PATTERN RECOGNITION

Topic: Face Recognition

Due Date: 18-3-2023

Link of code :

Report BY:

Name	ID	Group
Mazen Yasser	7234	G1
Ahmed El-Selmy	6449	G1
Shereen Mabrouk	6844	G1

Supervisor

DR. Marwan Torki

- Overview
- Code & Illustration
- Bonus

A.Overview:

The first part of this project aims to train the model to identify a person out of a selection of 40 people. The dataset is a collection of 400 black and white images for 40 people, 10 photos for each person. The dimensions of each image are 92x112.

The dataset is split into two sets, a training set, and a test set. PCA and LDA algorithms are used to project the dataset and a KNN classifier is used to predict the test set.

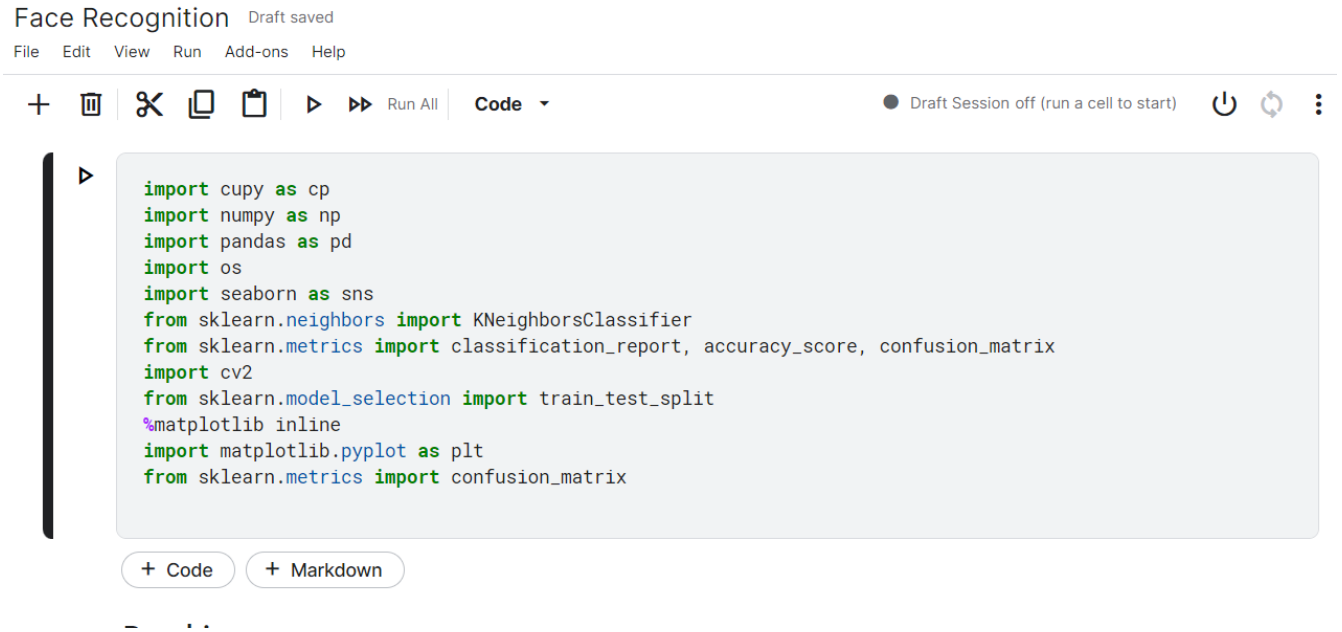
The second part of this project solves another classification problem faces vs non-faces. Other images of non-faces are added to the existing dataset and the same steps are repeated on two classes instead of 40 classes

.....

B. Code & Illustration:

We used Kaggle Notebook

First of all we add the necessary imports are then made which will be used throughout the project.



Then we read an image for testing

Face Recognition Draft saved

File Edit View Run Add-ons Help

Run All Code Draft Session off (run a cell to start)

Read images

[2]:
```
import cv2
img = cv2.imread('/kaggle/input/att-database-of-faces/s1/1.pgm', cv2.IMREAD_UNCHANGED)
print(np.array(img, dtype='float64').flatten())
plt.imshow(img,cmap='gray')
plt.show()
```

```
[48. 49. 45. ... 47. 46. 46.]
```

Reading the image from the folders and adding the data in the images matrix with labeling in labels vector

Face Recognition Draft saved

File Edit View Run Add-ons Help

Run All Code Draft Session off (run a cell to start)

0 20 40 60 80

+ Code + Markdown

[3]:
```
images = np.ones((1, 10304))
labels=np.array([])
for i in range(1,41):
  photos=[i for i in range(1, 10+1)]
  for photo in photos:
    img = cv2.imread('/kaggle/input/att-database-of-faces/s'+str(i)+"/"+str(photo)+".pgm",0) # zero for gra
    img = np.reshape(img, (1, 10304))
    images = np.append(images, img, axis=0)
    labels = np.append(labels, i)
labels=np.array(labels)
images=images[1:,]
```

[4]:
```
images
```

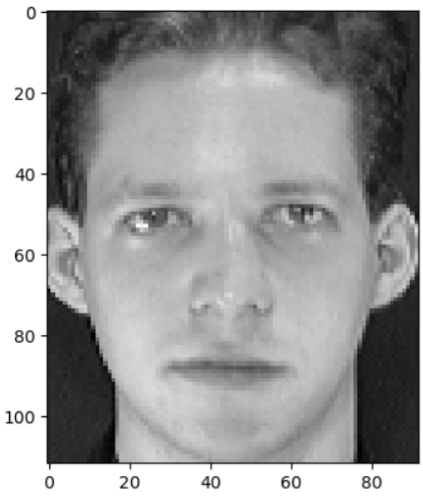
[4]:
```
array([[ 48.,  49.,  45., ...,  47.,  46.,  46.],
       [ 60.,  60.,  62., ...,  32.,  34.,  34.],
       [ 39.,  44.,  53., ...,  29.,  26.,  29.],
       ...,
       [125., 119., 124., ...,  36.,  39.,  40.],
       [119., 120., 120., ...,  89.,  94.,  85.],
       [125., 124., 124., ...,  36.,  35.,  34.]])
```

Face Recognition Draft saved

File Edit View Run Add-ons Help

Run All Code Draft Session off (run a cell to start)

[6]:
```
plt.imshow(images[0].reshape((112, 92)), cmap='gray')
plt.show()
```



+ Code + Markdown

##Train Test Split

Train & test split

[7]:
```
X_train=images[1::2] # Odd
X_test=images[0::2]  # Even
y_train=labels[1::2] # Odd
y_test=labels[0::2]  # Even
```

[8]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(200, 10304)
(200, 10304)
```

[9]:
```
images.shape
```

[9]: (400, 10304)

Then we Create PCA Code according to this algorithm

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

Classification using PCA Code:

[10]:

```
#Calculating mean
def calculate_mean(data):
    return np.mean(data,axis=0,keepdims=True)

#Centralize data
def centeralize(data):
    return data - calculate_mean(data)

#Covariance Matrix
def calculate_covariance_matrix(data):
    z = centeralize(data)
    return (np.matmul(np.transpose(z), z)) / len(data)

#Calculating Eigen values & Eigen vectors
def calculate_eigen_vectors(data):
    cov = calculate_covariance_matrix(data)
    eig_values, eig_vectors = cp.linalg.eigh(cov)
    #Sorting eigenvalues and eigenvectors respectivily
    idx = eig_values.argsort()[::-1]
    eig_values = eig_values[idx]
    eig_vectors = eig_vectors[:,idx]
    return eig_values, eig_vectors
```

Run All | Code | Draft Session off (run a cell to start)

```
#Choosing dimensionality
def dimensionality(alpha, eig_values):
    sum = np.sum(eig_values)
    r = 0
    i = 0
    for value in eig_values:
        r = r + eig_values[i]
        i = i + 1
        if (r / float(sum) >= alpha):
            break
    return i

def reduce_matrix(r, matrix):
  U = matrix[:,:r]
  return U

def PCA(alpha,X_train,X_test):
    eig_values, eig_vectors = calculate_eigen_vectors(X_train)
    print("Eigenvectors shape = "+str(eig_vectors.shape))
    r = dimensionality(alpha, eig_values)
    print("r = ", r)
    U = reduce_matrix(r, eig_vectors)
    print("U shape = "+str(U.shape))
    projected_training = np.dot(X_train, U)
    projected_testing = np.dot(X_test, U)
    print("projected_training shape = "+str(projected_training.shape))
    print("projected_testing shape = "+str(projected_testing.shape))
    return projected_training, projected_testing
```

Face Recognition Draft saved

File Edit View Run Add-ons Help

Run All | Code | Draft Session off (run a cell to start)

[11]:

```
def classify_and_show_failure(X_train, y_train, X_test,org_test, y_test, n_neighbors):
#     from google.colab.patches import cv2_imshow
    model = KNeighborsClassifier(n_neighbors=n_neighbors)
    model.fit(X_train, y_train)
    test_pred = model.predict(X_test)
    for i in range(len(y_train)):
      if int(test_pred[i])!=int(y_test[i]):
        print("Failure Case")
        print(i)
        if int(test_pred[i])==2:
          print("Predicted non face but actual is face")
          plt.figure()
          plt.imshow(org_test[i].get().reshape((112, 92)), cmap='gray')
          plt.axis('off')
          plt.show()
        else:
          print("Predicted face but actual is non face")
          plt.figure()
          plt.imshow(org_test[i].get().reshape((112, 92)), cmap='gray')
          plt.axis('off')
          plt.show()
    acc=accuracy_score(y_test,test_pred)
    print("acc = "+str(acc))
    return acc
```

Run All | Code | Draft Session off (run a cell to start)

[12]:

```
def test_pca(X_train,X_test,y_train,y_test,show_failure):
  accuracy=[]
  alphas_dict_proj={} # each key represents a value of alpha , the value of each key is a list containing two matrices
  alphas_dict_acc={}
  alphas_dict_acc["0.8"]=[]
  alphas_dict_acc["0.85"]=[]
  alphas_dict_acc["0.9"]=[]
  alphas_dict_acc["0.95"]=[]
  model = KNeighborsClassifier(n_neighbors=1)
  alphas=[0.8,0.85,0.9,0.95]
  for alpha in alphas:
    matrices=[]
    projected_training, projected_testing = PCA(alpha,X_train,X_test)
    matrices.append(projected_training.get())
    matrices.append(projected_testing.get())
    alphas_dict_proj[str(alpha)]=matrices
    model.fit(projected_training.get(), y_train.get())
    test_pred = model.predict(projected_testing.get())
    acc=accuracy_score(y_test.get(), test_pred)
    print("ACCURACY FOR ALPHA:"+ str(alpha)+' AND K = 1 IS :- '+str(acc))
    alphas_dict_acc[str(alpha)].append(acc)
    accuracy.append(acc)
    cm=confusion_matrix(y_test.get(),test_pred)
    print("CONFUSION MATRIX:")
    print(cm)
    print("=============")
    if show_failure == 1:
```

Run All | Code | Draft Session off (run a cell to start)

```
      print("=============")
      if show_failure == 1:
        accu=classify_and_show_failure(projected_training.get(), y_train.get(), projected_testing.get(),X_test, y_test.ge
  import matplotlib.pyplot as plt
  plt.figure(figsize=(15,5))
  plt.plot(alphas, accuracy,'--bo', label='line with marker')
  plt.xlabel("Alpha")
  plt.ylabel("Accuracy")
  plt.title("Accuracy using K = 1")
  plt.show()
  avg_accuracies=[]
  avg_accuracies.append(sum(accuracy)/len(accuracy))
  Ks=[3,5,7]
  plt.figure(figsize=(10,10))
  for k in Ks:
    kAccuracy=[]
    model = KNeighborsClassifier(n_neighbors=k)
    alphas=[0.8,0.85,0.9,0.95]
    for alpha in alphas:
      projected_training=alphas_dict_proj[str(alpha)][0]
      projected_testing=alphas_dict_proj[str(alpha)][1]
      model.fit(projected_training, y_train.get())
      test_pred = model.predict(projected_testing)
      acc=accuracy_score(y_test.get(), test_pred)
      print("ACCURACY FOR ALPHA:"+ str(alpha)+' AND K = ' + str(k)+' IS :- '+str(acc))
      alphas_dict_acc[str(alpha)].append(acc)
      kAccuracy.append(accuracy_score(y_test.get(), test_pred))
      cm=confusion_matrix(y_test.get(),test_pred)
      print("CONFUSION MATRIX:")
```

```
      print(cm)
      print("=============")
      if show_failure == 1:
        accu=classify_and_show_failure(projected_training, y_train.get(), projected_testing,X_test, y_test.get(), k)
    print("Done for k = "+str(k)+"\n======================\n")
    plt.plot(alphas, kAccuracy,'--bo', label='line with marker')
    plt.title("Accuracy using K = "+ str(k))
    plt.xlabel("Alpha")
    plt.ylabel("Accuracy")
    plt.show()
    avg_accuracies.append(sum(kAccuracy)/len(kAccuracy))
  index=1
  plt.figure(figsize=(10,10))
  for alpha in alphas:
    plt.subplot(2, 2, index)
    index+=1
    plt.plot([1,3,5,7], alphas_dict_acc[str(alpha)],'--bo', label='line with marker')
    plt.xlabel("K Value")
    plt.ylabel("Accuracy")
    plt.title("Different accuracies for alpha = "+str(alpha))
  plt.show()
  plt.figure(figsize=(10,10))
  plt.plot([1,3,5,7], avg_accuracies,'--bo', label='line with marker')
  plt.xlabel("K Value")
  plt.ylabel("Avg Accuracy")
  plt.title("Average accuracy for different values of K")
  plt.show()
```

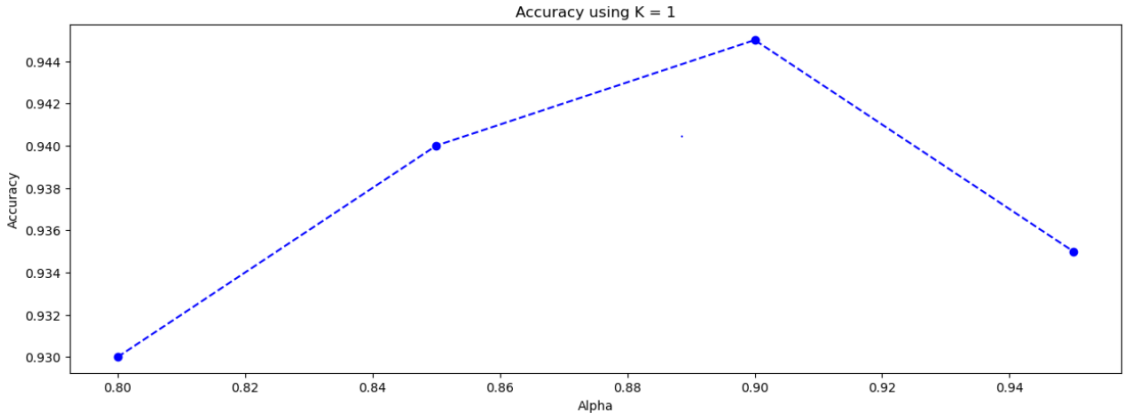
Outputs: We will see the accuracy with different K values and the accuracy with alphas [0.8 ,0.85,0.9,0.95] and the average accuracy with different k

```
test_pca(cp.asarray(X_train),cp.asarray(X_test),cp.asarray(y_train),cp.asarray(y_test),0)
```

```
Eigenvectors shape = (10304, 10304)
r =  37
U shape = (10304, 37)
projected_training shape = (200, 37)
projected_testing shape = (200, 37)
ACCURACY FOR ALPHA:0.8 AND K = 1 IS :- 0.93
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 5 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 2]]
=============
Eigenvectors shape = (10304, 10304)
r =  53
U shape = (10304, 53)
projected_training shape = (200, 53)
projected_testing shape = (200, 53)
ACCURACY FOR ALPHA:0.85 AND K = 1 IS :- 0.94
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 5 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 2]]
-------------
```

At k=1

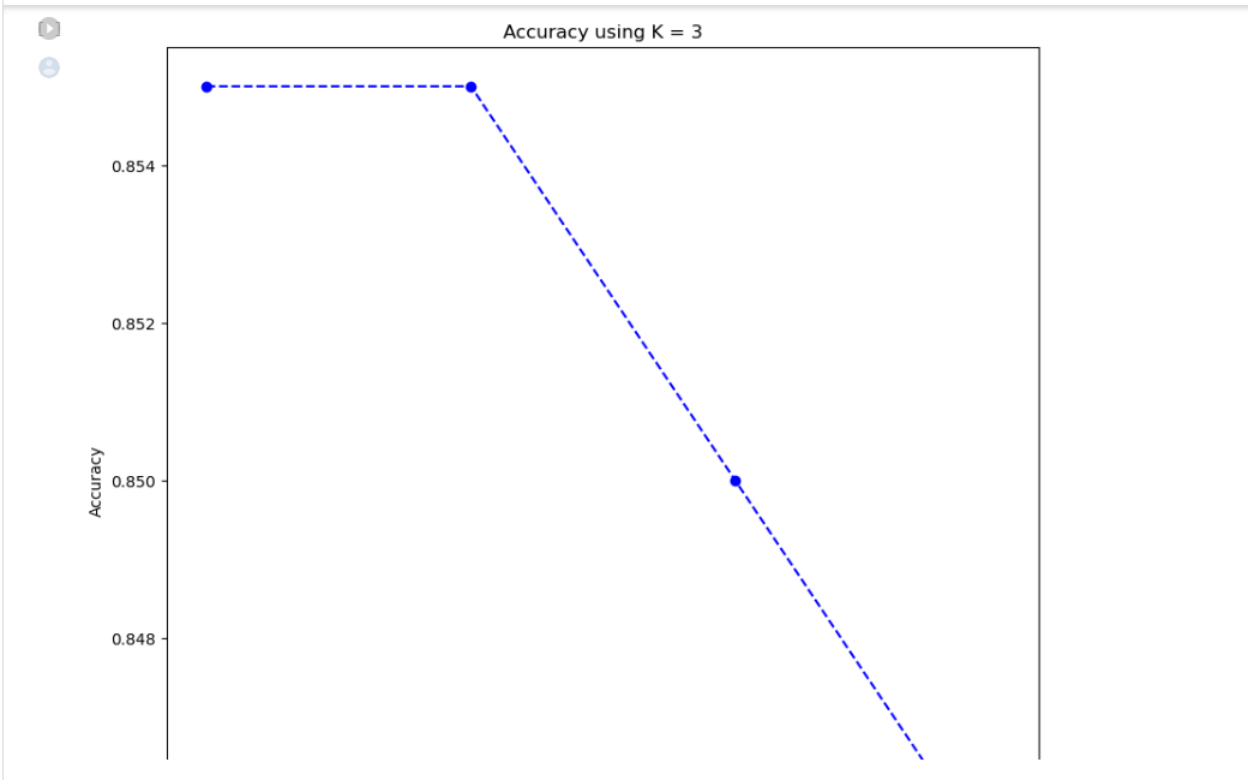
```
Eigenvectors shape = (10304, 10304)
r =  116
U shape = (10304, 116)
projected_training shape = (200, 116)
projected_testing shape = (200, 116)
ACCURACY FOR ALPHA:0.95 AND K = 1 IS :- 0.935
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 5 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 2]]
============
```



In the previous plot we can see that when the maximum accuracy using k=1 is at alpha of value 0.95

```
ACCURACY FOR ALPHA:0.8 AND K = 3 IS :- 0.855
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 2]]
============
ACCURACY FOR ALPHA:0.85 AND K = 3 IS :- 0.855
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 2]]
============
ACCURACY FOR ALPHA:0.9 AND K = 3 IS :- 0.85
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 2]]
============
ACCURACY FOR ALPHA:0.95 AND K = 3 IS :- 0.845
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 2]]
============
```

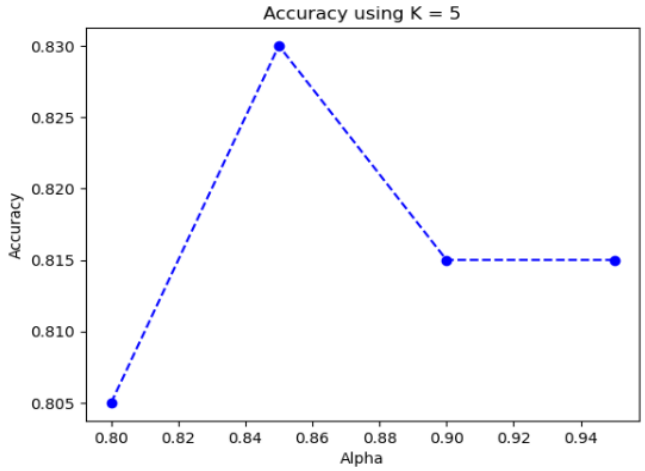
At k=3




```
ACCURACY FOR ALPHA:0.8 AND K = 5 IS :- 0.805
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 1]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 5 IS :- 0.83
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 1]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 5 IS :- 0.815
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 1]]
=============
```

At k=5

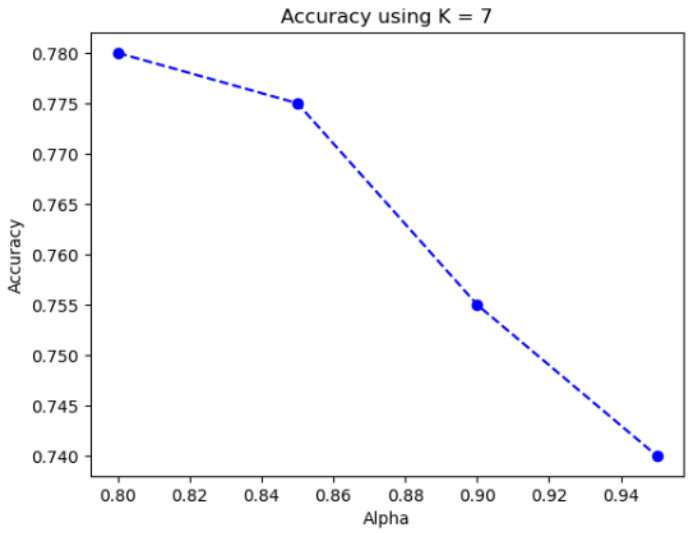
```
ACCURACY FOR ALPHA:0.95 AND K = 5 IS :- 0.815
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 1]]
=============
Done for k = 5
======================
```



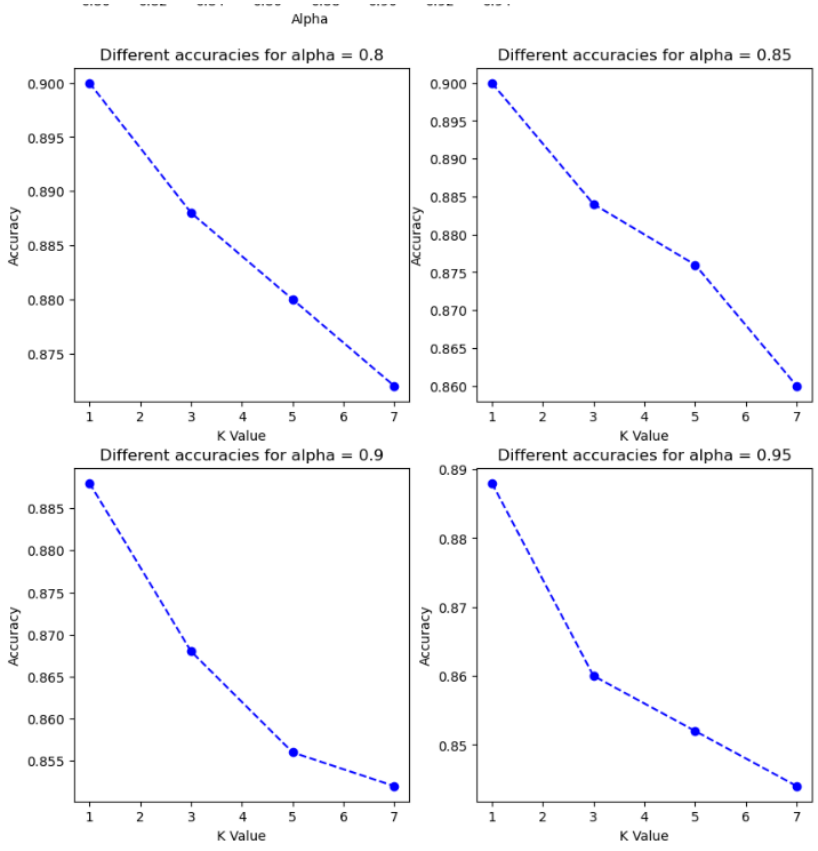

```
test_pca(cp.asarray(X_train),cp.asarray(X_test),cp.asarray(y_train),cp.asarray(y_test),0)
ACCURACY FOR ALPHA:0.8 AND K = 7 IS :- 0.78
CONFUSION MATRIX:
[[3 2 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 5 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 0]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 7 IS :- 0.775
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 5 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 0]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 7 IS :- 0.755
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 5 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 1]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 7 IS :- 0.74
CONFUSION MATRIX:
[[3 1 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 1]]
=============
Done for k = 7
======================
```

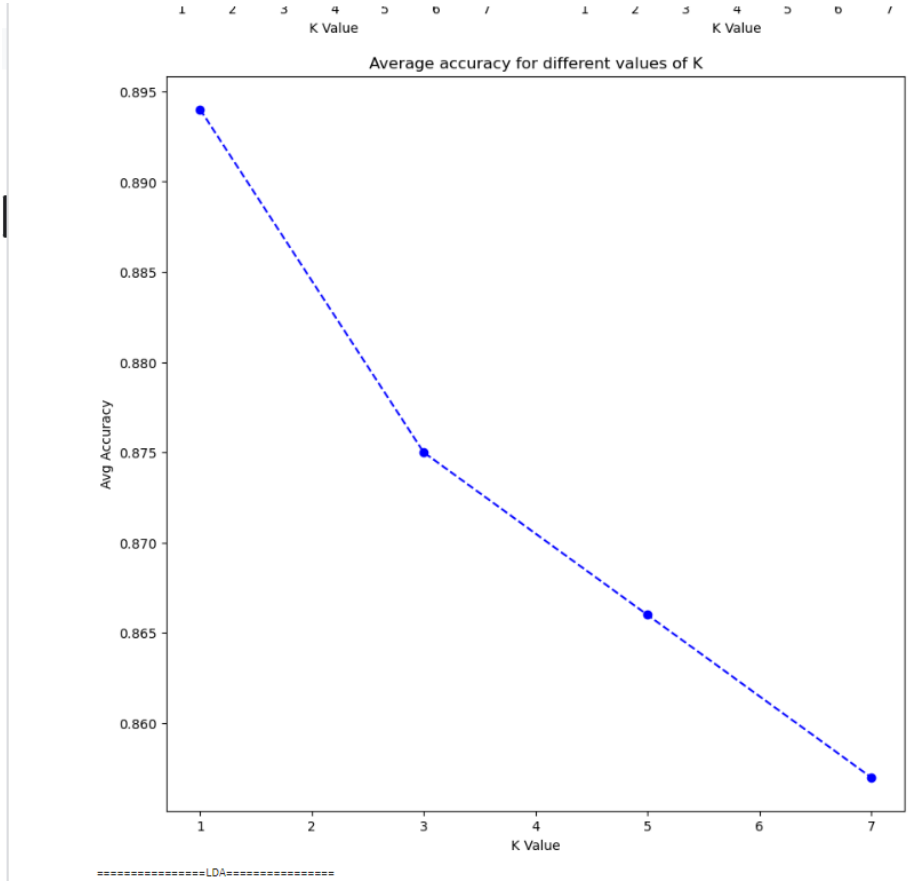
At k=7

==========



Plots for alphas





Insights From The previous outputs:

-When we increase the value of k the accuracy decreases due to noise as when the value of k is large the algorithm may start to include neighbors that are not relevant to prediction so this leads to poor accuracy

- When we increase alpha, the accuracy decreases as we take more eigenvectors therefore, we don’t make use of dim reduction.

Then we Create LDA Code according to this algorithm

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

Classification using LDA Code:

[14]:
```
def project(data,proj_matrix):
  return np.dot(proj_matrix,data.T).T
```

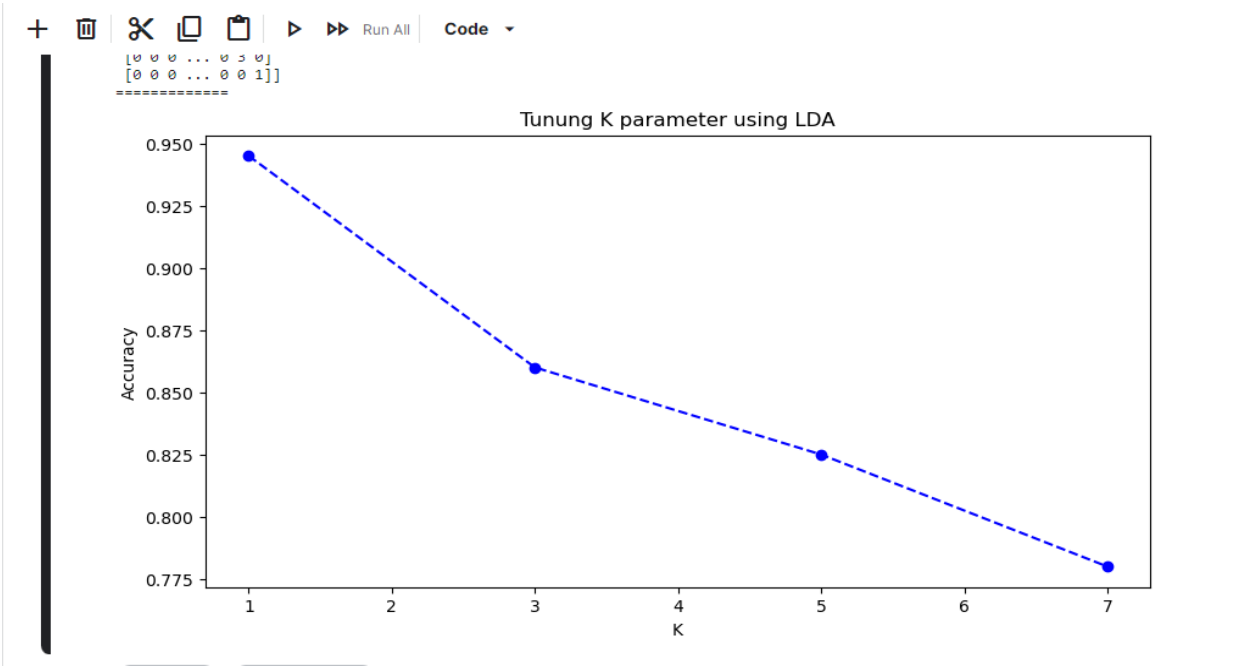
[15]:
```
def binary_lda(train_data,test_data,X_train,X_test,eigenvectors):
    classes_means=np.zeros((2,10304))
    classes_matrix=[]
    for i in range(2):
        class_matrix=train_data[train_data['target']==(i+1)]
        class_matrix=class_matrix.drop(['target'], axis=1)
        class_matrix2=np.asarray(class_matrix).astype('float64')
        classes_matrix.append(class_matrix2)
        class_mean=np.mean(class_matrix2,axis=0,keepdims=True)
        classes_means[i]=class_mean
    Z=[]
    for i in range(2):
        centered=cp.asarray(classes_matrix[i])-cp.asarray(classes_means[i])
        Z.append(centered)
    all_classes_mean = np.mean(train_data, axis=0)
    all_classes_mean = all_classes_mean[:-1]
    all_classes_mean=np.asarray(all_classes_mean).astype('float64')
    Sb=cp.zeros((10304,10304))
    all_classes_mean=all_classes_mean.reshape(10304,1)
    for i in range(2):
        class_mean=classes_means[i].reshape(10304,1)
        print("Number of samples in class "+str(i)+" equal "+str(len(classes_matrix[i])))
        Sb+=len(classes_matrix[i])*np.dot((cp.asarray(class_mean)-cp.asarray(all_classes_mean)),(cp.asarray(class_mean)-cp.asarray(all_classes_mean)).T)
    S=np.ones((10304,10304))
    for i in range(2):
        S+=np.dot(cp.asnumpy(Z[i].T),cp.asnumpy(Z[i]))
    eig_values,eig_vectors=cp.linalg.eigh(cp.dot(cp.asarray(np.linalg.inv(S)),Sb))
    eig_values = eig_values[::-1]
    eig_vectors = eig_vectors[:,::-1]
    selected=eig_vectors[:,:eigenvectors]
    train_proj,test_proj=project(X_train,selected.T),project(X_test,selected.T)
    return train_proj,test_proj
```

```
def get_classes_means(X_train,train_size):
  classes_means=np.zeros((40,10304))
  classes_matrix=[] # to be used in another steps
  for i in range(40):
    class_matrix=X_train[i*train_size:(i+1)*train_size,:]
    classes_matrix.append(class_matrix)
    class_mean=np.mean(class_matrix,axis=0,keepdims=True)
    classes_means[i]=class_mean.get()
  return classes_means,classes_matrix
def between_class_scatter(X_train,classes_means,train_size):
  all_classes_mean=np.mean(X_train,axis=0,keepdims=True)
  Sb=cp.zeros((10304,10304))
  all_classes_mean=all_classes_mean.reshape(10304,1)
  for i in range(40):
    class_mean=classes_means[i].reshape(10304,1)
    Sb+=train_size*np.dot((cp.asarray(class_mean)-cp.asarray(all_classes_mean)),(cp.asarray(class_mean)-cp.asarray(all_classes_mean)).T)
  return Sb
def center_class_matrix(classes_matrix,classes_means):
  Z=[]
  for i in range(40):
    centered=cp.asarray(classes_matrix[i])-cp.asarray(classes_means[i])
    Z.append(centered)
  return Z
def class_scatter_matrix(Z):
  S=np.ones((10304,10304))
  for i in range(40):
    S+=np.dot(cp.asnumpy(Z[i].T),cp.asnumpy(Z[i]))
  return S
def project(data,proj_matrix):
  return np.dot(proj_matrix,data.T).T
def LDA(X_train,X_test,eigenvectors,train_size):
  classes_means,classes_matrix=get_classes_means(X_train,train_size)
  Sb=between_class_scatter(X_train,classes_means,train_size)
  Z=center_class_matrix(classes_matrix,classes_means)
  S=class_scatter_matrix(Z)
  eig_values,eig_vectors=cp.linalg.eigh(cp.dot(cp.asarray(np.linalg.inv(S)),Sb))
  eig_values = eig_values[::-1]
  eig_vectors = eig_vectors[:,::-1]
  selected=eig_vectors[:,:eigenvectors]
  train_proj,test_proj=project(X_train,selected.T),project(X_test,selected.T)
  return train_proj,test_proj
```

```
def test_lda(X_train,X_test,y_train,y_test,eigenvectors,train_size,show_failure):
  train_proj,test_proj=LDA(X_train,X_test,eigenvectors,train_size)
  Ks=[1,3,5,7]
  lda_accuracies=[]
  for k in Ks:
    model=KNeighborsClassifier(n_neighbors=k)
    model.fit(train_proj.get(), y_train.get())
    test_pred = model.predict(test_proj.get())
    acc=accuracy_score(y_test.get(), test_pred)
    print("Accuracy using k = "+str(k)+" = "+str(acc))
    lda_accuracies.append(acc)
    cm=confusion_matrix(y_test.get(),test_pred)
    print("CONFUSION MATRIX:")
    print(cm)
    print("=============")
    if show_failure == 1:
      accu=classify_and_show_failure(train_proj.get(), y_train.get(), test_proj.get(),X_test, y_test.get(), k)
  plt.figure(figsize=(10,5))
  plt.plot(Ks, lda_accuracies,'--bo', label='line with marker')
  plt.title("Tunung K parameter using LDA")
  plt.xlabel("K")
  plt.ylabel("Accuracy")
  plt.show()
```

```
[16]: test_lda(cp.asarray(X_train),cp.asarray(X_test),cp.asarray(y_train),cp.asarray(y_test),39,5,0)
```

```
Accuracy using k = 1 = 0.945
CONFUSION MATRIX:
[[4 0 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 5 ... 0 0 0]
 ...
 [0 0 0 ... 4 0 0]
 [0 0 0 ... 0 5 0]
 [0 0 0 ... 0 0 3]]
=============
Accuracy using k = 3 = 0.86
CONFUSION MATRIX:
[[4 0 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 2]]
=============
Accuracy using k = 5 = 0.825
CONFUSION MATRIX:
[[4 0 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 2]]
=============
Accuracy using k = 7 = 0.78
CONFUSION MATRIX:
[[4 0 0 ... 0 0 0]
 [0 5 0 ... 0 0 0]
 [0 0 4 ... 0 0 0]
 ...
 [0 0 0 ... 5 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 1]]
=============
```



Insights From The previous outputs:

As we see in the previous plot when the values of k increase the accuracy decrease due to noise

In general, LDA was expected to perform better than PCA as this is a classification problem. It resulted in slightly better accuracy, but it was very close which reflects how well the eigen faces using the PCA works.

FACES AND NON FACES :

First of all we read the dataset of non-faces then we do the same steps as the previous classification problem

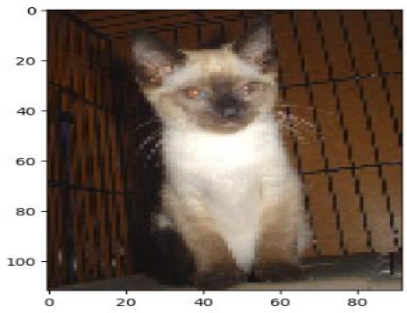
```
+ Code  + Markdown

[72]:
# from IPython.display import clear_output
# Modify the first path to be the path of the data in your drive
# !unzip /content/drive/MyDrive/PatternRecognition/Assignment1Data/animals.zip -d /content/sample_data/animals
# clear_output(wait=False)
```

##Read Images

```
# from google.colab.patches import cv2_imshow
import cv2
img2 = plt.imread('/kaggle/input/animals/animals/647.jpg',0)
img2=cv2.resize(img2,(92,112))
print(np.array(img2, dtype='float64').flatten())
plt.imshow(img2,cmap='gray',vmin=0, vmax=255)
# failure = plt.imread('/content/sample_data/animalss/animals/647.jpg',0) # zero for gray scale
# plt.imshow(failure,cmap='gray')
```

```
[ 46.  22.  21. ... 142. 128. 118.]
<matplotlib.image.AxesImage at 0x7f2d91327c90>
```




```
+ Code  + Markdown

[18]:
images1 = np.ones((1, 10304))
labels1=np.array([])
photos=[i for i in range(1, 800+1)]
for photo in photos:
    img1 = cv2.imread('/kaggle/input/animals/animals/'+str(photo)+".jpg",0) # zero for gray scale
    img1 = np.resize(img1,(92,112))
    img1 = np.reshape(img1, (1, 10304))
    images1 = np.append(images1, img1, axis=0)
    labels1 = np.append(labels1, 2)
labels1=np.array(labels1)
images1=images1[1:,]
```

```
+ Code  + Markdown

[19]:
print(np.array(images1[3], dtype='float64').flatten())
```

```
[131. 132. 135. ... 215. 209. 205.]
```

```
[20]:
print(images1.shape)
print(labels1.shape)
```

```
(800, 10304)
(800,)
```

##Adjusting first dataset's label

[21]:
```
labels2= np.array([])
for i in range(1,401):
  labels2 = np.append(labels2,1)
labels2.shape
```

[21]: (400,)

##Adding both datasets

[22]:
```
images
```

[22]:
```
array([[ 48.,  49.,  45., ...,  47.,  46.,  46.],
       [ 60.,  60.,  62., ...,  32.,  34.,  34.],
       [ 39.,  44.,  53., ...,  29.,  26.,  29.],
       ...,
       [125., 119., 124., ...,  36.,  39.,  40.],
       [119., 120., 120., ...,  89.,  94.,  85.],
       [125., 124., 124., ...,  36.,  35.,  34.]])
```

```
total_images= np.concatenate((images, images1))
total_labels= np.concatenate((labels2, labels1))
```

+ Code + Markdown

[24]:
```
print(total_images.shape)
total_labels.shape
```

```
(1200, 10304)
```

[24]: (1200,)

[25]:
```
X_train1=total_images[1::2] # Odd
X_test1=total_images[0::2]  # Even
y_train1=total_labels[1::2] # Odd
y_test1=total_labels[0::2]  # Even
```

##Checking dimensions

[26]:
```
print(X_train1.shape)
print(X_test1.shape)
print(X_train1[1])
print(y_train1[1])
```

```
(600, 10304)
(600, 10304)
[63. 53. 35. ... 41. 10. 24.]
1.0
```

LDA Face vs non face 400 and (100,200,300,400,600,800)

[27]:

```
loops = [100,200,300,400,600,800]
for loop in loops:
  print('RUNNING PCA AND LDA WITH NONFACES = '+ str(loop))
  print('-------------------------------------')
  print('-------------------------------------')
  photos=[i for i in range(1, loop+1)]
  images1 = np.ones((1, 10304))
  labels1=np.array([])
  for photo in photos:
      img1 = cv2.imread('/kaggle/input/animals/animals/'+str(photo)+".jpg",0) # zero for gray scale
      img1 = cv2.resize(img1,(92,112))
      img1 = np.reshape(img1, (1, 10304))
      images1 = np.append(images1, img1, axis=0)
      labels1 = np.append(labels1, 2)
  labels1=np.array(labels1)
  images1=images1[1:,]

  total_images= np.concatenate((images, images1))
  total_labels= np.concatenate((labels2, labels1))

  X_train1=total_images[1::2] # Odd
  X_test1=total_images[0::2]  # Even
  y_train1=total_labels[1::2] # Odd
  y_test1=total_labels[0::2]  # Even
  if loop == 400:
    print("================PCA================")
    test_pca(cp.asarray(X_train1),cp.asarray(X_test1),cp.asarray(y_train1),cp.asarray(y_test1),1)
    print("================LDA================")
    test_lda(cp.asarray(X_train1),cp.asarray(X_test1),cp.asarray(y_train1),cp.asarray(y_test1),2,5,1)
  else:
    print("================PCA================")
    test_pca(cp.asarray(X_train1),cp.asarray(X_test1),cp.asarray(y_train1),cp.asarray(y_test1),0)
    print("================LDA================")
    test_lda(cp.asarray(X_train1),cp.asarray(X_test1),cp.asarray(y_train1),cp.asarray(y_test1),2,5,0)
```

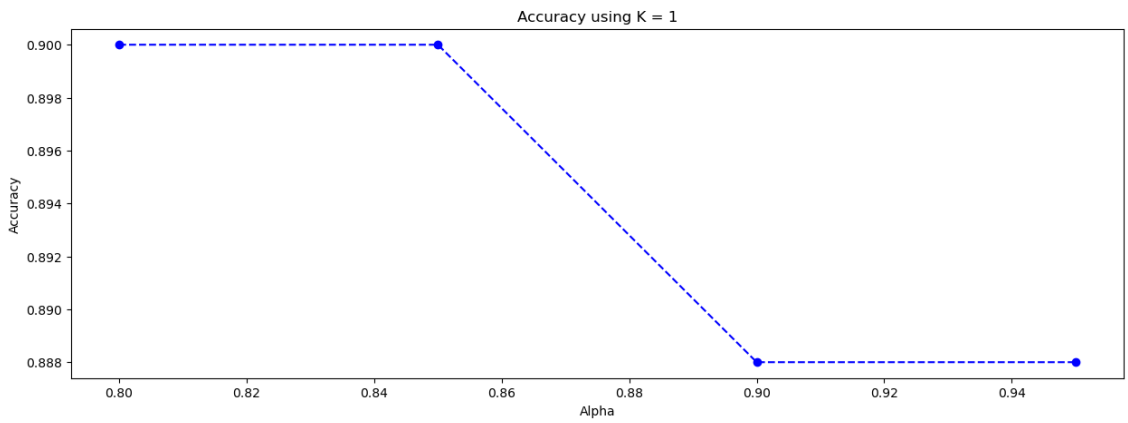
Note :
Same algorithm used for the faces problem but instead picking 39 dominant eigenvectors to classify 40 classes, this problem we got only two classes so we used 1 dominant eigenvector.

The following outputs will show the comparison between 400 faces and different number of non faces [100-200-400-600-800] in PCA and LDA

400 Face and 100 Non-face:

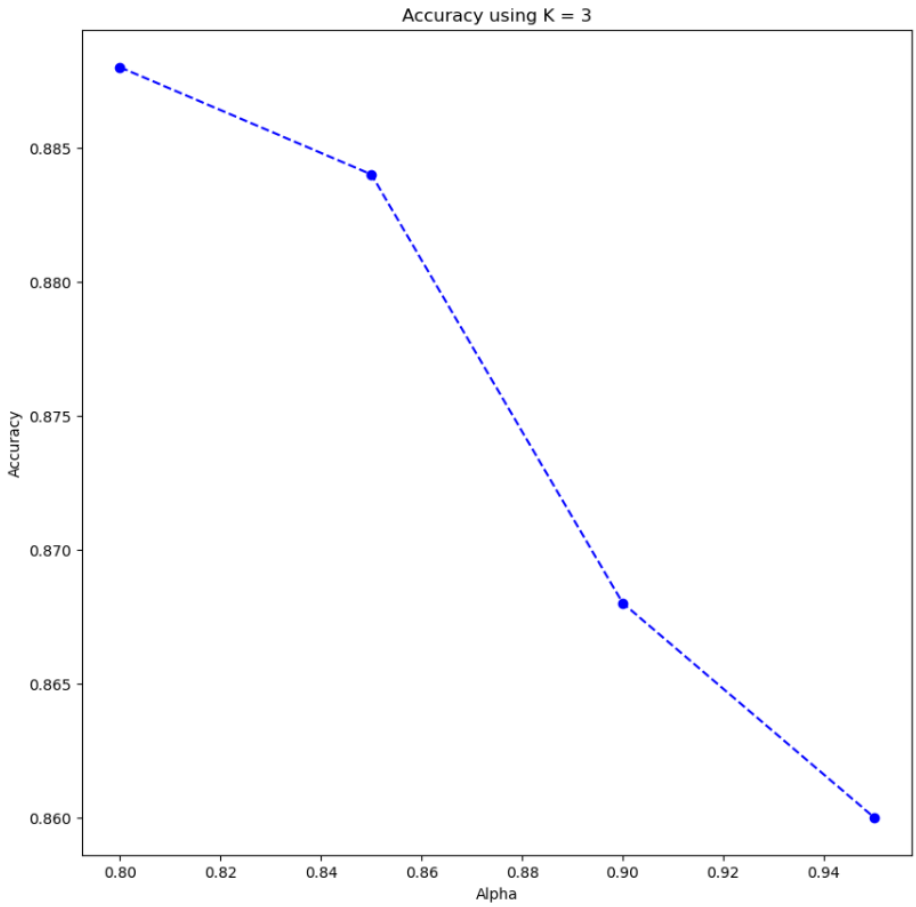
PCA:

```
RUNNING PCA AND LDA WITH NONFACES = 100
-------------------------------------
-------------------------------------
================PCA================
Eigenvectors shape = (10304, 10304)
r =  35
U shape = (10304, 35)
projected_training shape = (250, 35)
projected_testing shape = (250, 35)
ACCURACY FOR ALPHA:0.8 AND K = 1 IS :- 0.9
CONFUSION MATRIX:
[[200   0]
 [ 25  25]]
=============
Eigenvectors shape = (10304, 10304)
r =  51
U shape = (10304, 51)
projected_training shape = (250, 51)
projected_testing shape = (250, 51)
ACCURACY FOR ALPHA:0.85 AND K = 1 IS :- 0.9
CONFUSION MATRIX:
[[200   0]
 [ 25  25]]
=============
Eigenvectors shape = (10304, 10304)
r =  77
U shape = (10304, 77)
projected_training shape = (250, 77)
projected_testing shape = (250, 77)
ACCURACY FOR ALPHA:0.9 AND K = 1 IS :- 0.888
CONFUSION MATRIX:
[[200   0]
 [ 28  22]]
=============
Eigenvectors shape = (10304, 10304)
r =  125
U shape = (10304, 125)
projected_training shape = (250, 125)
projected_testing shape = (250, 125)
ACCURACY FOR ALPHA:0.95 AND K = 1 IS :- 0.888
CONFUSION MATRIX:
[[200   0]
 [ 28  22]]
=============
```

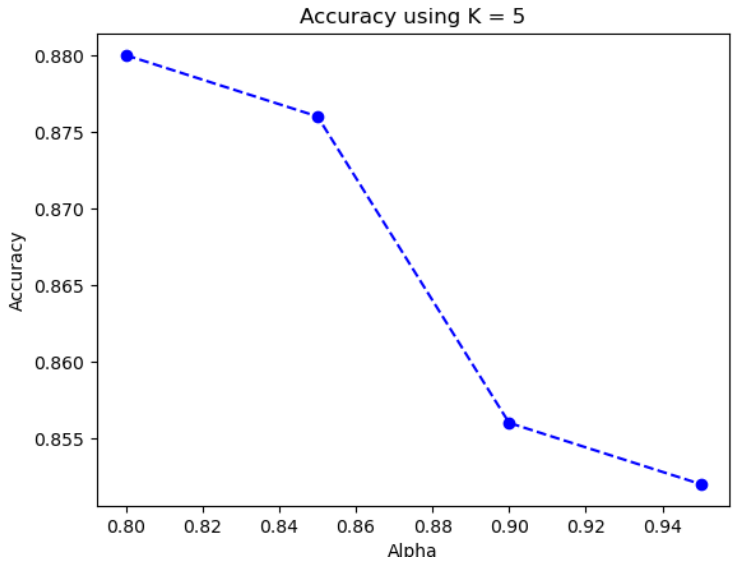


0.80 0.82 0.84 0.8

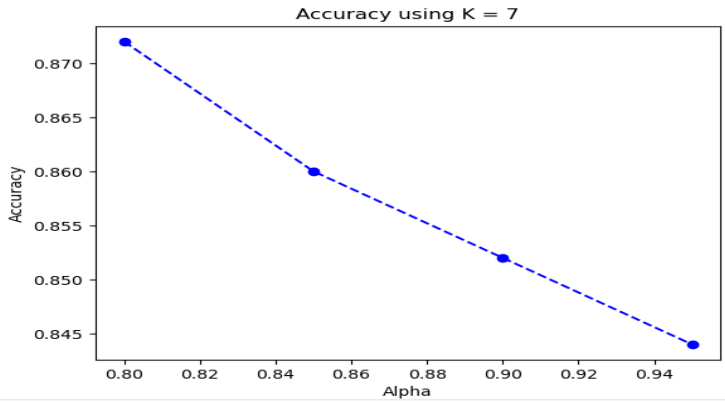
```
ACCURACY FOR ALPHA:0.8 AND K = 3 IS :- 0.888
CONFUSION MATRIX:
[[200   0]
 [ 28  22]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 3 IS :- 0.884
CONFUSION MATRIX:
[[200   0]
 [ 29  21]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 3 IS :- 0.868
CONFUSION MATRIX:
[[200   0]
 [ 33  17]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 3 IS :- 0.86
CONFUSION MATRIX:
[[200   0]
 [ 35  15]]
=============
Done for k = 3
======================
```



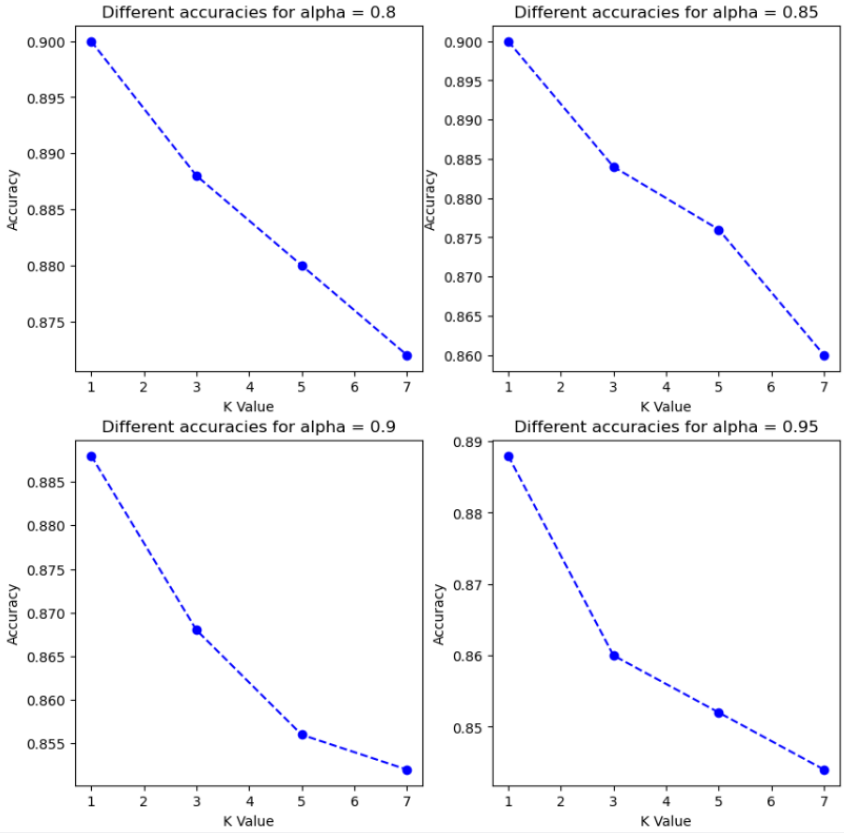
```
ACCURACY FOR ALPHA:0.8 AND K = 5 IS :- 0.88
CONFUSION MATRIX:
[[200   0]
 [ 30  20]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 5 IS :- 0.876
CONFUSION MATRIX:
[[200   0]
 [ 31  19]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 5 IS :- 0.856
CONFUSION MATRIX:
[[200   0]
 [ 36  14]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 5 IS :- 0.852
CONFUSION MATRIX:
[[200   0]
 [ 37  13]]
=============
Done for k = 5
======================
```

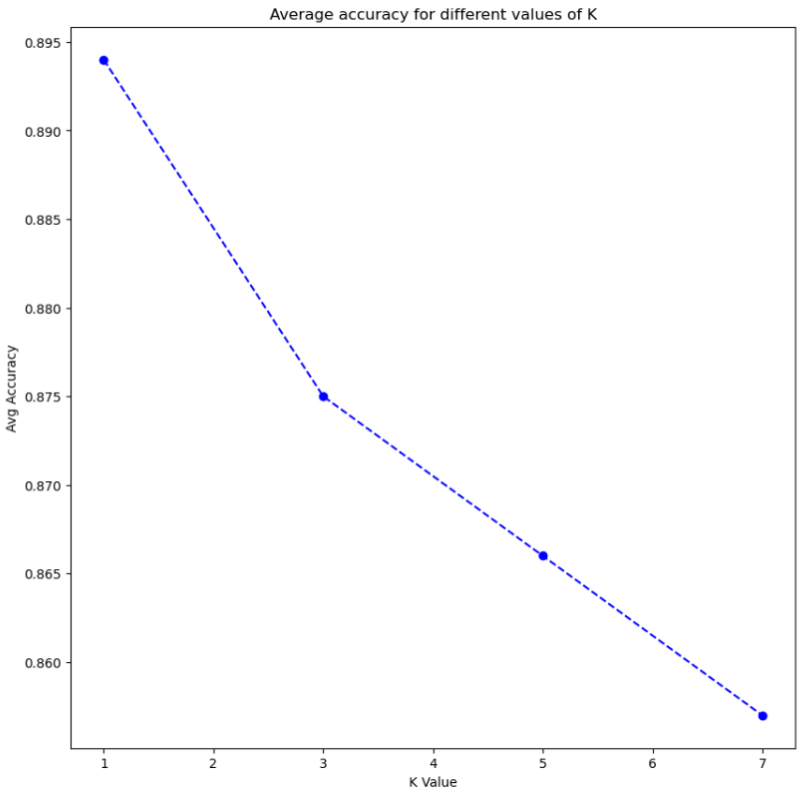


```
ACCURACY FOR ALPHA:0.8 AND K = 7 IS :- 0.872
CONFUSION MATRIX:
[[200   0]
 [ 32  18]]
============
ACCURACY FOR ALPHA:0.85 AND K = 7 IS :- 0.86
CONFUSION MATRIX:
[[200   0]
 [ 35  15]]
============
ACCURACY FOR ALPHA:0.9 AND K = 7 IS :- 0.852
CONFUSION MATRIX:
[[200   0]
 [ 37  13]]
============
ACCURACY FOR ALPHA:0.95 AND K = 7 IS :- 0.844
CONFUSION MATRIX:
[[200   0]
 [ 39  11]]
============
Done for k = 7
======================
```

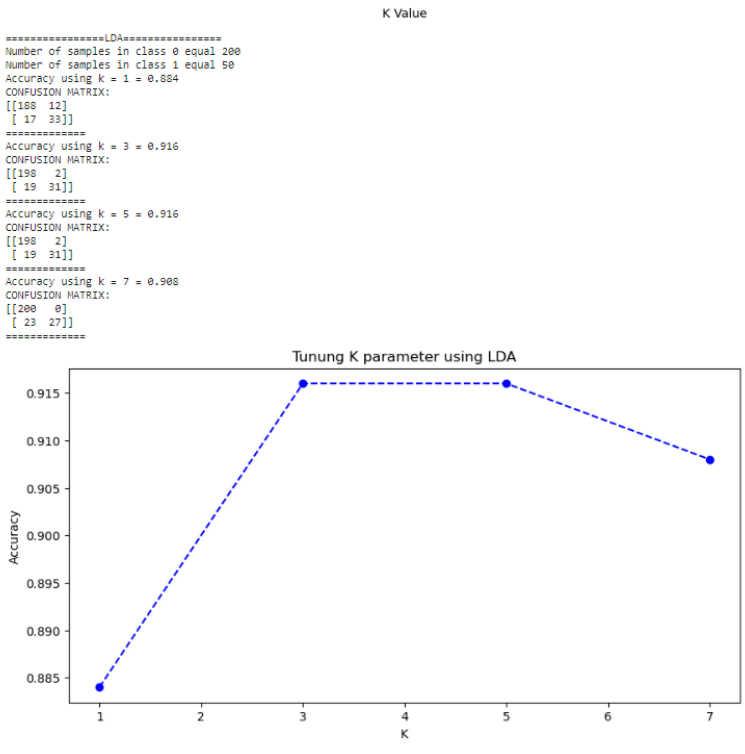


Run All Code




Average accuracy for different values of K
Avg Accuracy
K Value
0.895
0.890
0.885
0.880
0.875
0.870
0.865
0.860
1
2
3
4
5
6
7

LDA:


K Value
==========LDA==========
Number of samples in class 0 equal 200
Number of samples in class 1 equal 50
Accuracy using k = 1 = 0.884
CONFUSION MATRIX:
[[188 12]
[17 33]]
==========
Accuracy using k = 3 = 0.916
CONFUSION MATRIX:
[[198 2]
[19 31]]
==========
Accuracy using k = 5 = 0.916
CONFUSION MATRIX:
[[198 2]
[19 31]]
==========
Accuracy using k = 7 = 0.908
CONFUSION MATRIX:
[[200 0]
[23 27]]
==========
Tunung K parameter using LDA
Accuracy
K
0.915
0.910
0.905
0.900
0.895
0.890
0.885
1
2
3
4
5
6
7

400 Face and 400 Non-face:

PCA

NOTE: we plot failure cases

```
RUNNING PCA AND LDA WITH NONFACES = 400
----------------------------------
----------------------------------
===============PCA===============
Eigenvectors shape = (10304, 10304)
r =  42
U shape = (10304, 42)
projected_training shape = (400, 42)
projected_testing shape = (400, 42)
ACCURACY FOR ALPHA:0.8 AND K = 1 IS :- 0.93
CONFUSION MATRIX:
[[200   0]
 [ 28 172]]
=============
Failure Case
219
Predicted face but actual is non face
```

```
Failure Case
220
```

```
Failure Case
241
Predicted face but actual is non face
```

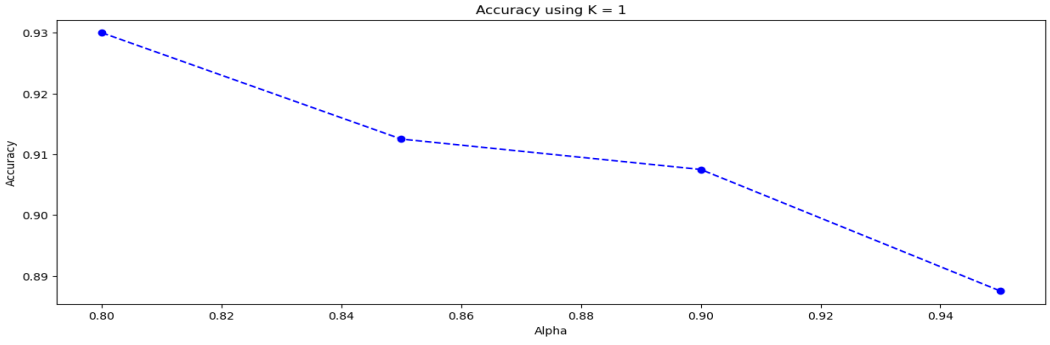
```
Failure Case
```

```
Failure Case
322
Predicted face but actual is non face
```



```
Failure Case
328
Predicted face but actual is non face
```


```
acc = 0.8875
```



Alpha

```
ACCURACY FOR ALPHA:0.8 AND K = 3 IS :- 0.8875
CONFUSION MATRIX:
[[200   0]
 [ 45 155]]
===========
Failure Case
204
Predicted face but actual is non face
<Figure size 1000x1000 with 0 Axes>
```

```
Failure Case
212
Predicted face but actual is non face
```

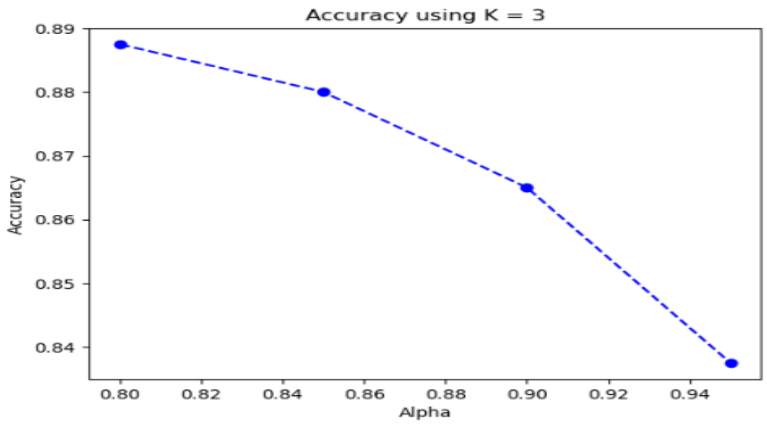
```
Failure Case
384
Predicted face but actual is non face
```

```
Failure Case
385
Predicted face but actual is non face
```

```
Failure Case
386
Predicted face but actual is non face
```

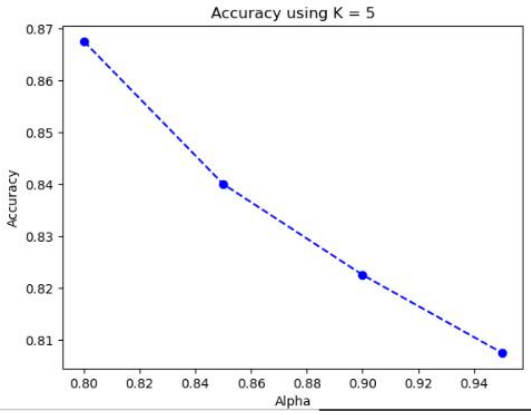
```
Failure Case
386
Predicted face but actual is non face
```

```
acc = 0.8375
Done for k = 3
=======================
```

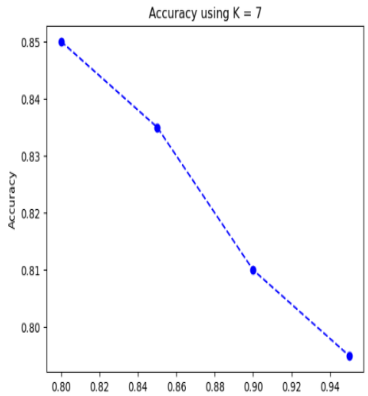


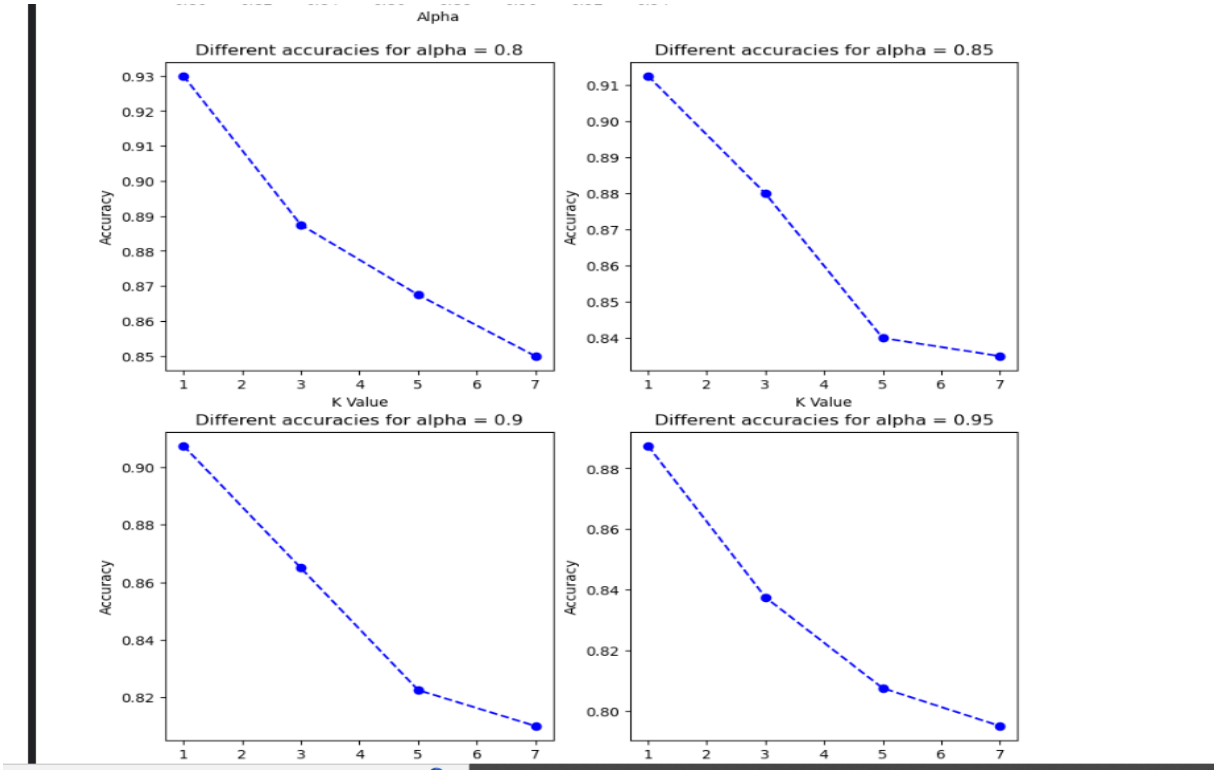
```
Failure Case
386
Predicted face but actual is non face
```

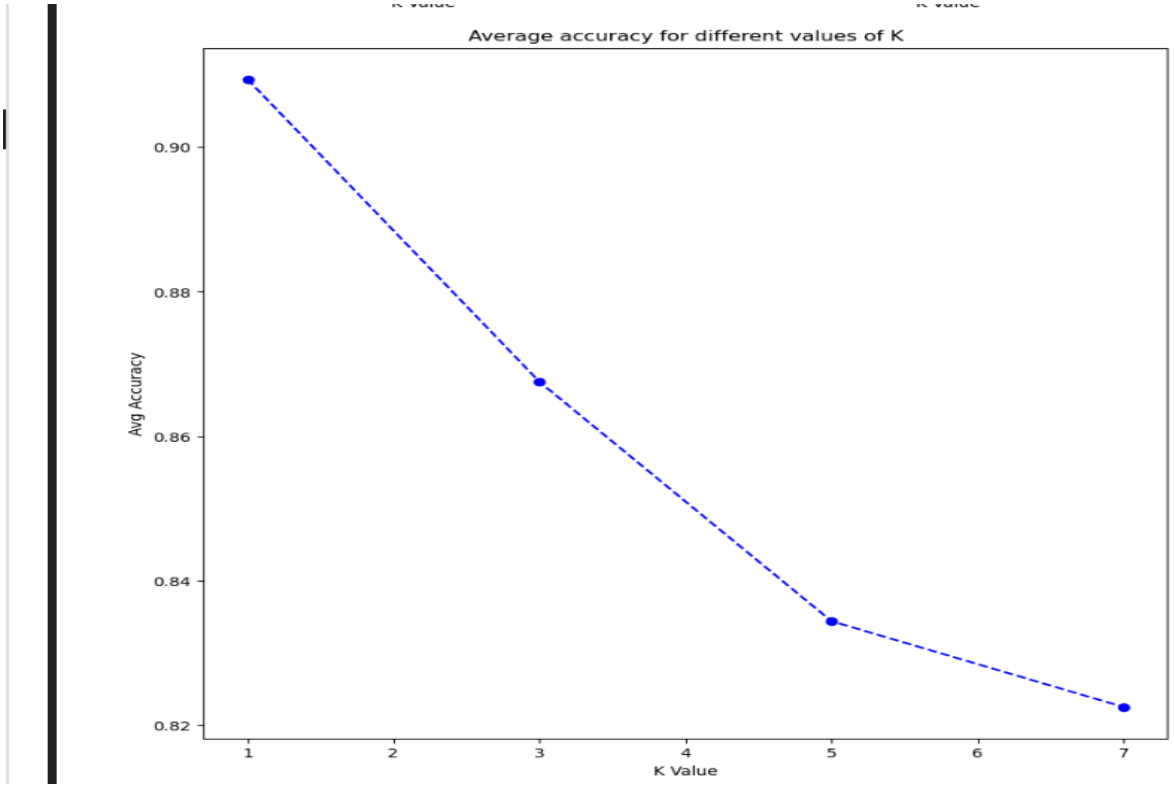
```
acc = 0.8075
Done for k = 5
======================
```



```
acc = 0.795
Done for k = 7
======================
```




Alpha
Different accuracies for alpha = 0.8
Different accuracies for alpha = 0.85
Different accuracies for alpha = 0.9
Different accuracies for alpha = 0.95
Accuracy
K Value


Average accuracy for different values of K
Avg Accuracy
K Value

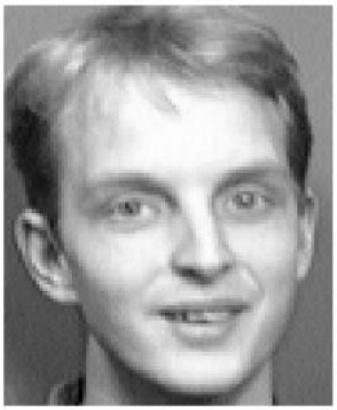
LDA:

K Value

```
================LDA===============
Number of samples in class 0 equal 200
Number of samples in class 1 equal 200
Accuracy using k = 1 = 0.81
CONFUSION MATRIX:
[[166  34]
 [ 42 158]]
============
Failure Case
12
Predicted non face but actual is face
```

```
Failure Case
17
Predicted non face but actual is face
```

```
Predicted non face but actual is face
```



```
Failure Case
201
Predicted face but actual is non face
```

```
Failure Case
17
Predicted non face but actual is face
```

```
Failure Case
18
Predicted non face but actual is face
```

Run All Code

```
Failure Case
382
Predicted face but actual is non face
```

```
Failure Case
386
Predicted face but actual is non face
```

Failure Case
386
Predicted face but actual is non face

acc = 0.8675
Accuracy using k = 5 = 0.8575
CONFUSION MATRIX:
[[187 13]
 [44 156]]
============
Failure Case
73
Predicted non face but actual is face

acc = 0.8575
Accuracy using k = 7 = 0.8575
CONFUSION MATRIX:
[[191   9]
 [48 152]]
============
Failure Case
116
Predicted non face but actual is face

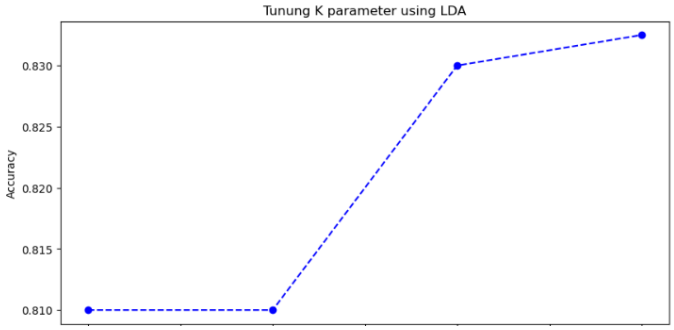
Failure Case
151
Predicted non face but actual is face

Failure Case
386
Predicted face but actual is non face

Failure Case
393
Predicted face but actual is non face

```
Failure Case
393
Predicted face but actual is non face
```

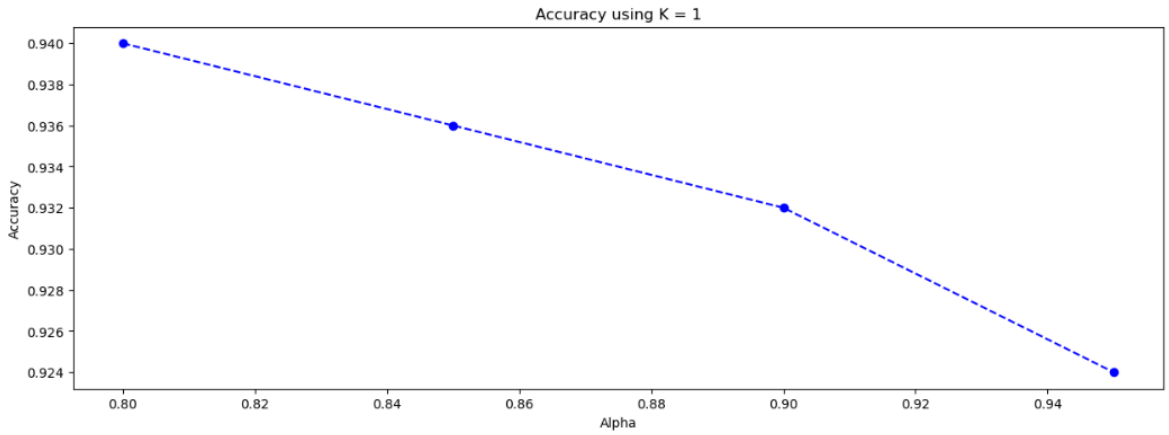
```
acc = 0.8325
```

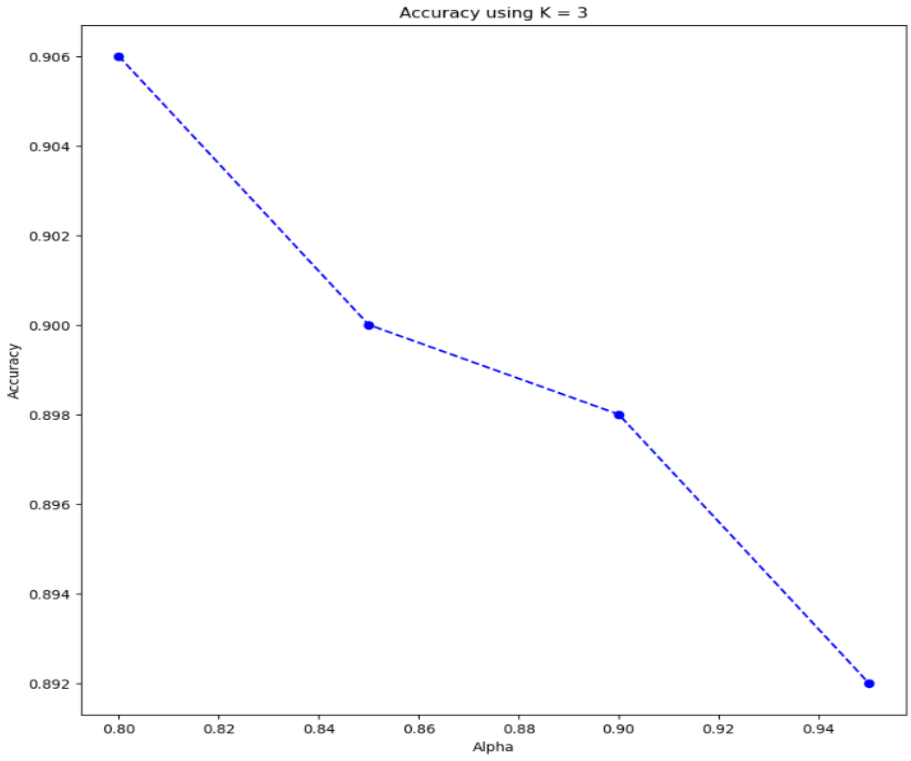


400 Face and 600 Non-face:

```
RUNNING PCA AND LDA WITH NONFACES = 600
----------------------------------------
----------------------------------------
================PCA================
Eigenvectors shape = (10304, 10304)
r =  43
U shape = (10304, 43)
projected_training shape = (500, 43)
projected_testing shape = (500, 43)
ACCURACY FOR ALPHA:0.8 AND K = 1 IS :- 0.94
CONFUSION MATRIX:
[[200   0]
 [ 30 270]]
=============
Eigenvectors shape = (10304, 10304)
r =  70
U shape = (10304, 70)
projected_training shape = (500, 70)
projected_testing shape = (500, 70)
ACCURACY FOR ALPHA:0.85 AND K = 1 IS :- 0.936
CONFUSION MATRIX:
[[200   0]
 [ 32 268]]
=============
Eigenvectors shape = (10304, 10304)
r =  117
U shape = (10304, 117)
projected_training shape = (500, 117)
projected_testing shape = (500, 117)
ACCURACY FOR ALPHA:0.9 AND K = 1 IS :- 0.932
CONFUSION MATRIX:
[[200   0]
 [ 34 266]]
=============
Eigenvectors shape = (10304, 10304)
r =  207
U shape = (10304, 207)
projected_training shape = (500, 207)
projected_testing shape = (500, 207)
ACCURACY FOR ALPHA:0.95 AND K = 1 IS :- 0.924
CONFUSION MATRIX:
[[200   0]
 [ 38 262]]
=============
```

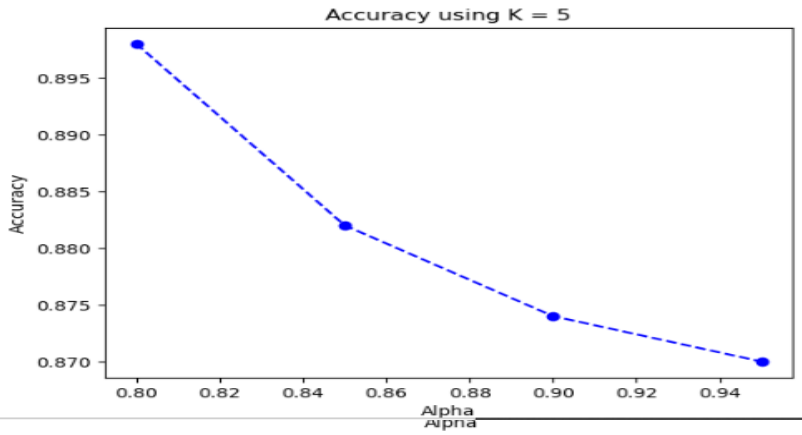
==========


Accuracy using K = 1
Accuracy
Alpha
0.940
0.938
0.936
0.934
0.932
0.930
0.928
0.926
0.924
0.80
0.82
0.84
0.86
0.88
0.90
0.92
0.94


Accuracy using K = 3
Accuracy
Alpha
0.906
0.904
0.902
0.900
0.898
0.896
0.894
0.892
0.80
0.82
0.84
0.86
0.88
0.90
0.92
0.94

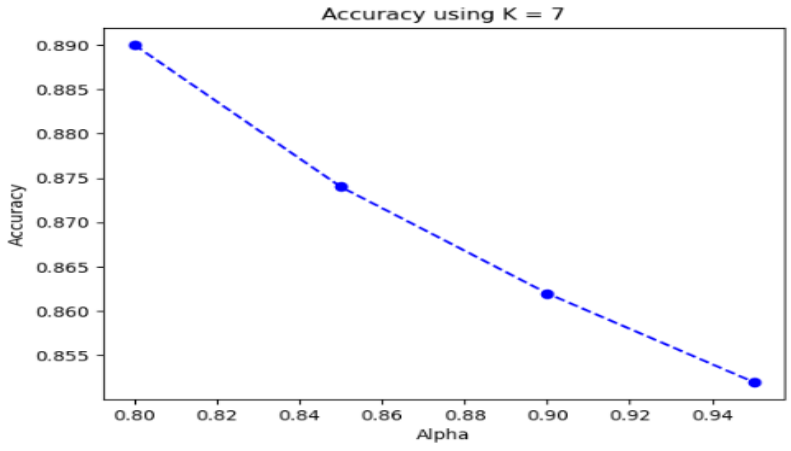
Alpna

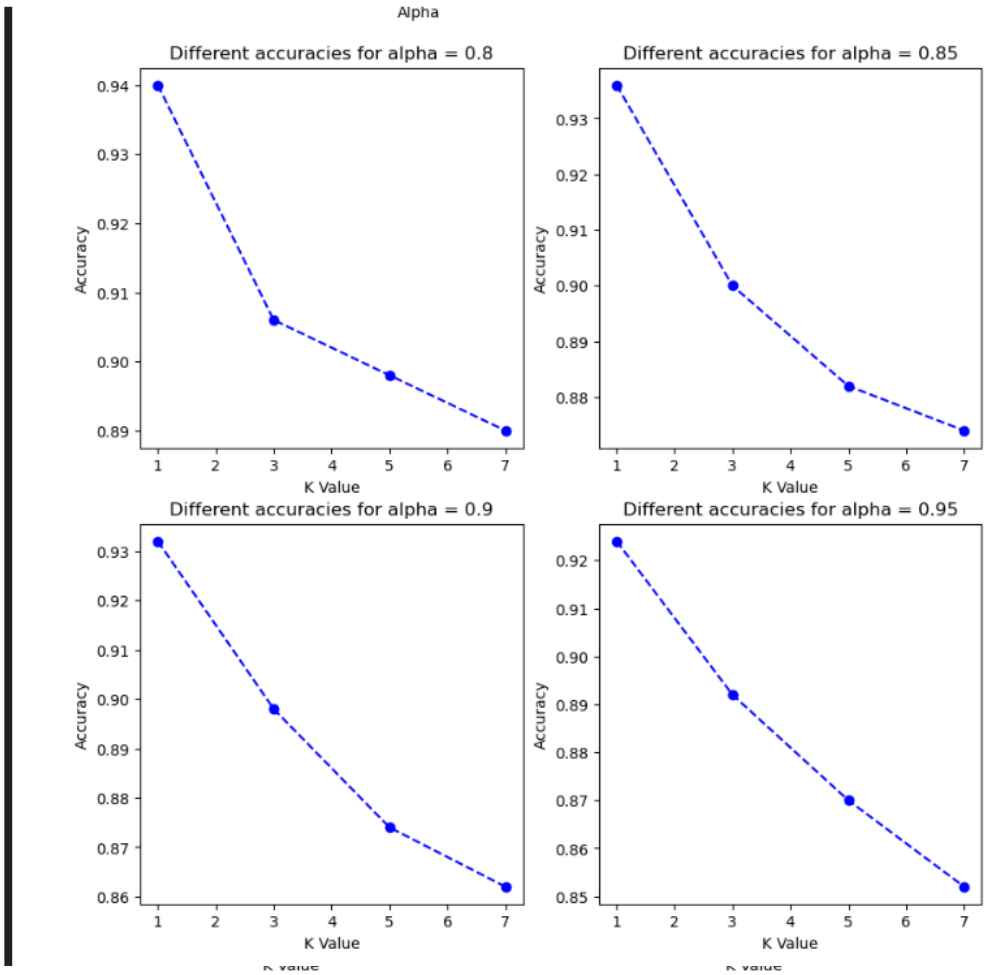
```
ACCURACY FOR ALPHA:0.8 AND K = 5 IS :- 0.898
CONFUSION MATRIX:
[[200   0]
 [ 51 249]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 5 IS :- 0.882
CONFUSION MATRIX:
[[200   0]
 [ 59 241]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 5 IS :- 0.874
CONFUSION MATRIX:
[[200   0]
 [ 63 237]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 5 IS :- 0.87
CONFUSION MATRIX:
[[200   0]
 [ 65 235]]
=============
Done for k = 5
======================
```

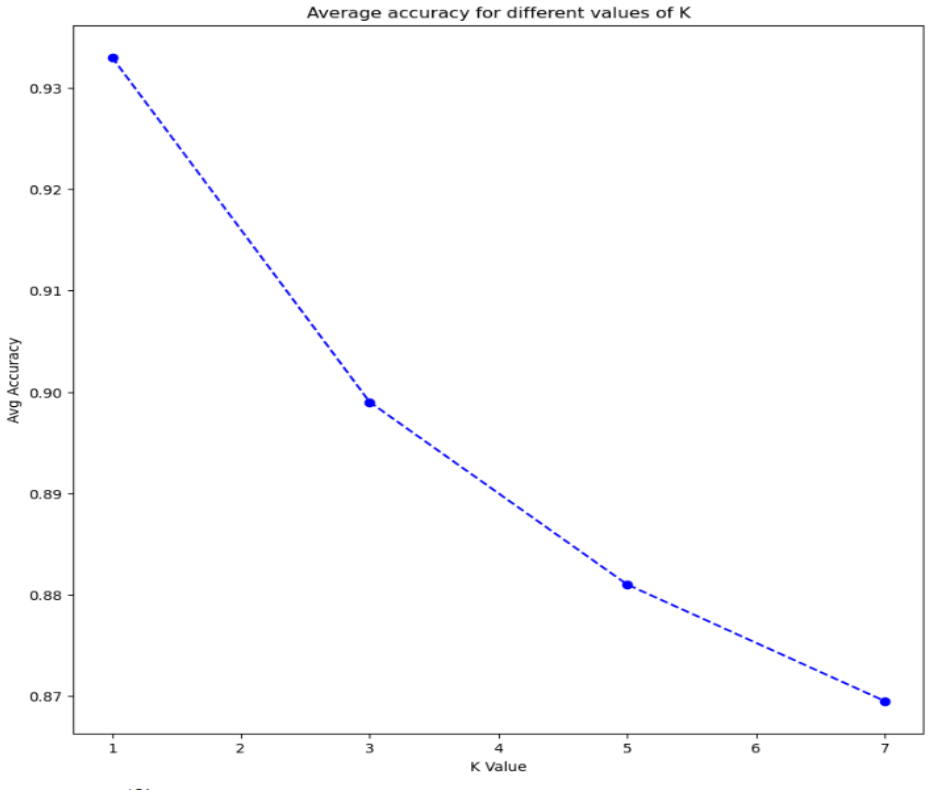


Alpna

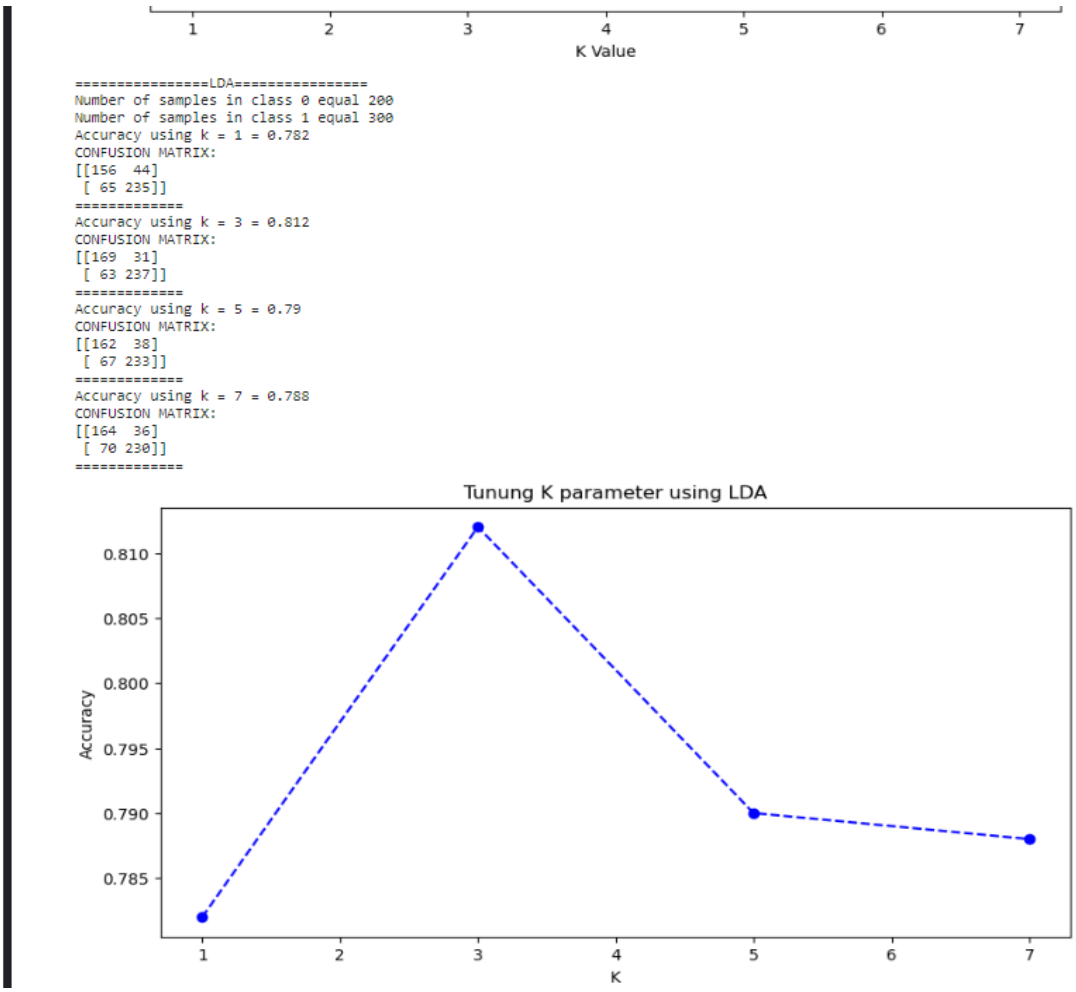
```
ACCURACY FOR ALPHA:0.8 AND K = 7 IS :- 0.89
CONFUSION MATRIX:
[[200   0]
 [ 55 245]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 7 IS :- 0.874
CONFUSION MATRIX:
[[200   0]
 [ 63 237]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 7 IS :- 0.862
CONFUSION MATRIX:
[[200   0]
 [ 69 231]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 7 IS :- 0.852
CONFUSION MATRIX:
[[200   0]
 [ 74 226]]
=============
Done for k = 7
======================
```




Alpha
Different accuracies for alpha = 0.8
Different accuracies for alpha = 0.85
Different accuracies for alpha = 0.9
Different accuracies for alpha = 0.95
Accuracy
K Value


Average accuracy for different values of K
Avg Accuracy
K Value

LDA:



Insights From The previous outputs:

-When the number of non faces increase the accuracy increase

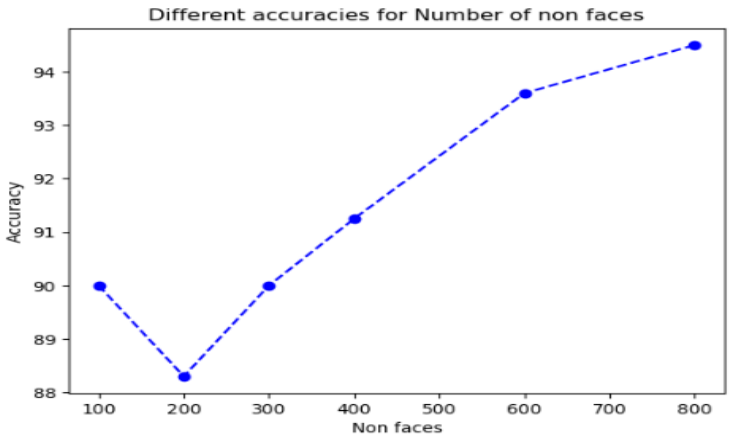
- when we increase the number of non faces it almost see all the data as non faces so the probability to detect it as non face increase so he may detect face as non face

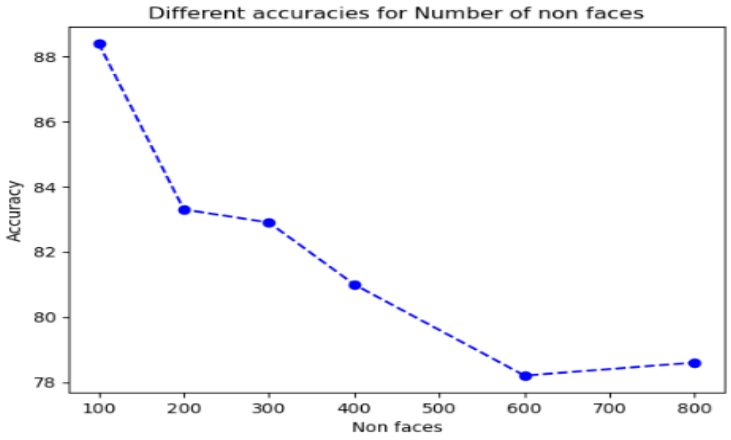
- The most miss identifications due to faces that are closed to camera

- The accuracy of LDA is low due to using dominant eigenvector 1

[39]:

```
pca_accuracies=[90,88.3,90,91.25,93.6,94.5]
lda_accuracies=[81.6,81.3,82,82.57,82.6,81.67]
plt.plot(loops,pca_accuracies,'--bo', label='line with marker')
plt.xlabel("Non faces")
plt.ylabel("Accuracy")
plt.title("Different accuracies for Number of non faces")
plt.show()
plt.plot(loops,lda_accuracies,'--bo', label='line with marker')
plt.xlabel("Non faces")
plt.ylabel("Accuracy")
plt.title("Different accuracies for Number of non faces")
plt.show()
```





Insights From The previous outputs:

-The accuracy measure is deceptive because it is more accustomed to non faces

C.Bonus:

Bonus 1

[28]:
```
def split_data(images,labels):
  X_test = np.ones((1, 10304))
  X_train = np.ones((1, 10304))
  y_train = np.array([])
  y_test = np.array([])
  i=0
  while i < 400:
    train=images[i:i+7]
    train_label=labels[i:i+7]
    test=images[i+7:i+10]
    test_label=labels[i+7:i+10]
    X_train=np.append(X_train,train,axis=0)
    X_test=np.append(X_test,test,axis=0)
    y_train=np.append(y_train,train_label)
    y_test=np.append(y_test,test_label)
    i+=10
  X_train = X_train[1:,]
  X_test = X_test[1:,]
  return X_train,X_test,y_train,y_test
```

[29]:
```
X_train_new,X_test_new,y_train_new,y_test_new=split_data(images,labels)
```

+ Code + Markdown

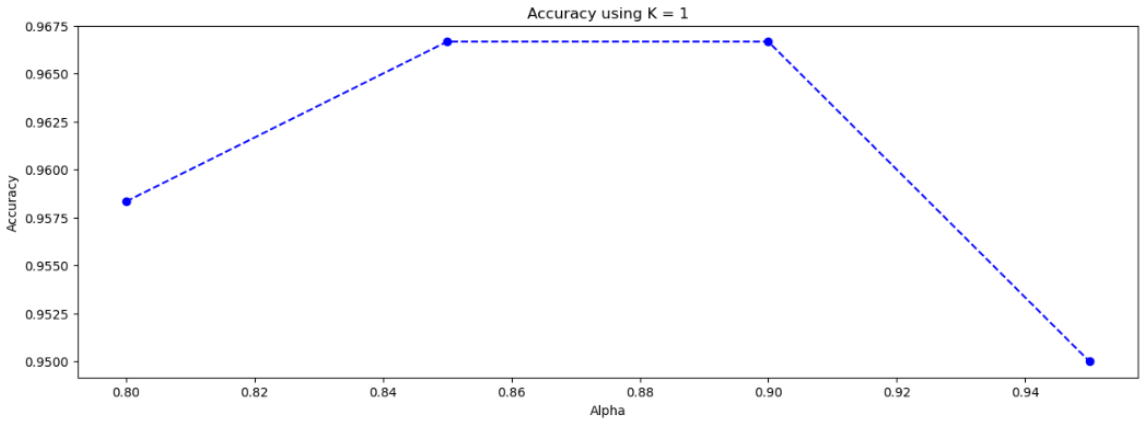
[30]:
```
test_pca(cp.asarray(X_train_new),cp.asarray(X_test_new),cp.asarray(y_train_new),cp.asarray(y_test_new),0)
test_lda(cp.asarray(X_train_new),cp.asarray(X_test_new),cp.asarray(y_train_new),cp.asarray(y_test_new),39,7,0)
```

We create function split_data to split the ratio of data to 0.7 for training and 0.3 for testing

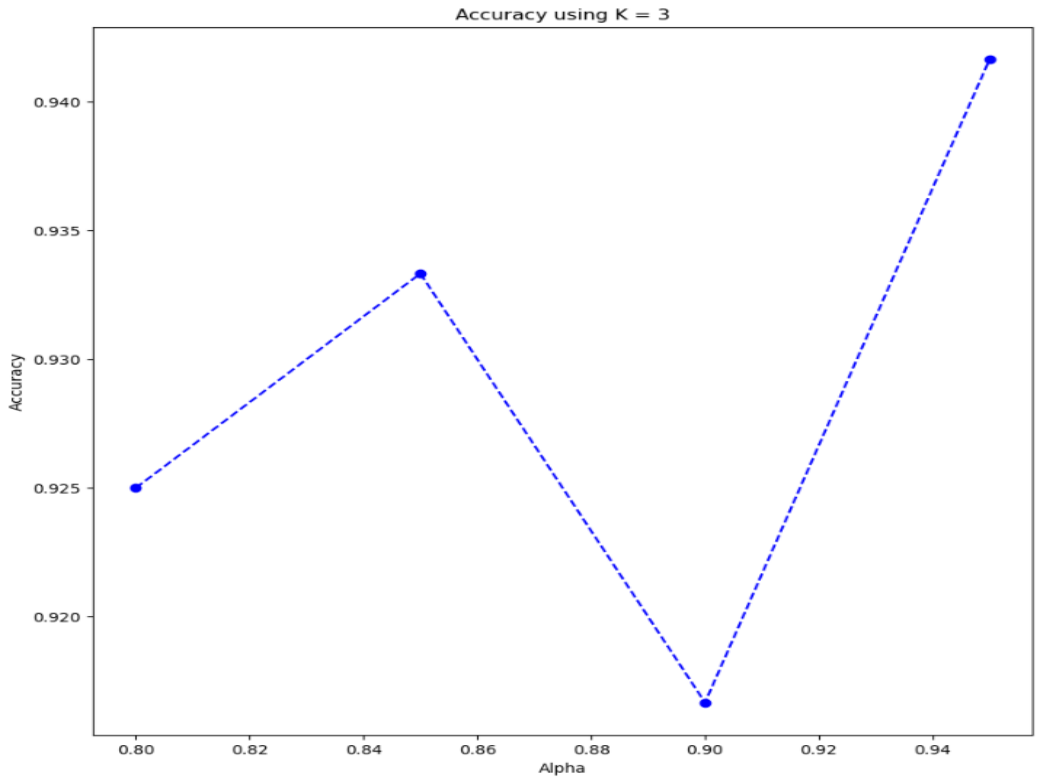
Insights For The following outputs:

At large values of k the accuracy of 70%-30% will be better than 50%-50% because we increase the number of neighbor that vote correctly in training data so the noise will be neglectable

```
Eigenvectors shape = (10304, 10304)
r =  39
U shape = (10304, 39)
projected_training shape = (280, 39)
projected_testing shape = (120, 39)
ACCURACY FOR ALPHA:0.8 AND K = 1 IS :- 0.9583333333333334
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 3]]
=============
Eigenvectors shape = (10304, 10304)
r =  57
U shape = (10304, 57)
projected_training shape = (280, 57)
projected_testing shape = (120, 57)
ACCURACY FOR ALPHA:0.85 AND K = 1 IS :- 0.9666666666666667
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 3]]
=============
Eigenvectors shape = (10304, 10304)
r =  89
U shape = (10304, 89)
projected_training shape = (280, 89)
projected_testing shape = (120, 89)
ACCURACY FOR ALPHA:0.9 AND K = 1 IS :- 0.9666666666666667
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 3]]
=============
Eigenvectors shape = (10304, 10304)
r =  145
U shape = (10304, 145)
projected_training shape = (280, 145)
projected_testing shape = (120, 145)
ACCURACY FOR ALPHA:0.95 AND K = 1 IS :- 0.95
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
```

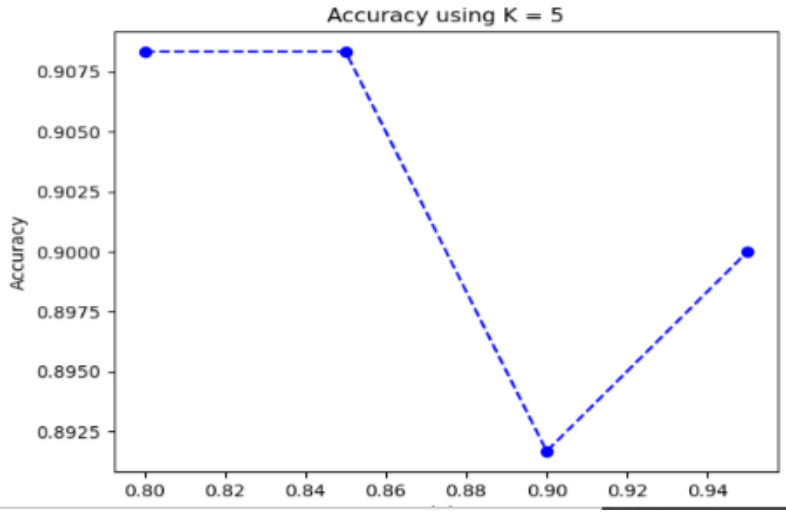


```
ACCURACY FOR ALPHA:0.8 AND K = 3 IS :- 0.925
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 3 IS :- 0.9333333333333333
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 3 IS :- 0.9166666666666666
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 2 0]
 [0 0 0 ... 0 0 2]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 3 IS :- 0.9416666666666667
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
Done for k = 3
======================
```

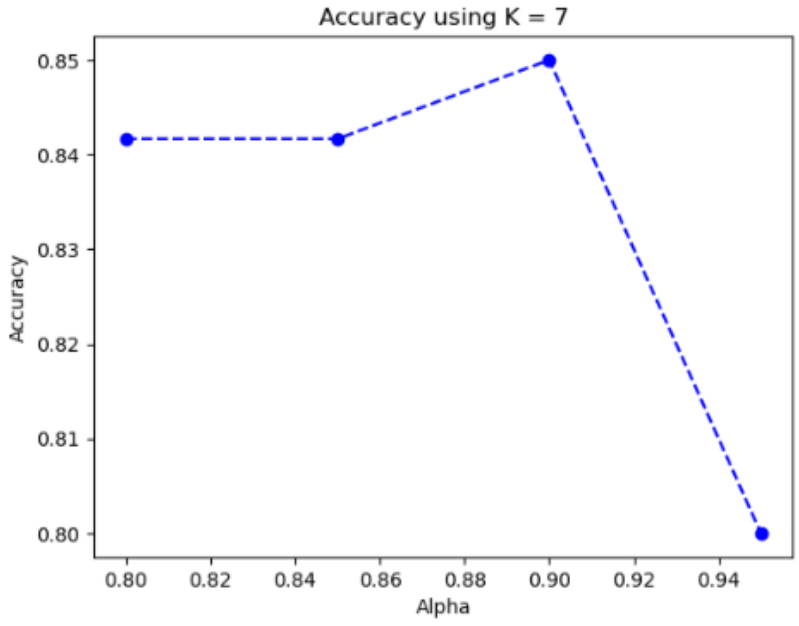


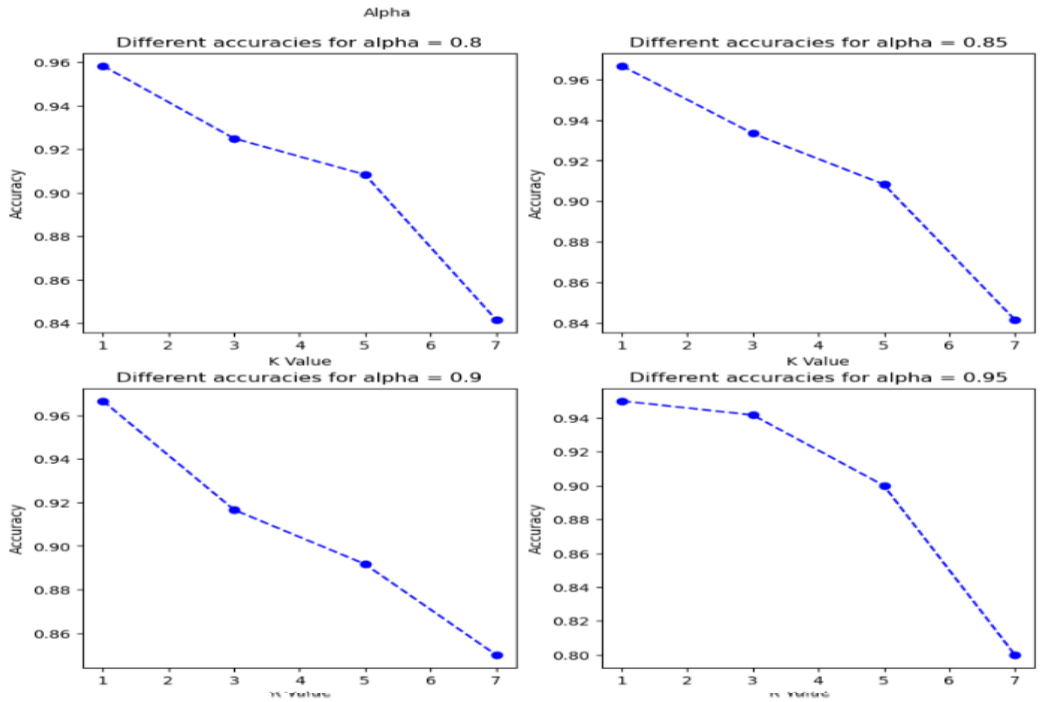
Alpha

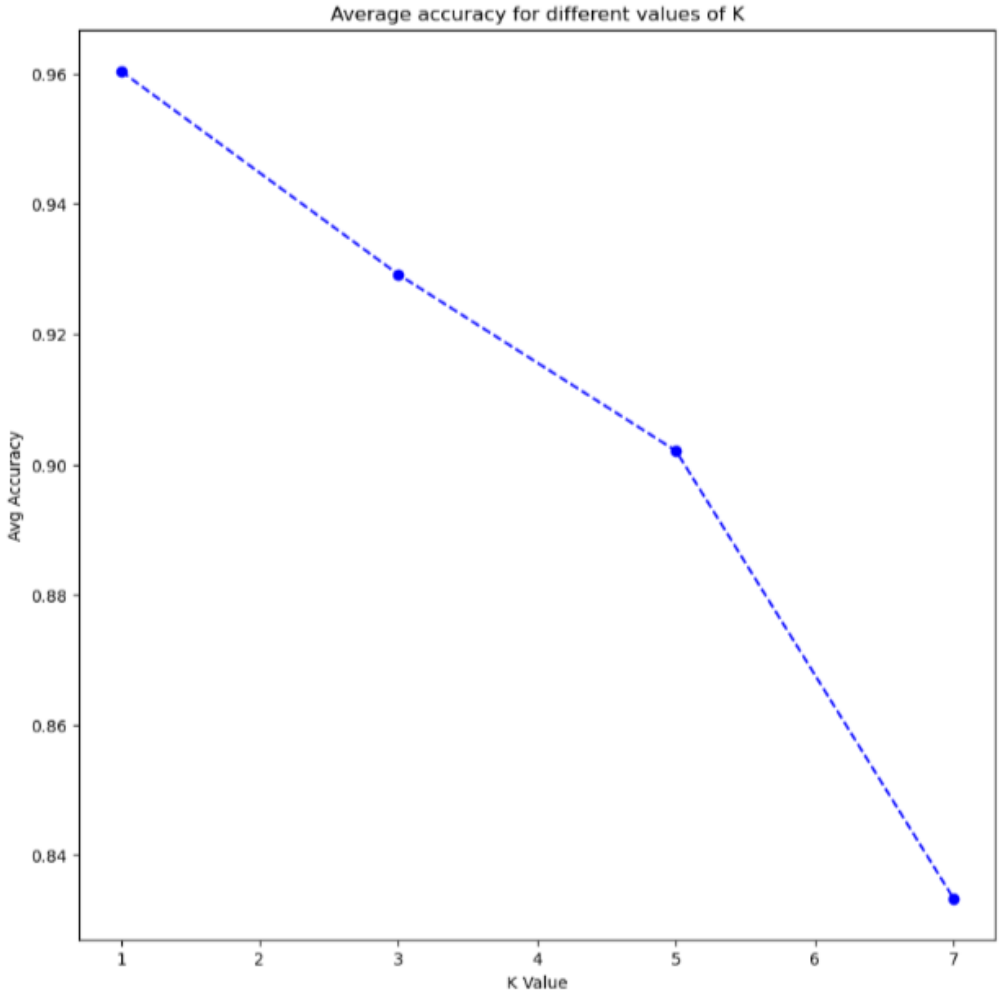
```
ACCURACY FOR ALPHA:0.8 AND K = 5 IS :- 0.9083333333333333
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 1]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 5 IS :- 0.9083333333333333
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 1]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 5 IS :- 0.8916666666666667
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 1]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 5 IS :- 0.9
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 1]]
=============
Done for k = 5
====================
```



```
ACCURACY FOR ALPHA:0.8 AND K = 7 IS :- 0.8416666666666667
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
ACCURACY FOR ALPHA:0.85 AND K = 7 IS :- 0.8416666666666667
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
ACCURACY FOR ALPHA:0.9 AND K = 7 IS :- 0.85
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
ACCURACY FOR ALPHA:0.95 AND K = 7 IS :- 0.8
CONFUSION MATRIX:
[[2 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 2]]
=============
Done for k = 7
======================
```

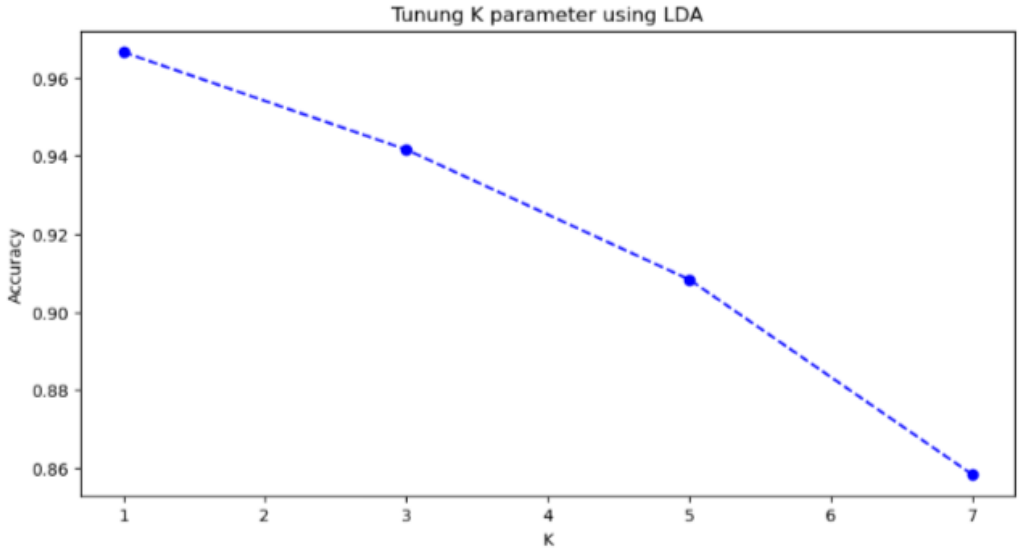



Alpha
Different accuracies for alpha = 0.8
Different accuracies for alpha = 0.85
Different accuracies for alpha = 0.9
Different accuracies for alpha = 0.95
Accuracy
K Value


Average accuracy for different values of K
Avg Accuracy
K Value

K Value

```
Accuracy using k = 1 = 0.9666666666666667
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 3]]
============
Accuracy using k = 3 = 0.9416666666666667
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 3]]
============
Accuracy using k = 5 = 0.9083333333333333
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 2 0]
 [0 0 0 ... 0 0 2]]
============
Accuracy using k = 7 = 0.8583333333333333
CONFUSION MATRIX:
[[3 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 2 0]
 [0 0 0 ... 0 0 2]]
============
```



BONUS 2:

We used 3 variations for PCA which are

[Randomized PCA -Incremental PCA-Kernel PCA]

And one variation for LDA which is [Regularized LDA]

.....

PCA:

Bonus 2

[31]:
```
def classify_knn(X_train,X_test,y_train,y_test,k):
  model=KNeighborsClassifier(n_neighbors=k)
  model.fit(X_train, y_train)
  test_pred = model.predict(X_test)
  acc=accuracy_score(y_test, test_pred)
  print("Accuracy using k = "+str(k)+" = "+str(acc*100))
```

Randomized PCA

The classical PCA uses the low-rank matrix approximation to estimate the principal components. However, this method becomes costly and makes the whole process difficult to scale, for large datasets.

By randomizing how the singular value decomposition of the dataset happens, we can approximate the first K principal components quickly than classical PCA.

[32]:
```
from sklearn.decomposition import PCA
rpca = PCA(n_components=91,svd_solver='randomized')
X_train_rpca = rpca.fit_transform(X_train)
X_test_rpca = rpca.transform(X_test)
```

[33]:
```
Ks=[1,3,5,7]
for k in Ks:
  classify_knn(X_train_rpca,X_test_rpca,y_train,y_test,k)
```

```
Accuracy using k = 1 = 94.0
Accuracy using k = 3 = 84.0
Accuracy using k = 5 = 81.5
Accuracy using k = 7 = 74.0
```

Incremental PCA

Incremental PCA can be used when the dataset is too large to fit in the memory.Here we split the dataset into mini-batches where each batch can fit into the memory and then feed it one mini-batch at a moment to the IPCA algorithm.

+ Code + Markdown

[34]:
```
from sklearn.decomposition import IncrementalPCA
ipca = IncrementalPCA(n_components=91,batch_size=100)
X_train_ipca = ipca.fit_transform(X_train)
X_test_ipca = ipca.transform(X_test)
```

[35]:
```
Ks=[1,3,5,7]
for k in Ks:
  classify_knn(X_train_ipca,X_test_ipca,y_train,y_test,k)
```

```
Accuracy using k = 1 = 94.5
Accuracy using k = 3 = 84.0
Accuracy using k = 5 = 81.5
Accuracy using k = 7 = 73.5
```

Kernel PCA

Kernel PCA is a technique which uses the so-called kernel trick and projects the linearly inseparable data into a higher dimension where it is linearly separable

+ Code + Markdown

[36]:
```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components=91, kernel='linear')
X_train_kpca = kpca.fit_transform(X_train)
X_test_kpca = kpca.transform(X_test)
```

[37]:
```
Ks=[1,3,5,7]
for k in Ks:
  classify_knn(X_train_kpca,X_test_kpca,y_train,y_test,k)
```

```
Accuracy using k = 1 = 94.5
Accuracy using k = 3 = 84.0
Accuracy using k = 5 = 81.5
Accuracy using k = 7 = 73.5
```

LDA:

- Regularized LDA

RDA is similar to LDA but introduces regularization to the covariance matrix in order to improve the performance of the classifier when there are more variables than observations or when the variables are highly correlated.

[ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
# Create the model with regularized lda
#auto means Regularized LDA and degree of shrinkage based on data
#1 --> we can set the shrinkage to 1 to give high shrinkage and in our data set accuracy will be 89%
lda_regularized = LinearDiscriminantAnalysis(solver='eigen', shrinkage='auto')
# Fit the model to the training data
lda_regularized.fit(X_train, y_train)
# Predict the class labels for the testing data
y_pred = lda_regularized.predict(X_test)
# get the accuracy of regularized lda
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

We will compare the accuracy of normal PCA with the previous PCAs and The Normal LDA with Regularized LDA when n_components=77

1. Normal PCA & Randomized PCA:

k	Normal PCA Accuracy [alpha 0.9]	Randomized PCA Accuracy
1	94.5	94.5
3	85.0	85.0
5	81.5	81.5
7	75.5	76.5

2. Normal PCA & Incremental PCA:

k	Normal PCA Accuracy [alpha 0.9]	Incremental PCA Accuracy
1	94.5	94.5
3	85.0	85.0
5	81.5	81.5
7	75.5	75.5

We can see that they have the same accuracy because entire dataset can fit into memory and the batch size used in incremental PCA is set to be equal to the size of the entire dataset
Time complexity of normal PCA O(n^3)
Time complexity of incremental PCA O(m*n^2)

3. Normal PCA & kernel PCA:

k	Normal PCA Accuracy [alpha 0.9]	Kernel PCA Accuracy
1	94.5	94.5
3	85.0	85.0
5	81.5	82.5
7	75.5	75.5

4. Normal LDA & Regularized LDA:

Regularized LDA Accuracy
98.5

We get from the previous value that the regularized LDA has better accuracy than normal LDA because regularized LDA can help to reduce the variance of the estimated covariance matrix, which can improve the accuracy of the classifier
Normal LDA and Regularized have same time complexity they differ according to size and complexity of dataset