ASSIGNMENT 1 REPORT

Student1: Yasmin Ashraf Mohamed Eshra

Student2: Habiba Hossam Bakr

Student3: Salma Mohamed Hasan Barakat

1- Downloading the Dataset and Understanding the Format:

```
[15] from numpy.linalg import eig
     from numpy import linalg as LA
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score
     from sklearn.preprocessing import StandardScaler
     from random import seed
     from random import randrange
     from math import sqrt
     from sklearn.model_selection import train_test_split
     import matplotlib.pyplot as plt
     from sklearn.metrics import confusion_matrix
     import seaborn as sns
```

```
## Face dataset
from google.colab.patches import cv2_imshow
from PIL import Image
from numpy import asarray
import cv2
import numpy as np

numpydata = [[[0 for k in range(10304)] for j in range(10)]for i in range(40)]
## array of 40 x 10 = 400 cell each = vector of 10304 element
for i in range(1,41):
  for j in range(1,11):
    path = "/content/s"+str(i)+"/"+str(j)+".pgm"
    img = cv2.imread(path,0)  # 0 to indicate greyscale image
    numpydata[i-1][j-1] = asarray(img).flatten() ## flatten to convert 2D array to a vector
```

2- Generating the Data Matrix and the Label vector:

```
D = np.array(np.zeros((400, 10304)))  ## initialization of the matrix 400 x 10304
k = 0
for i in range(40):
  for j in range(10):
    D[k] =  np.asmatrix(numpydata[i][j]) ## take each cell image from numpydata to be a row in the matrix
    k = k + 1
y = np.array(np.zeros((400, 1)))
end = 0
for i in range(40):  ## labeled vector 400 x 1 , range of values 1:40
  start = end
  end = 10 + 10*i
  y[start:end] = i+1
```

3- Splitting the Dataset into Training and Test sets:

```
# odd rows for training 200x10304
# even  rows for testing 200x10304
training=np.array(np.zeros((200,10304)))
testing=np.array(np.zeros((200,10304)))
ytesting=np.array(np.zeros((200,1)))
ytraining=np.array(np.zeros((200,1)))
j,k=0,0
for i in range (400):
  if (i%2==0):
    testing[j]=D[i]
    ytesting[j]=y[i]
    j=j+1
  else:
    training[k]=D[i]
    ytraining[k]=y[i]
    k=k+1
```

4- Classification using PCA:

a- Computing the projection matrix U:

```
# computing the mean:
meanVect = np.mean(training, axis=0)
# computing centered data:
zTrain = training - meanVect
# computing covariance matrix:
# covariance = zTranspose*z / n
cov_matrix = np.dot(zTrain.T, zTrain)/10304
# calculating eigen values and eigen vector:
values, vector = np.linalg.eigh(cov_matrix)
indx=values.argsort()[::-1]   #[::-1] to take all the list in reverse order
values = values[indx]
vector = vector[:,indx]
# computing the fraction of total variance:
total_eigenvalues = sum(values)
r = np.array([0,0,0,0])
alpha = np.array([0.8, 0.85, 0.9, 0.95])
for x in range(4):
  fraction = 0
  for i in range(10304):
    fraction += values[i]
    if fraction/total_eigenvalues >= alpha[x]:
      r[x]=i+1
      break
# # reduced basis:
U1 = vector[:,:r[0]]
U2 = vector[:,:r[1]]
U3 = vector[:,:r[2]]
U4 = vector[:,:r[3]]
```

b- Projecting the training set, and test sets separately using the same projection matrix:

```
#b) projecting the training set:
ProjTrain1 = np.dot(zTrain, U1)
ProjTrain2 = np.dot(zTrain, U2)
ProjTrain3 = np.dot(zTrain, U3)
ProjTrain4 = np.dot(zTrain, U4)
# projecting the test set:
# computing the mean of test:
meanVectTest = np.mean(testing, axis=0)
# computing centered data:
zTest = testing - meanVectTest
ProjTest1 = np.dot(zTest, U1)
ProjTest2 = np.dot(zTest, U2)
ProjTest3 = np.dot(zTest, U3)
ProjTest4 = np.dot(zTest, U4)
```

c- Using a simple classifier (first Nearest Neighbor to determine the class labels):

d- Reporting Accuracy for every value of alpha separately:

```
accuracy_pca = np.array([0,0,0,0])
knn_pca = KNeighborsClassifier(n_neighbors=1,weights='distance')
knn_pca.fit(ProjTrain1, ytraining)
pca_pred1 = knn_pca.predict(ProjTest1)
accuracy_pca1 = accuracy_score(ytesting, pca_pred1)

knn_pca.fit(ProjTrain2, ytraining)
pca_pred2 = knn_pca.predict(ProjTest2)
accuracy_pca2 = accuracy_score(ytesting, pca_pred2)

knn_pca.fit(ProjTrain3, ytraining)
pca_pred3 = knn_pca.predict(ProjTest3)
accuracy_pca3 = accuracy_score(ytesting, pca_pred3)

knn_pca.fit(ProjTrain4, ytraining)
pca_pred4 = knn_pca.predict(ProjTest4)
accuracy_pca4 = accuracy_score(ytesting, pca_pred4)

print("At alpha = 0.8, Accuracy:", accuracy_pca1)
print("At alpha = 0.85, Accuracy:", accuracy_pca2)
print("At alpha = 0.9, Accuracy:", accuracy_pca3)
print("At alpha = 0.95, Accuracy:", accuracy_pca4)
```

```
At alpha = 0.8, Accuracy: 0.93
At alpha = 0.85, Accuracy: 0.935
At alpha = 0.9, Accuracy: 0.94
At alpha = 0.95, Accuracy: 0.935
```

e- Finding a relation between alpha and classification accuracy:
as alpha increases, the accuracy increases, till reaching a specific alpha (0.95) where accuracy begins to decrease slightly.

5- Classification Using LDA:

a-

```
#class data
classlist=[]
for i in range (201):
  if (i%5==0 and i!=0):
    classlist.append(training[i-5:i,:])
sb=np.zeros((10304,10304))
sw=np.zeros((10304,10304))
b=np.zeros((1,10304))
overall_mean=np.mean(training,axis=0) #mean of training (μ) 1*10304
for i in range (40):
  mean=np.mean(classlist[i],axis=0)          # mean for class i (μk) 1*10304
  #sb
  b= mean-overall_mean                       #(μk-μ) 1*10304
  b.resize(10304,1)
  btrans=np.transpose(b)                 # 1*10304
  sb+=5*(b@btrans)                         #(10304*1)*(1*10304)=10304*10304
  #sw
  z=classlist[i]-mean                          #centralized data for class i
  ztrans= np.transpose(z)
  sw+=ztrans@z
#get sw inverse
sinv=np.linalg.inv(sw)
# get sinv*sb
x=sinv@sb
#get eigenvalues and eigenvectors
eigenvalues,eigenvectors=np.linalg.eigh(x)
idx=eigenvalues.argsort()[::-1]   #[::-1] to take all the list in reverse order
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]
projection=eigenvectors[:,0:39] # take first 39 eignvectors U 10304*39
```

b- Projecting the training set, and test sets separately using the same projection matrix U:

```
projection=eigenvectors[:,0:39] # take first 39 eignvectors U 10304*39
#project training set using projection matrix
ztraining=training-overall_mean #200x10304
projecttrain=ztraining@projection #200x39
ztesting=testing-overall_mean
projecttest=ztesting@projection #200x39
```

c- Using a simple classifier (first Nearest Neighbor to determine the class labels):

```
#first Nearest Neighbor to determine the class labels
knn_lda = KNeighborsClassifier(n_neighbors=1,weights='distance')
knn_lda.fit(projecttrain, ytraining)
lda_pred = knn_lda.predict(projecttest)
accuracy_lda = accuracy_score(ytesting, lda_pred)
```

d- Reporting accuracy for the multiclass LDA:

```
print("Accuracy of LDA k=1 :\n", accuracy_lda*100)
```

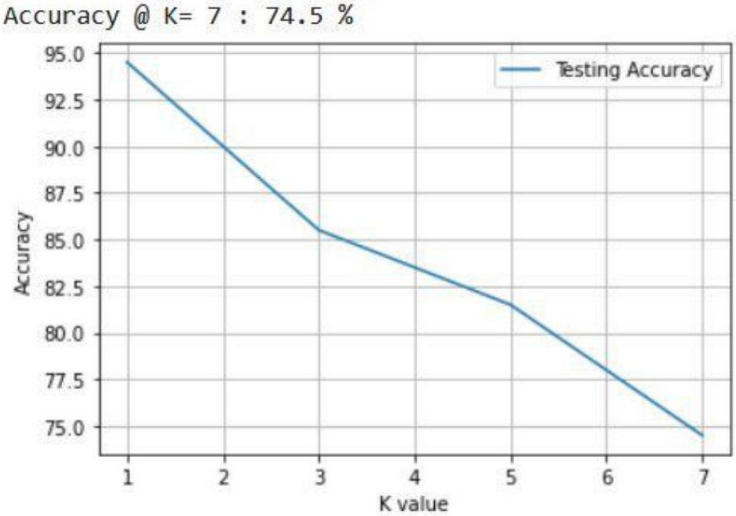
```
Accuracy of LDA k=1 :
 94.5
```

e- Comparing results with that of PCA:
accuracy of LDA is slightly higher than that of PCA, where the highest accuracy of PCA was 94% at alpha = 0.9.

6- Classifier Tuning:

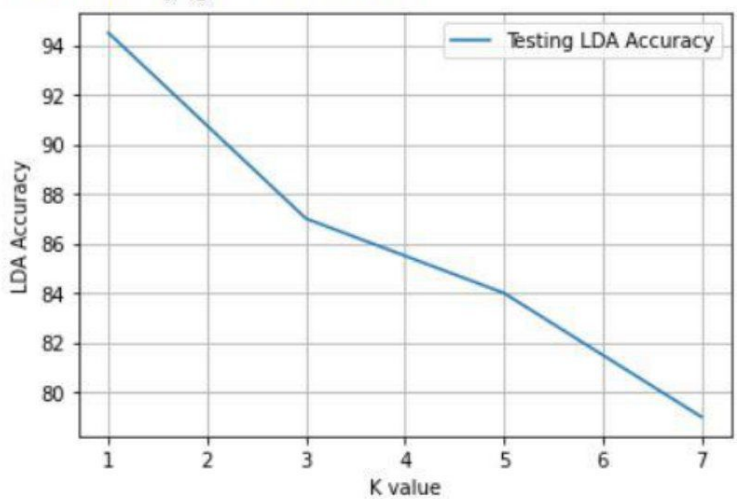
```
x_train = training
y_train = ytraining.ravel()
x_test = testing
y_test = ytesting.ravel()

# Loop over K values
knn_array = [1,3,5,7]
train_accuracy = np.empty(len(knn_array))
test_accuracy = np.empty(len(knn_array))
test_accuracyLDA = np.empty(len(knn_array))
test_accuracyPCA =  np.zeros((4,4))

for i, k in enumerate(knn_array):
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(x_train, y_train)
  y_pred = knn.predict(x_test)
  test_accuracy[i] = accuracy_score(y_test, y_pred)*100

# Generate plot  KNN
for i in range(4):
  print("Accuracy @ K=",knn_array[i],":", test_accuracy[i],"%")
plt.plot(knn_array, test_accuracy, label = 'Testing Accuracy')
plt.legend()
plt.xlabel('K value')
plt.ylabel('Accuracy')
plt.grid()
plt.show()
```

```
Accuracy @ K= 1 : 94.5 %
Accuracy @ K= 3 : 85.5 %
Accuracy @ K= 5 : 81.5 %
Accuracy @ K= 7 : 74.5 %
```



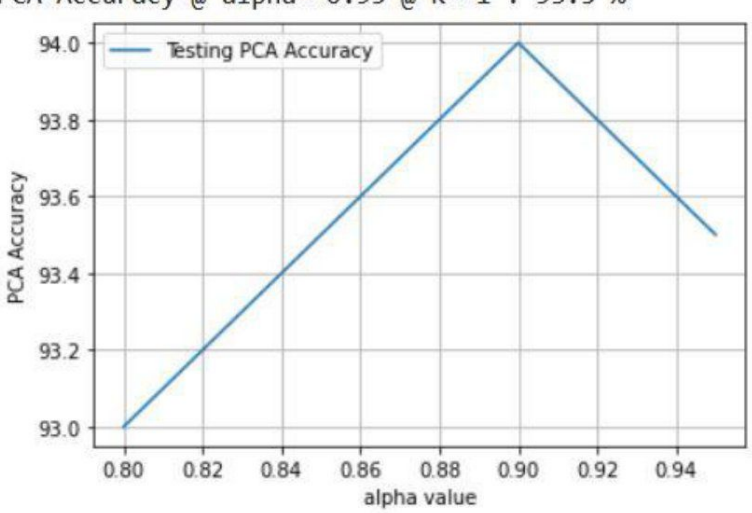
```
for i, k in enumerate(knn_array):
  knnLDA = KNeighborsClassifier(n_neighbors=k)
  knnLDA.fit(projecttrain, ytraining.ravel())
  y_predLDA = knnLDA.predict(projecttest)
  test_accuracyLDA[i] = accuracy_score(ytesting.ravel(), y_predLDA)*100

# Generate plot  LDA
for i in range(4):
  print("LDA Accuracy @ K=",knn_array[i],":", test_accuracyLDA[i],"%")
plt.plot(knn_array, test_accuracyLDA, label = 'Testing LDA Accuracy')
plt.legend()
plt.xlabel('K value')
plt.ylabel('LDA Accuracy')
plt.grid()
plt.show()
```

```
LDA Accuracy @ K= 1 : 94.5 %
LDA Accuracy @ K= 3 : 87.0 %
LDA Accuracy @ K= 5 : 84.0 %
LDA Accuracy @ K= 7 : 79.0 %
```

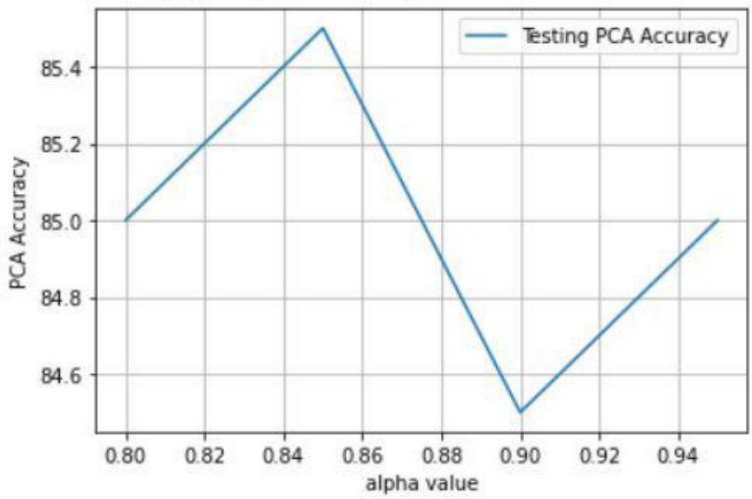


```
for i, k in enumerate(knn_array):
  knnPCA = KNeighborsClassifier(n_neighbors=k)
  knnPCA.fit(ProjTrain1, ytraining.ravel())
  PCApred1 = knnPCA.predict(ProjTest1)
  test_accuracyPCA[i,0] = accuracy_score(ytesting, PCApred1)*100

  knnPCA = KNeighborsClassifier(n_neighbors=k)
  knnPCA.fit(ProjTrain2, ytraining.ravel())
  PCApred2 = knnPCA.predict(ProjTest2)
  test_accuracyPCA[i,1] = accuracy_score(ytesting, PCApred2)*100

  knnPCA = KNeighborsClassifier(n_neighbors=k)
  knnPCA.fit(ProjTrain3, ytraining.ravel())
  PCApred3 = knnPCA.predict(ProjTest3)
  test_accuracyPCA[i,2] = accuracy_score(ytesting, PCApred3)*100

  knnPCA = KNeighborsClassifier(n_neighbors=k)
  knnPCA.fit(ProjTrain4, ytraining.ravel())
  PCApred4 = knnPCA.predict(ProjTest4)
  test_accuracyPCA[i,3] = accuracy_score(ytesting, PCApred4)*100

# Generate plot  PCA
for i in range(4): #k
  for j in range(4):
    print("PCA Accuracy @ alpha=",alpha[j],"@ K=",knn_array[i],":", test_accuracyPCA[i][j],"%")
  plt.plot(alpha,test_accuracyPCA[i,:], label = 'Testing PCA Accuracy')
  plt.legend()
  plt.xlabel('alpha value')
  plt.ylabel('PCA Accuracy')
  plt.grid()
  plt.show()
```

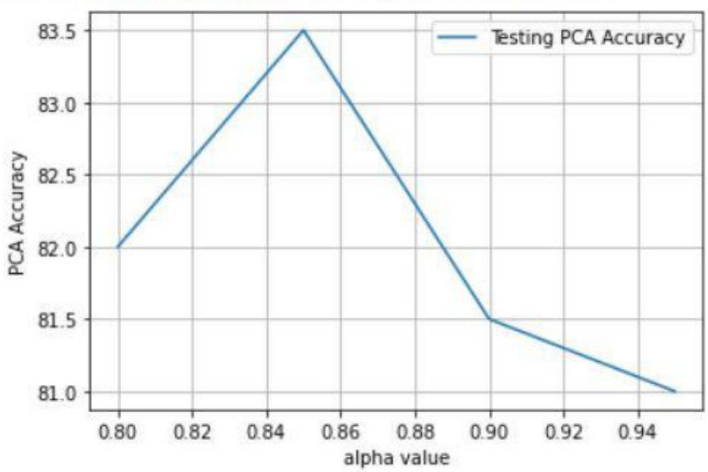
```
PCA Accuracy @ alpha= 0.8 @ K= 1 : 93.0 %
PCA Accuracy @ alpha= 0.85 @ K= 1 : 93.5 %
PCA Accuracy @ alpha= 0.9 @ K= 1 : 94.0 %
PCA Accuracy @ alpha= 0.95 @ K= 1 : 93.5 %
```



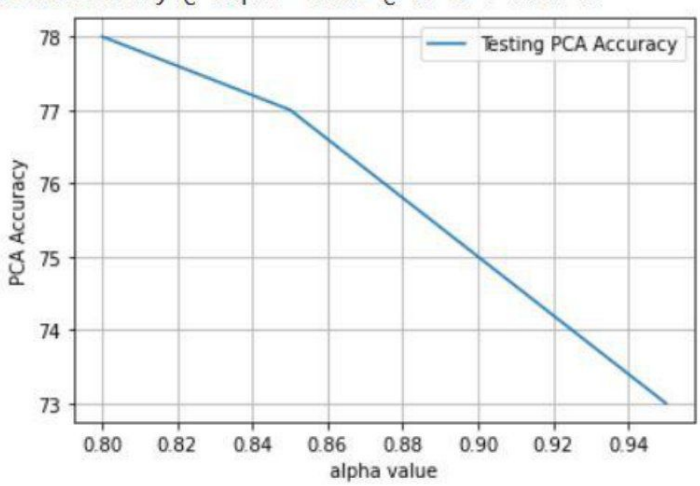
```
PCA Accuracy @ alpha= 0.8 @ K= 3 : 85.0 %
PCA Accuracy @ alpha= 0.85 @ K= 3 : 85.5 %
PCA Accuracy @ alpha= 0.9 @ K= 3 : 84.5 %
PCA Accuracy @ alpha= 0.95 @ K= 3 : 85.0 %
```




```
PCA Accuracy @ alpha= 0.8 @ K= 5 : 82.0 %
PCA Accuracy @ alpha= 0.85 @ K= 5 : 83.5 %
PCA Accuracy @ alpha= 0.9 @ K= 5 : 81.5 %
PCA Accuracy @ alpha= 0.95 @ K= 5 : 81.0 %
```




```
PCA Accuracy @ alpha= 0.8 @ K= 7 : 78.0 %
PCA Accuracy @ alpha= 0.85 @ K= 7 : 77.0 %
PCA Accuracy @ alpha= 0.9 @ K= 7 : 75.0 %
PCA Accuracy @ alpha= 0.95 @ K= 7 : 73.0 %
```



7) faces vs non-faces:

- Download the Dataset and Understand the Format:

```
## Non Face dataset
nonFace_data = [[0 for k in range(10304)]for i in range(401)]
## array of 40 x 10 = 400 cell each = vector of 10304 element
for i in range(1,401):
  path = "/content/mix/"+str(i)+".jpg"
  image = cv2.imread(path,0)  # 0 to indicate greyscale image
  image = cv2.resize(image, (92,112))
  nonFace_data[i-1] = asarray(image).flatten()
  # cv2_imshow(image)
```

- Generate the Data Matrix and the Label vector

```
] D_non = np.array(np.zeros((400, 10304)))  ## initialization of the matrix 400 x 10304
  k = 0
  l = 0
  for i in range(400):
    k = 0
    for j in range(10304):
      D_non[l][k] =  np.asarray(nonFace_data[i][j]) ## take each cell image from numpydata to be a row in the matrix
      k = k + 1
    l = l +1
  D = np.array(np.zeros((400, 10304)))  ## initialization of the matrix 400 x 10304
  k = 0
  for i in range(40):
    for j in range(10):
      D[k] =  np.asmatrix(numpydata[i][j]) ## take each cell image from numpydata to be a row in the matrix
      k = k + 1
  total_data = np.concatenate((D,D_non))
  label = np.array(np.zeros((800, 1)))
  for i in range (400):
    label[i] = 1
  for i in range (400,800):
    label[i] = 2
```

- Faces vs Non-Face using PCA:

```
accuracy_pca = np.zeros((5,4))
fail_pca=np.empty([4, 400])
f = 0
for h in range (400,801,100):    ## h = 800
  # odd rows for training 200x10304
  # even  rows for testing 200x10304
  training=np.zeros((int(h/2),10304))
  testing=np.zeros((int(h/2),10304))
  ytesting=np.zeros((int(h/2),1))
  ytraining=np.zeros((int(h/2),1))
  j,k=0,0
  for i in range (h):
    if (i%2==0):
      testing[j]=total_data[i]
      ytesting[j]=label[i]
      j=j+1
    else:
      training[k]=total_data[i]
      ytraining[k]=label[i]
      k=k+1
  # computing the mean:
  meanVect = np.mean(training, axis=0)
  # computing centered data:
  zTrain = training - meanVect
  # computing covariance matrix:
  # covariance = zTranspose*z / n
  cov_matrix = np.dot(zTrain.T, zTrain)/10304
  # calculating eigen values and eigen vector:
  values, vector = np.linalg.eigh(cov_matrix)
  indx=values.argsort()[::-1]   #[::-1] to take all the list in reverse order
  values = values[indx]
  vector = vector[:,indx]
```

```
  # computing the fraction of total variance:
  total_eigenvalues = sum(values)
  r = np.array([0,0,0,0])
  alpha = np.array([0.8, 0.85, 0.9, 0.95])
  for x in range(4):
    fraction = 0
    for i in range(10304):
      fraction += values[i]
      if fraction/total_eigenvalues >= alpha[x]:
        r[x]=i+1
        break
  # # reduced basis:
  U1 = vector[:,:r[0]]
  U2 = vector[:,:r[1]]
  U3 = vector[:,:r[2]]
  U4 = vector[:,:r[3]]
  #b) projecting the training set:
  ProjTrain1 = np.dot(zTrain, U1)
  ProjTrain2 = np.dot(zTrain, U2)
  ProjTrain3 = np.dot(zTrain, U3)
  ProjTrain4 = np.dot(zTrain, U4)
  # projecting the test set:
  # computing the mean of test:
  meanVectTest = np.mean(testing, axis=0)
  # computing centered data:
  zTest = testing - meanVectTest
  ProjTest1 = np.dot(zTest, U1)
  ProjTest2 = np.dot(zTest, U2)
  ProjTest3 = np.dot(zTest, U3)
  ProjTest4 = np.dot(zTest, U4)
  knn_pca = KNeighborsClassifier(n_neighbors=1,weights='distance')
  knn_pca.fit(ProjTrain1, ytraining.ravel())
  pca_pred1 = knn_pca.predict(ProjTest1)
  accuracy_pca[f][0] = accuracy_score(ytesting, pca_pred1)
```

```
knn_pca.fit(ProjTrain2, ytraining.ravel())
pca_pred2 = knn_pca.predict(ProjTest2)
accuracy_pca[f][1] = accuracy_score(ytesting, pca_pred2)

knn_pca.fit(ProjTrain3, ytraining.ravel())
pca_pred3 = knn_pca.predict(ProjTest3)
accuracy_pca[f][2] = accuracy_score(ytesting, pca_pred3)

knn_pca.fit(ProjTrain4, ytraining.ravel())
pca_pred4 = knn_pca.predict(ProjTest4)
accuracy_pca[f][3]= accuracy_score(ytesting, pca_pred4)

if (h==800):
  for v in range (4):
    ytestre1=np.reshape(ytesting,(1,400))
    locals()["predre"+str(v+1)]=np.reshape(locals()["pca_pred"+str(v+1)],(1,400))
    fail_pca[v][:]=locals()["predre"+str(v+1)]==ytestre1
    cm = confusion_matrix(ytesting, locals()["pca_pred"+str(v+1)])
    ax= plt.subplot(2,2,v+1)
    sns.heatmap(cm,annot=True, fmt='g', ax=ax)
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['face', 'nonface']); ax.yaxis.set_ticklabels(['face', 'nonface']);
    plt.tight_layout()
f = f + 1
```

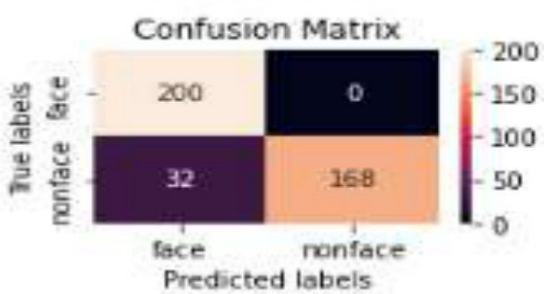
confusion matrix to represent number of failure and success cases

number of faces=400 , number of non-faces=400

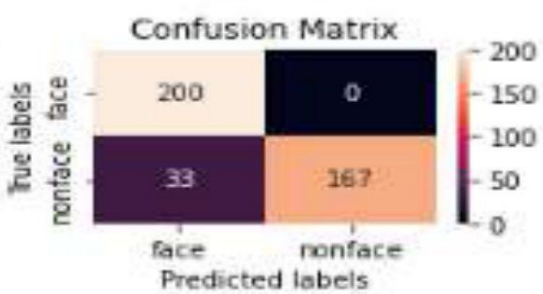
alpha=[0.8,0.85,0.9,0.95]

Alpha=0.8

Alpha=0.85



Alpha=0.9



Alpha=0.95

- Show failure images

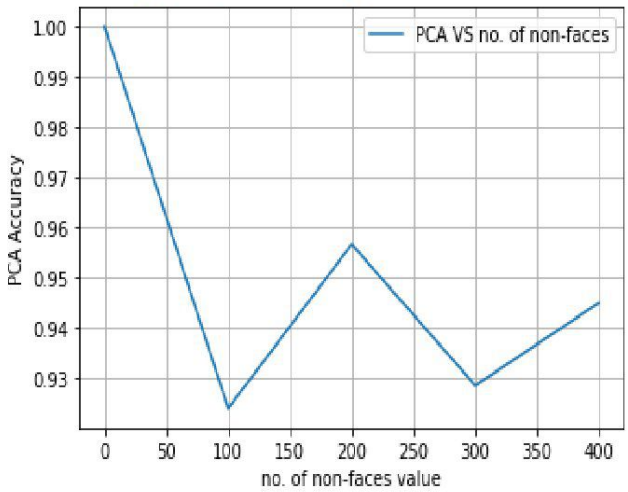
```
for k in range(4):
  print("for alpha=",alpha[k])
  for j in range (fail_pca.shape[1]):
    if (fail_pca[k][j]==False):
      xx=testing[j]
      xx=xx[0:10304]
      XX=np.reshape(xx,(112,92))
      img=Image.fromarray(XX)
      img.show()
```

- Plot the accuracy vs the number of non-faces images while fixing the number of face images.

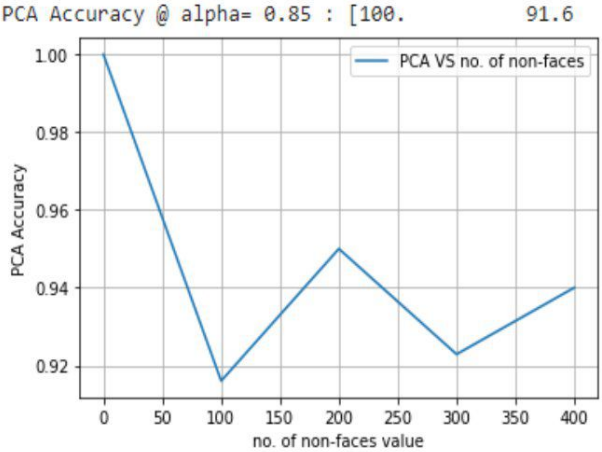
```
print("PCA accuracy VS Number of Non-faces data")
print(accuracy_pca)
print()
h = [0,100,200,300,400]
for i in range(4):  ## i alpha
  print("PCA Accuracy @ alpha=",alpha[i],":",accuracy_pca[:,i]*100,"%")
  plt.plot(h,accuracy_pca[:,i], label = 'PCA VS no. of non-faces')
  plt.legend()
  plt.xlabel('no. of non-faces value')
  plt.ylabel('PCA Accuracy')
  plt.grid()
  plt.show()
```

```
PCA accuracy VS Number of Non-faces data
[[1.         1.         1.         1.        ]
 [0.924      0.916      0.904      0.904     ]
 [0.95666667 0.95       0.92333333 0.89666667]
 [0.92857143 0.92285714 0.9        0.88      ]
 [0.945      0.94       0.92       0.9175    ]]
```

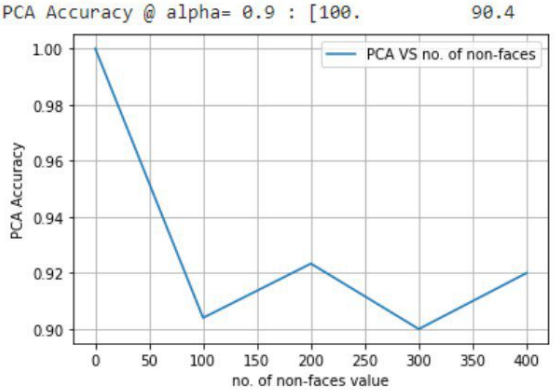
PCA Accuracy @ alpha= 0.8 : [100. 92.4 95.66666667 92.85714286 94.5] %



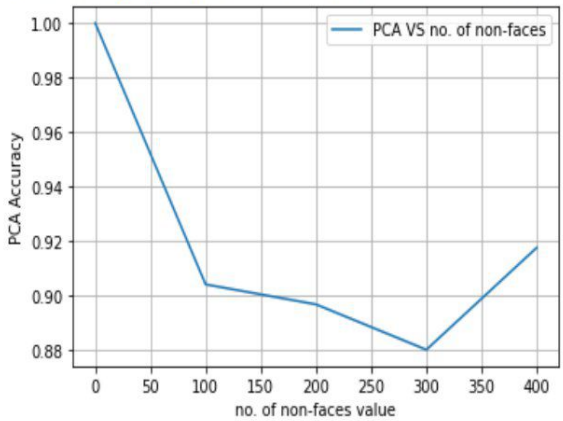
PCA Accuracy @ alpha= 0.85 : [100. 91.6 95. 92.28571429 94.] %



PCA Accuracy @ alpha= 0.9 : [100. 90.4 92.33333333 90. 92.] %



```
PCA Accuracy @ alpha= 0.95 : [100.          90.4         89.66666667  88.          91.75       ] %
```



- face vs non-face using LDA:

```
accuracy_lda = np.zeros((5,4))
w = 0
#class data
for h in range (400,801,100):     ### h = 800
  training=np.zeros((int(h/2),10304))
  testing=np.zeros((int(h/2),10304))
  ytesting=np.zeros((int(h/2),1))
  ytraining=np.zeros((int(h/2),1))
  j,k=0,0
  for i in range (h):
    if (i%2==0):
      testing[j]=total_data[i]
      ytesting[j]=label[i]
      j=j+1
    else:
      training[k]=total_data[i]
      ytraining[k]=label[i]
      k=k+1

  classlist=[]
  classlist.append(training[0:200,:])
  classlist.append(training[200:int(h/2),:])
  mean=[]
  mean.append(np.mean(training[0:200,:],axis=0))
mean.append(np.mean(training[200:int(h/2),:],axis=0))
bs = np.zeros((10304,10304))
mean[1][np.isnan(mean[1])] = 0
b1=mean[0]-mean[1]
b1.resize(10304,1)
b1trans=np.transpose(b1)
bs=b1@b1trans
sw = np.zeros((10304,10304))
for i in range (2):
  zz=classlist[i]-mean[i]
  zztrans=np.transpose(zz)
  sw+=zztrans@zz
#get sw inverse
sinv=np.linalg.inv(sw)
# get sinv*sb
x=sinv@bs
eigenvalues,eigenvectors=np.linalg.eigh(x)
idx=eigenvalues.argsort()[::-1]   #[::-1] to take all the list in reverse order
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]
projection=eigenvectors[:,0:1] # take first 39 eignvectors U 10304*39
overall_mean=np.mean(training,axis=0)
ztraining=training-overall_mean
projecttrain=ztraining@projection
```

```
ztesting=testing-overall_mean
projecttest=ztesting@projection
#first Nearest Neighbor to determine the class labels
kValues = [1,3,5,7]
fail_lda=np.empty([4, 400])
for v in range(4):
  knn_lda = KNeighborsClassifier(n_neighbors=kValues[v],weights='distance')
  knn_lda.fit(projecttrain, ytraining.ravel())
  lda_pred = knn_lda.predict(projecttest)
  accuracy_lda[w][v] = accuracy_score(ytesting, lda_pred)
  # providing actual and predicted values
  if (h==800):
    predre=np.reshape(lda_pred,(1,400))
    ytestre=np.reshape(ytesting,(1,400))
    fail_lda[v][:]=predre==ytestre
    cm = confusion_matrix(ytesting, lda_pred)
    ax= plt.subplot(2, 2,v+1)
    sns.heatmap(cm,annot=True, fmt='g', ax=ax)
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('Confusion Matrix ');
    ax.xaxis.set_ticklabels(['face', 'nonface']); ax.yaxis.set_ticklabels(['face', 'nonface']);
    plt.tight_layout()
w = w + 1
```

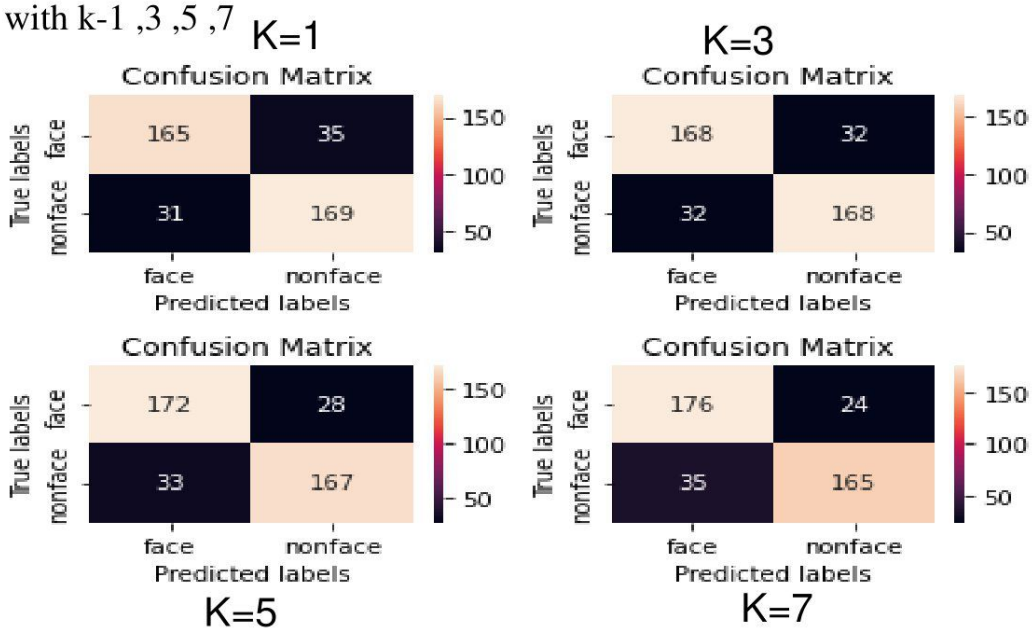
- How many dominant eigenvectors will you use for the LDA solution?

Number of classes -1 =2-1=1

Confusion matrix represents number of failure and success

number of faces=400 , number of non-faces=400

with k-1 ,3 ,5 ,7



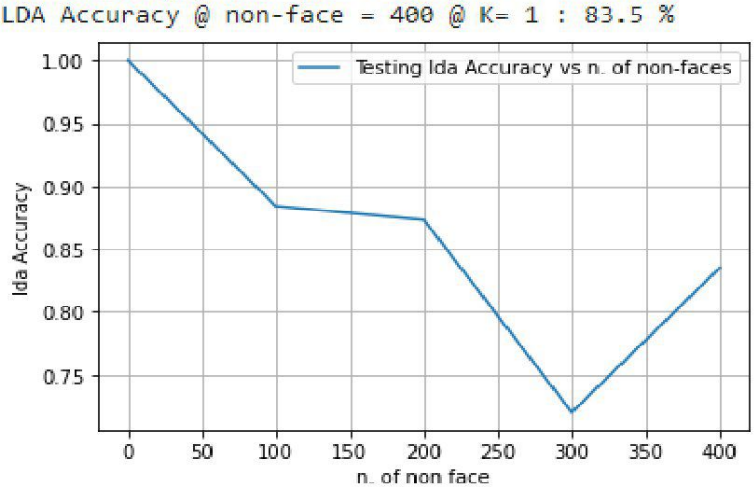
- show failure images.

```
K_nei=[1,3,5,7]
for k in range(4):
  print("for k=",K_nei[k])
  for j in range (fail_lda.shape[1]):
    if (fail_lda[k][j]==False):
      x=testing[j]
      x=x[0:10304]
      X=np.reshape(x,(112,92))
      img=Image.fromarray(X)
      img.show()
```

- plot the accuracy vs the number of non-faces images while fixing the number of face images.

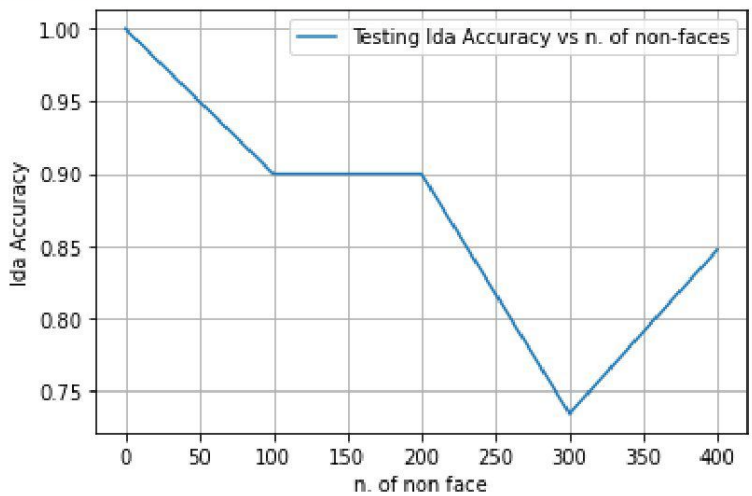
```
h = [0,100,200,300,400]
for i in range(4): # k
  for j in range(5): #non face
    print("LDA Accuracy @ non-face =",h[j],"@ K=",kValues[i],":", accuracy_lda[j][i]*100,"%")
  plt.plot(h,accuracy_lda[:,i], label = 'Testing lda Accuracy vs n. of non-faces')
  plt.legend()
  plt.xlabel('n. of non face')
  plt.ylabel('lda Accuracy')
  plt.grid()
  plt.show()
```

```
LDA Accuracy @ non-face = 0 @ K= 1 : 100.0 %
LDA Accuracy @ non-face = 100 @ K= 1 : 88.4 %
LDA Accuracy @ non-face = 200 @ K= 1 : 87.33333333333333 %
LDA Accuracy @ non-face = 300 @ K= 1 : 72.0 %
LDA Accuracy @ non-face = 400 @ K= 1 : 83.5 %
```

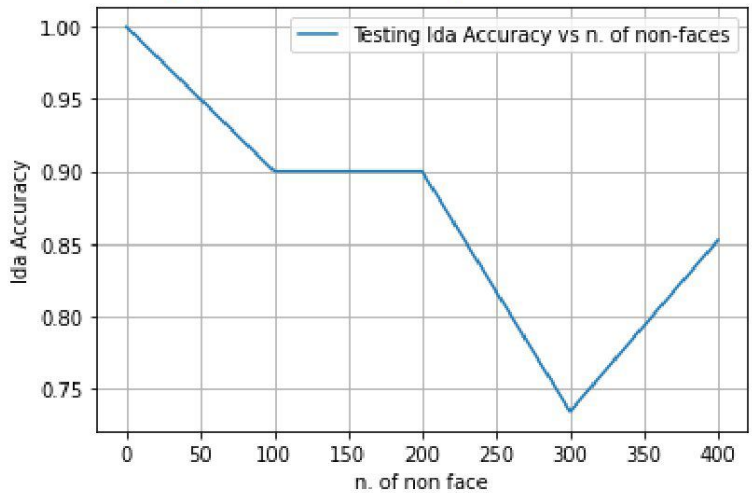


```
LDA Accuracy @ non-face = 0 @ K= 3 : 100.0 %
LDA Accuracy @ non-face = 100 @ K= 3 : 89.60000000000001 %
LDA Accuracy @ non-face = 200 @ K= 3 : 89.0 %
LDA Accuracy @ non-face = 300 @ K= 3 : 74.0 %
LDA Accuracy @ non-face = 400 @ K= 3 : 84.0 %
```


```
LDA Accuracy @ non-face = 0 @ K= 5 : 100.0 %
LDA Accuracy @ non-face = 100 @ K= 5 : 90.0 %
LDA Accuracy @ non-face = 200 @ K= 5 : 90.0 %
LDA Accuracy @ non-face = 300 @ K= 5 : 73.42857142857143 %
LDA Accuracy @ non-face = 400 @ K= 5 : 84.75 %
```



```
LDA Accuracy @ non-face = 0 @ K= 7 : 100.0 %
LDA Accuracy @ non-face = 100 @ K= 7 : 90.0 %
LDA Accuracy @ non-face = 200 @ K= 7 : 90.0 %
LDA Accuracy @ non-face = 300 @ K= 7 : 73.42857142857143 %
LDA Accuracy @ non-face = 400 @ K= 7 : 85.25 %
```



- Criticize the accuracy measure for large numbers of non-faces images in the training data:
  Accuracy starts to increase when number of non-faces is nearly equal to number of faces in training, the model will be able to classify them better as it trained on more of them

8. Bonus:

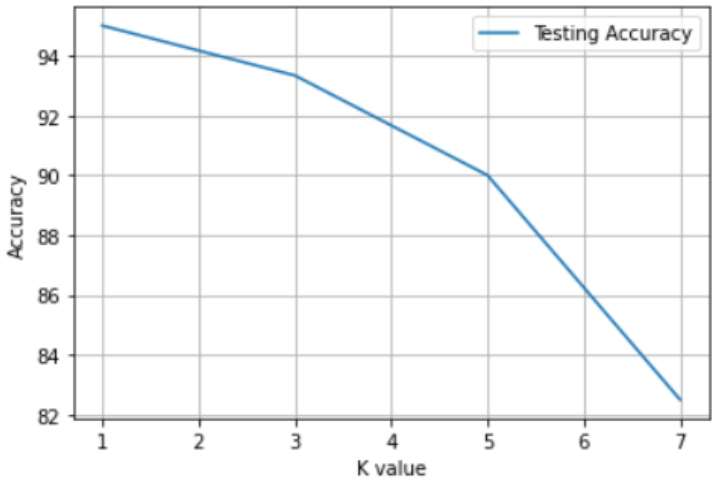
a. 70% Training and 30% Testing:

- **Splitting the dataset:**

```
# rows for training 280x10304
# rows for testing 120x10304
training=np.array(np.zeros((280,10304)))
testing=np.array(np.zeros((120,10304)))
ytesting=np.array(np.zeros((120,1)))
ytraining=np.array(np.zeros((280,1)))
j,k=0,0
for i in range (400):
  if (i%10<7):
    training[j]=D[i]
    ytraining[j]=y[i]
    j=j+1
  else:
    testing[k]=D[i]
    ytesting[k]=y[i]
    k=k+1
```

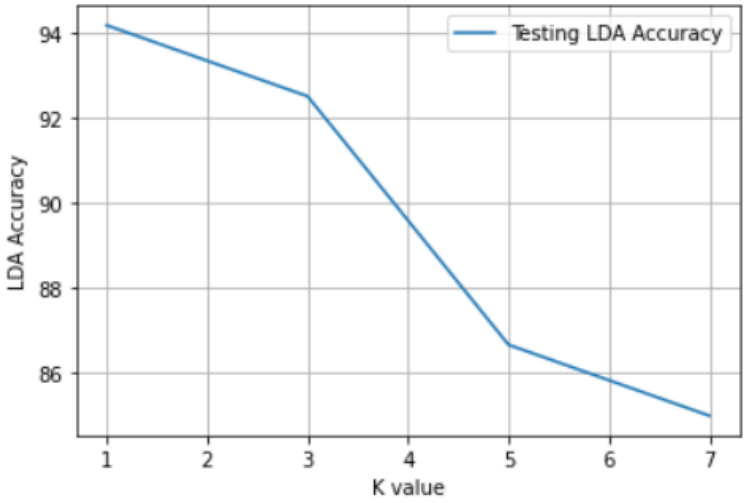
- **KNN accuracy:**

```
Accuracy @ K= 1 : 95.0 %
Accuracy @ K= 3 : 93.33333333333333 %
Accuracy @ K= 5 : 90.0 %
Accuracy @ K= 7 : 82.5 %
```



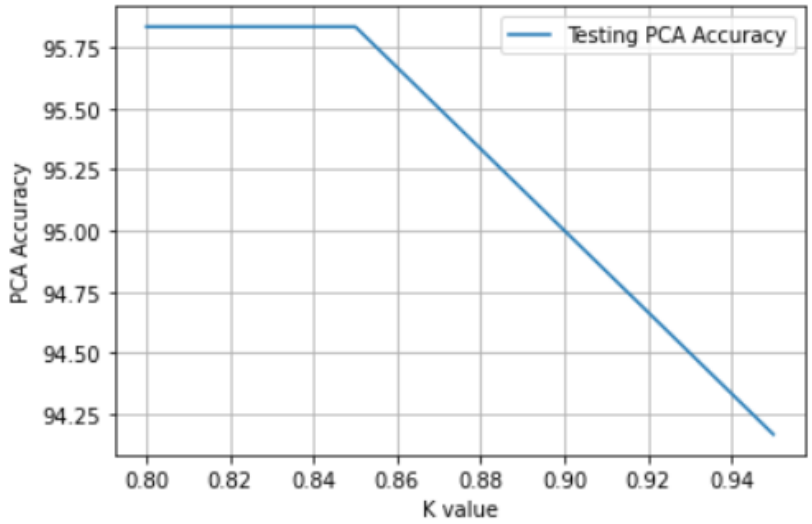
- **LDA accuracy:**

```
LDA Accuracy @ K= 1 : 94.16666666666667 %
LDA Accuracy @ K= 3 : 92.5 %
LDA Accuracy @ K= 5 : 86.66666666666667 %
LDA Accuracy @ K= 7 : 85.0 %
```

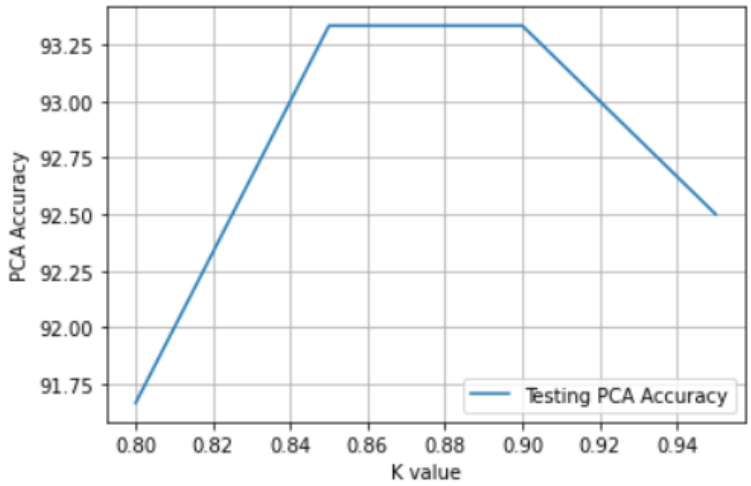


- **PCA accuracy:**

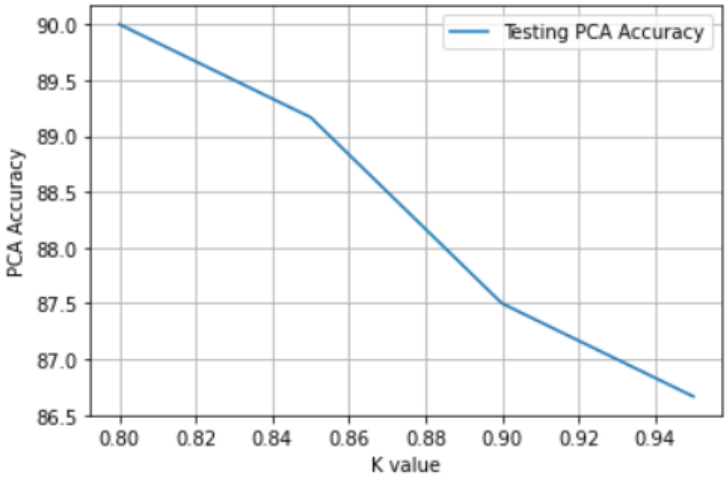
```
PCA Accuracy @ alpha= 0.8 @ K= 1 : 95.83333333333334 %
PCA Accuracy @ alpha= 0.85 @ K= 1 : 95.83333333333334 %
PCA Accuracy @ alpha= 0.9 @ K= 1 : 95.0 %
PCA Accuracy @ alpha= 0.95 @ K= 1 : 94.16666666666667 %
```



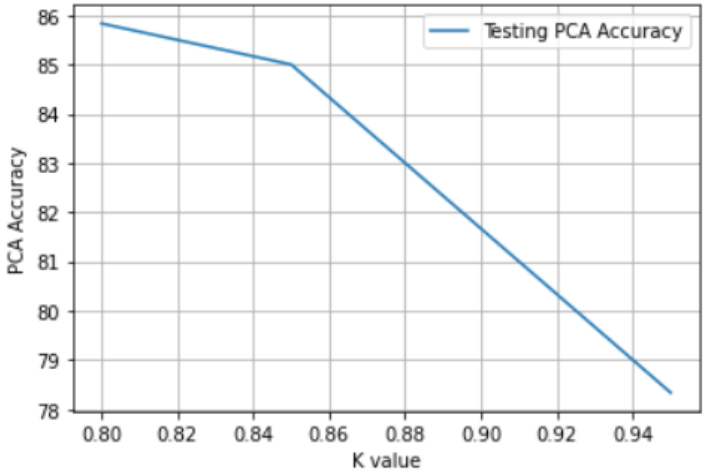
```
PCA Accuracy @ alpha= 0.8 @ K= 3 : 91.66666666666666 %
PCA Accuracy @ alpha= 0.85 @ K= 3 : 93.33333333333333 %
PCA Accuracy @ alpha= 0.9 @ K= 3 : 93.33333333333333 %
PCA Accuracy @ alpha= 0.95 @ K= 3 : 92.5 %
```




```
PCA Accuracy @ alpha= 0.8 @ K= 5 : 90.0 %
PCA Accuracy @ alpha= 0.85 @ K= 5 : 89.16666666666667 %
PCA Accuracy @ alpha= 0.9 @ K= 5 : 87.5 %
PCA Accuracy @ alpha= 0.95 @ K= 5 : 86.66666666666667 %
```




```
PCA Accuracy @ alpha= 0.8 @ K= 7 : 85.83333333333333 %
PCA Accuracy @ alpha= 0.85 @ K= 7 : 85.0 %
PCA Accuracy @ alpha= 0.9 @ K= 7 : 81.66666666666667 %
PCA Accuracy @ alpha= 0.95 @ K= 7 : 78.33333333333333 %
```



- **Observation:**

There is an improvement in the accuracy in 70% training and 30% testing compared with 50% training and 50% testing.

b. Variations on PCA algorithm

In this algorithm we used the SVD algorithm to calculate the PCA of the dataset.

- **Code:**

```
[4]  from numpy.linalg import svd
     import numpy.linalg as la
     # computing the mean:
     meanSVD = np.mean(training, axis=0)
     # computing centered data:
     SVDCentered = training - meanSVD
     U, S, Vt = la.svd(SVDCentered, full_matrices=False)
```

```
# computing the fraction of total variance:
total_sigma = sum(S)
r = np.array([0,0,0,0])
alpha = np.array([0.8, 0.85, 0.9, 0.95])
projSVDTrain = []
for x in range(4):
  fraction = 0
  for i in range(len(S)):
    fraction += S[i]
    if fraction/total_sigma >= alpha[x]:
      r[x]=i+1
      break
  projSVDTrain.append(S[0]*np.outer(U[:,0],Vt[0,:]))
  for i in range(1, r[x]):
    projSVDTrain[x] = projSVDTrain[x] + (S[i]*np.outer(U[:,i],Vt[i,:]))
```

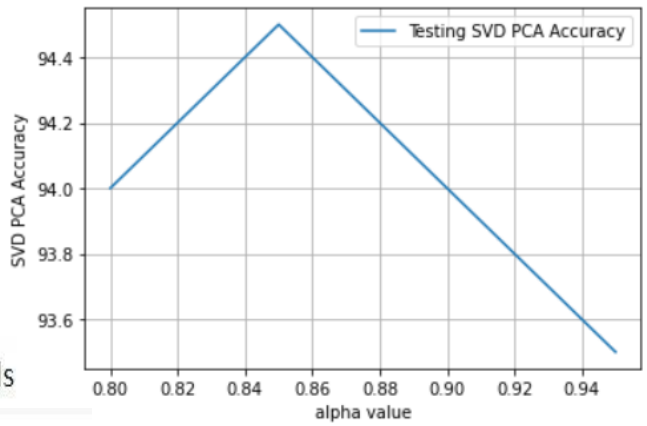
```
meannnSVD = np.mean(testing, axis=0)
# computing centered data:
SVDCenteredTest = testing - meannnSVD
uTest, sTest, vTest = la.svd(SVDCenteredTest, full_matrices=False)
total_sigmaTest = sum(sTest)
rTest = np.array([0,0,0,0])
# V = Vt.T
projSVDTest = []
for x in range(4):
  fraction = 0
  print(alpha[x])
  for i in range(len(sTest)):
    fraction += sTest[i]
    if fraction/total_sigmaTest >= alpha[x]:
      rTest[x]=i+1
      break
  projSVDTest.append(sTest[0]*np.outer(uTest[:,0],vTest[0,:]))
  for i in range(1, r[x]):
    projSVDTest[x] = projSVDTest[x] + (sTest[i]*np.outer(uTest[:,i],vTest[i,:]))
```

```
[7]  accuracySVD = [0,0,0,0]
     for i in range(4):
       knn_pca = KNeighborsClassifier(n_neighbors=1,weights='distance')
       knn_pca.fit(projSVDTrain[i], ytraining)
       pca_pred1 = knn_pca.predict(projSVDTest[i])
       accuracySVD[i] = accuracy_score(ytesting, pca_pred1)*100
     print(accuracySVD)
     plt.plot(alpha,accuracySVD, label = 'Testing SVD PCA Accuracy')
     plt.legend()
     plt.xlabel('alpha value')
     plt.ylabel('SVD PCA Accuracy')
     plt.grid()
     plt.show()
```

- **Output and observation:**

The accuracy is similar in both algorithms, but the SVD is much faster.

```
[94.0, 94.5, 94.0, 93.5]
```




```
The time taken by SVD:  2.58245587348938 seconds
```

```
The time taken by PCA:  305.0734724998474 seconds
```

Variations on LDA algorithm:

Using Quadratic Discriminant Analysis (QDA):

```
LDAstart = time.time()
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
qda = QuadraticDiscriminantAnalysis()
LDAend = time.time()
model = qda.fit(training, ytraining.ravel())
qda_pred=model.predict(testing)
accuracyqda = accuracy_score(ytesting, qda_pred)
print("accuarcy=",accuracyqda*100, "%")
print("The time taken by QDA: ", LDAend - LDAstart, "seconds")
```

```
accuarcy= 4.0 %
The time taken by QDA:  9.918212890625e-05 seconds
```

```
The time taken by LDA =  605.8511173725128 seconds
```

- **Observation:**
  QDA is similar to LDA based on the fact that there is an assumption of the observations being drawn from a normal distribution. The difference is that QDA assumes that each class has its own covariance matrix, while LDA does not.
  QDA is much faster but it is not accurate.