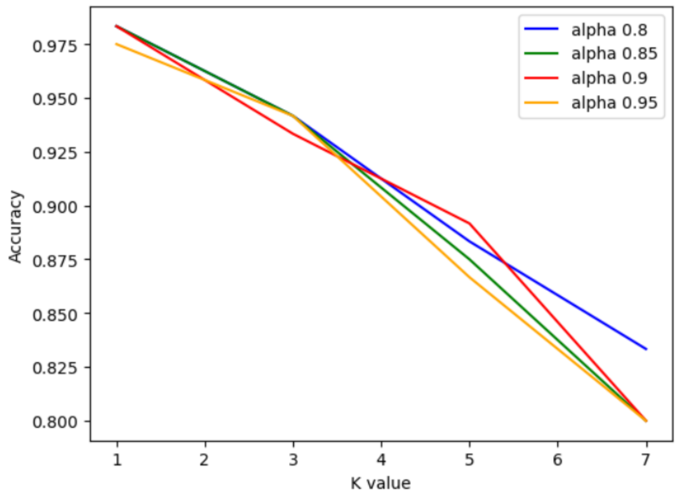
8. Bonus:

a. Different split (70% training – 30% testing):

PCA:

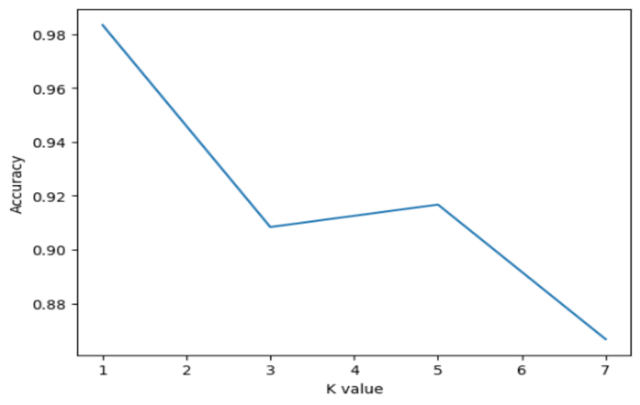
alpha	# Eigenvectors to consider	Accuracy (FNN)
0.8	40	0.983
0.85	60	0.983
0.9	92	0.983
0.95	148	0.975

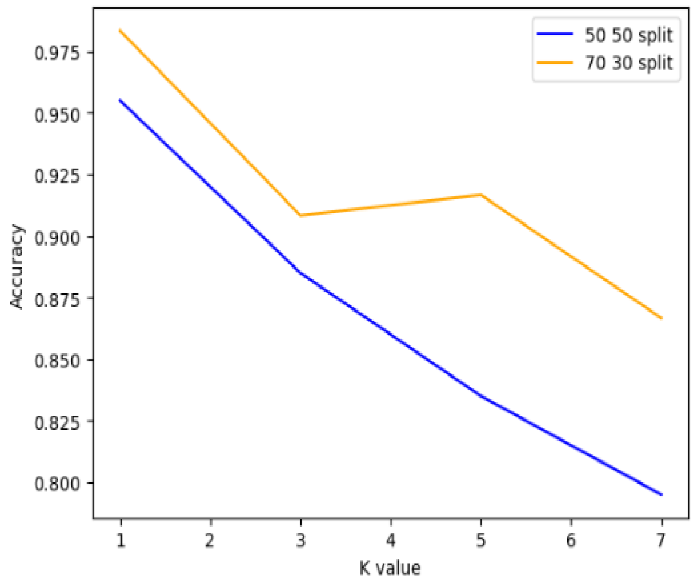
Classifier tuning: As the k value increases, the accuracy decreases.



LDA:

```
Accuracy for  1 NN =  0.9833333333333333
Accuracy for  3 NN =  0.9083333333333333
Accuracy for  5 NN =  0.9166666666666666
Accuracy for  7 NN =  0.8666666666666667
[0.9833333333333333, 0.9083333333333333, 0.9166666666666666, 0.8666666666666667]
```





As shown in the figure above, The accuracy of the new split is much higher than the old split.

b. PCA different algorithm:

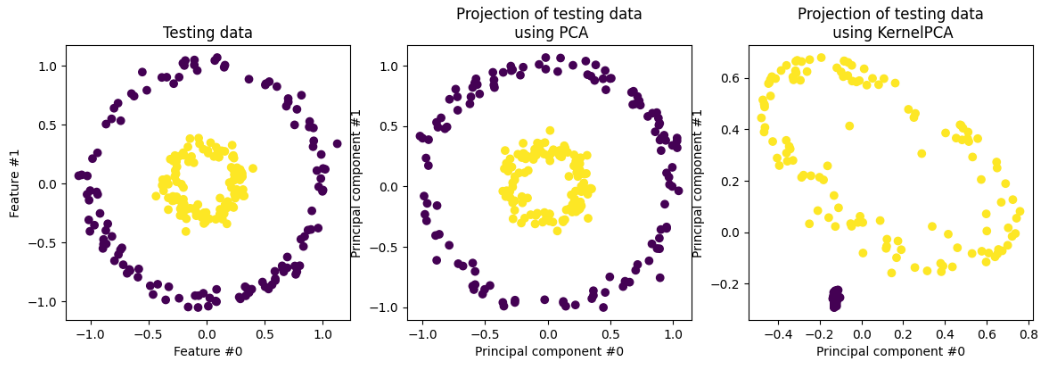


Figure 1online example of what Kernel PCA does

We recall that PCA transforms the data linearly. Intuitively, it means that the coordinate system will be centered, rescaled on each component with respected to its variance and finally be rotated. The obtained data from this transformation is isotropic and can now be projected on its *principal components*.

Thus, looking at the projection made using PCA (i.e. the middle figure), we see that there is no change regarding the scaling; indeed the data being two concentric circles centered in zero, the original data is already isotropic. However, we can see that the data have been rotated. As a conclusion, we see that such a projection would not help if define a linear classifier to distinguish samples from both classes.

Using a kernel allows to make a non-linear projection. Here, by using an RBF kernel, we expect that the projection will unfold the dataset while keeping approximately preserving the relative distances of pairs of data points that are close to one another in the original space.

We observe such behaviour in the figure on the right: the samples of a given class are closer to each other than the samples from the opposite class, untangling both sample sets. Now, we can use a linear classifier to separate the samples from the two classes.

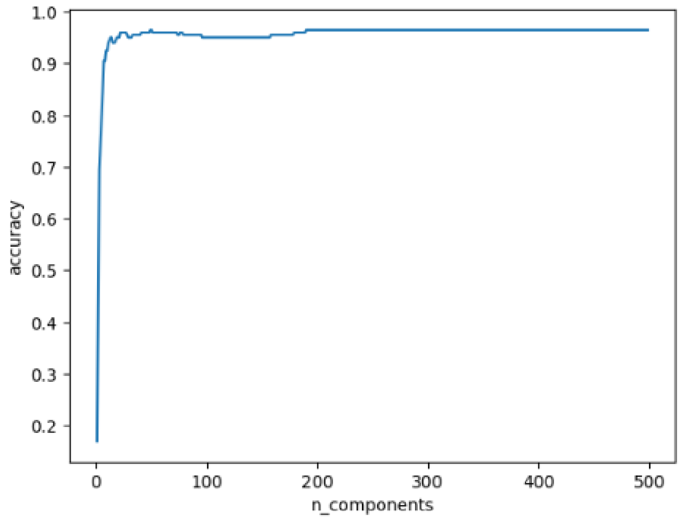
Number of eig vectors to consider	PCA	KernelPCA d=2 polynomail
36	0.97	0.955
52	0.975	0.96
76	0.97	0.96
116	0.965	0.95

Kernel PCA was developed in an effort to help with the classification of data whose decision boundaries are described by non-linear function.

Since data is separated better by a linear function , PCA performs better

```python
print(f'the best accuracy ={scores[np.argmax(scores)]}')
print(f'was for n_components={np.argmax(scores)}')
plt.plot(neighbours,scores)
plt.xlabel("n_components")
plt.ylabel("accuracy")
plt.show()
```

```
the best accuracy =0.965
was for n_components=48
```



KernelPCA tunning number of components to consider

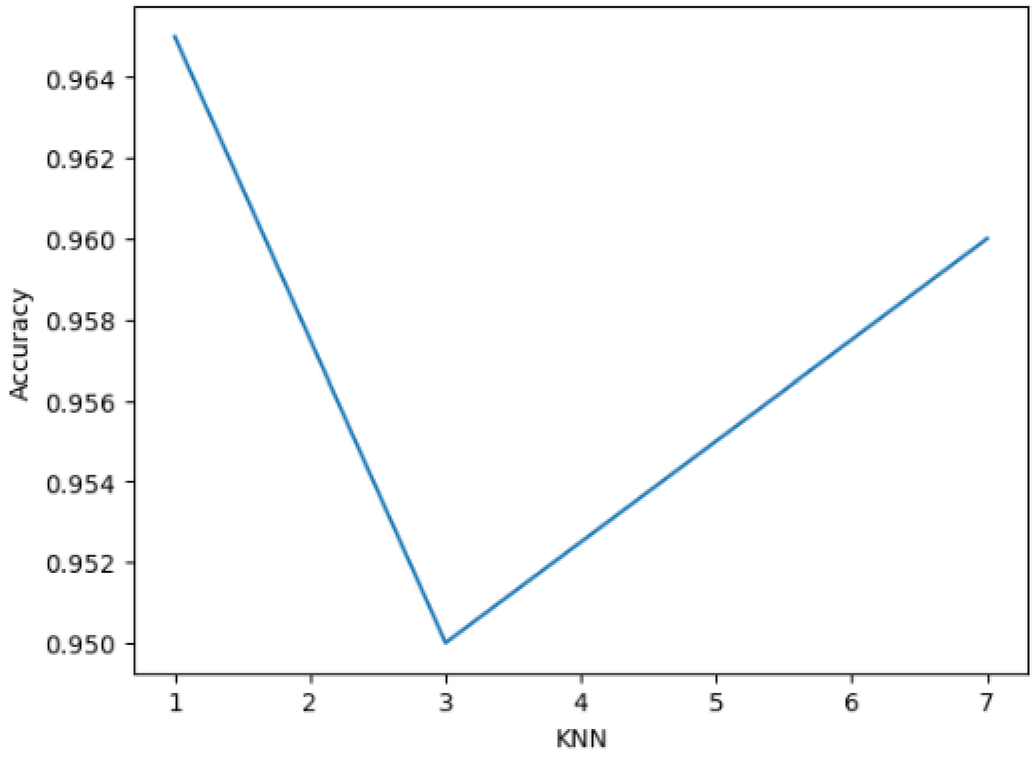
```python
#from sklearn.decomposition import KernelPCA
pca=KernelPCA(n_components=np.argmax(scores),kernel='poly',degree=2)
pca.fit(x_train)
x_test_reduced=pca.transform(x_test)
x_train_reduced=pca.transform(x_train)
```

LDA different algorithm:

Using LDA algorithm from Sklearn which by fiting a Gaussian density to each class.

It was so much fast the our naïve approach

	Naïve	**Sklearn**
Time	5.1 mins	0.7 seconds
Accuracy	95.5%	96.5%


Accuracy
KNN