

Artificial Intelligence

Project (2) Report

Face Recognition

Team members

Nouran Hisham → 6532

Google colab notebook link:

<https://colab.research.google.com/drive/1MH3LY7G16g2F8KpWskjgnBPf5egw-ph?usp=sharing>

Note: Detailed comments are added in the code for further explanation.

Note: Detailed output is present in the google colab notebook.

Problem Statement:

We intend to perform face recognition. Face recognition means that for a given image you can tell the subject id. Our database of subject is very simple. It has 40 subjects.

Dataset used is <https://www.kaggle.com/kasikrit/att-database-of-faces>

Imported libraries to be used are:

Importing needed libraries

```
✓ [38] import numpy as np
      from PIL import Image
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.metrics import accuracy_score
      from matplotlib import pyplot as plt
```

1. Download the Dataset and Understand the Format:

In this step we downloaded the data using the API token Kaggle offers which loads a (Kaggle.json) file that is uploaded to google colab and connects me to Kaggle.

1. Download the Dataset and Understand the Format

```
✓ [39] ! pip install -q kaggle
      ! mkdir ~/.kaggle
      ! cp kaggle.json ~/.kaggle/
      ! chmod 600 ~/.kaggle/kaggle.json
      ! kaggle datasets list
      ! kaggle datasets download kasikrit/att-database-of-faces
      ! unzip /content/att-database-of-faces.zip
```

In this project we use 2 different datasets, the original dataset from Kaggle and a mix of this dataset with non-face images. Also, the train-test split stage is done through 2 different approaches.

First, the data is divided equally between the training and testing giving 5 samples for each. The other approach is giving 7 samples for training purposes and testing on only 3 samples.

The following lists are what we needed to reach these:

Declaring needed lists

```
✓ [40] #lists for original dataset
0s data = []
data_labels = []

#lists for 5-5 train-test
train_data = []
train_data_labels = []
test_data = []
test_data_labels = []

#lists for 7-3 train-test
train_data73 = []
train_data_labels73 = []
test_data73 = []
test_data_labels73 = []

#lists for face-nonface dataset
data1 = []
data_label1 = []
face_nonface_training = []
face_nonface_training_label = []
face_nonface_test = []
face_nonface_test_label = []
```

2.Generate the Data Matrix and the Label vector:

This was done using PIL image by opening the image and resizing it to the result of multiplying its dimensions ($92 \times 112 = 10304$) and then showing the images to make sure they're loaded the right way.

2. Generate the Data Matrix and the Label vector

```
✓ [4] def generate_datamatrix():
18s id = 1
for id in range(1,41):
    for image_count in range(1,11):
        image = Image.open('s'+str(id)+'/'+str(image_count)+'.pgm')
        image.show()
        display(image)
        image_array = np.array(image)
        image_converted = np.resize(image_array, (10304))
        data.append(image_converted)
        data_labels.append(id)

generate_datamatrix()
```



3.Split the Dataset into Training and Test sets:

From the Data Matrix D400x10304 keep the odd rows for training and the even rows for testing. This will give you 5 instances per person for training and 5 instances per person for testing. This was done by checking each id of each image whether it's odd or even and place into its right dataset accordingly.

3. Split the Dataset into Training and Test sets (5 train - 5 test)

```
✓ [5] def split_train_test():  
0s   for i in range(400):  
       if i%2 == 0:  
           test_data.append(data[i])  
           test_data_labels.append(data_labels[i])  
       else:  
           train_data.append(data[i])  
           train_data_labels.append(data_labels[i])  
  
split_train_test()
```

4.Classification using PCA:

The PCA algorithm given in the assignment pdf was followed thoroughly step by step to reach the projected train and test datasets which are sent to a simple classifier (KNN) for classifying each image.

4. Classification using PCA

```
✓ [7] def PCA(train, test, train_labels, test_labels, alpha):  
0s   #computing mean  
       mean_train = np.mean(train, axis=0)  
       mean_test = np.mean(test, axis=0)  
  
       #computing centered matrix  
       z_train = train - mean_train  
       z_test = test - mean_test  
  
       #computing covariance matrix  
       cov_matrix = np.cov(z_train, bias = 1, rowvar = False)  
  
       #computing eigen values and eigen vectors  
       eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)  
  
       #sorting eigen values and eigen vectors  
       idx = eigenvalues.argsort()[::-1]  
       eigenvalues_sorted = eigenvalues[idx]  
       eigenvectors_sorted = eigenvectors[:,idx]
```

```

#choosing dimensionality
eigenvalues_sum = eigenvalues.sum()
sum = 0
fractional_variance = 0
r = 0
while (sum/eigenvalues_sum) < alpha:
    sum = sum + eigenvalues_sorted[r]
    r = r + 1

#reduced basis
reduced_matrix = eigenvectors_sorted[:, 0 : r]

#reduced dimensionality data
u_train = np.dot(reduced_matrix.T, z_train.T)
u_test = np.dot(reduced_matrix.T, z_test.T)

#using a simple classifier
accuracy = KNN(u_train,train_labels,u_test,test_labels)

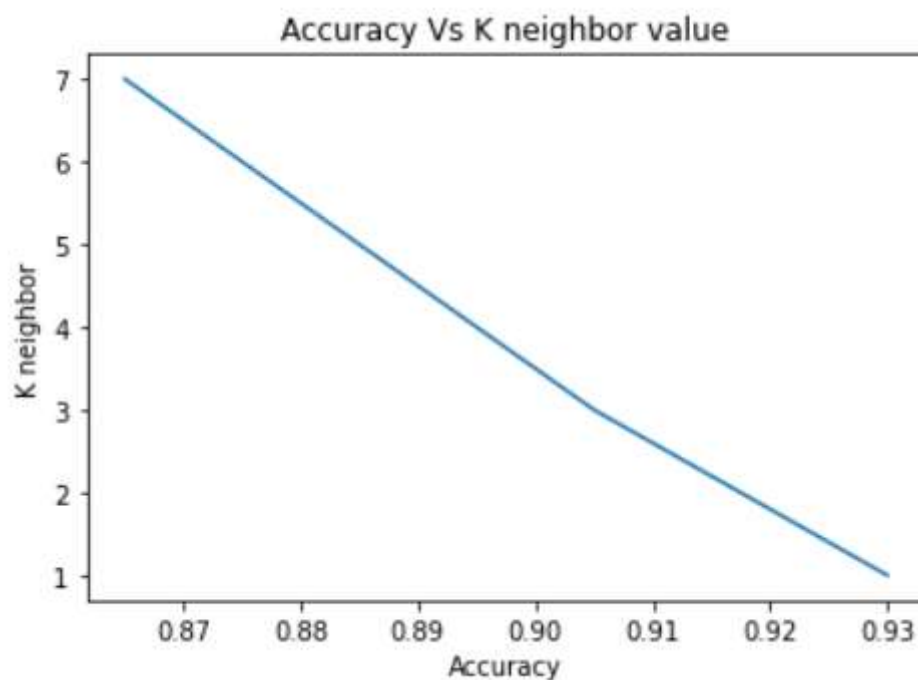
print("=====")
print("projected matrix shape"+str(u_train.shape))
print("=====")

return accuracy

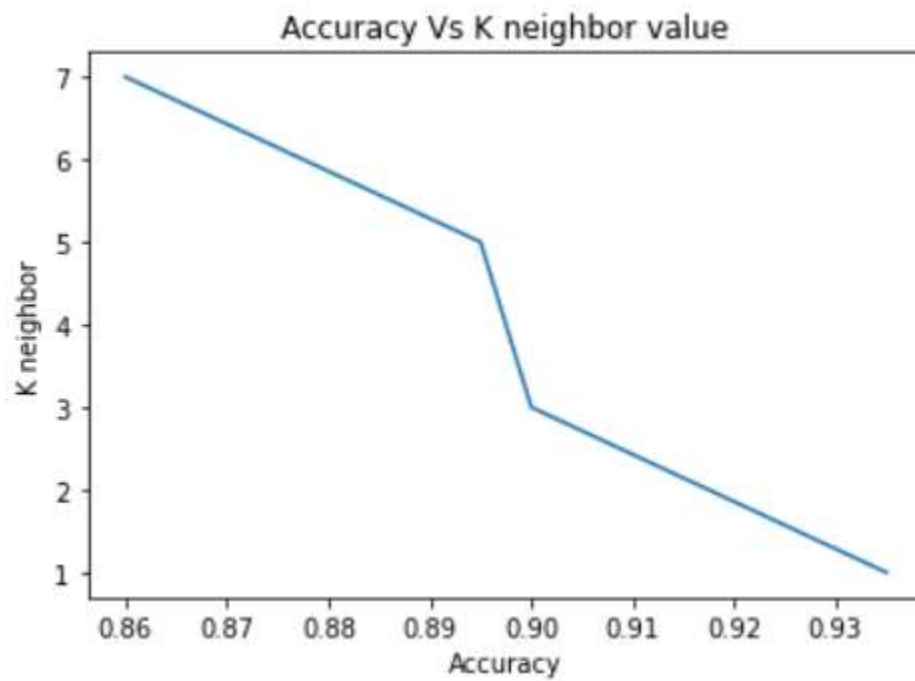
```

Note: It was noted that the more I increase the alpha, the accuracy decreases slightly.

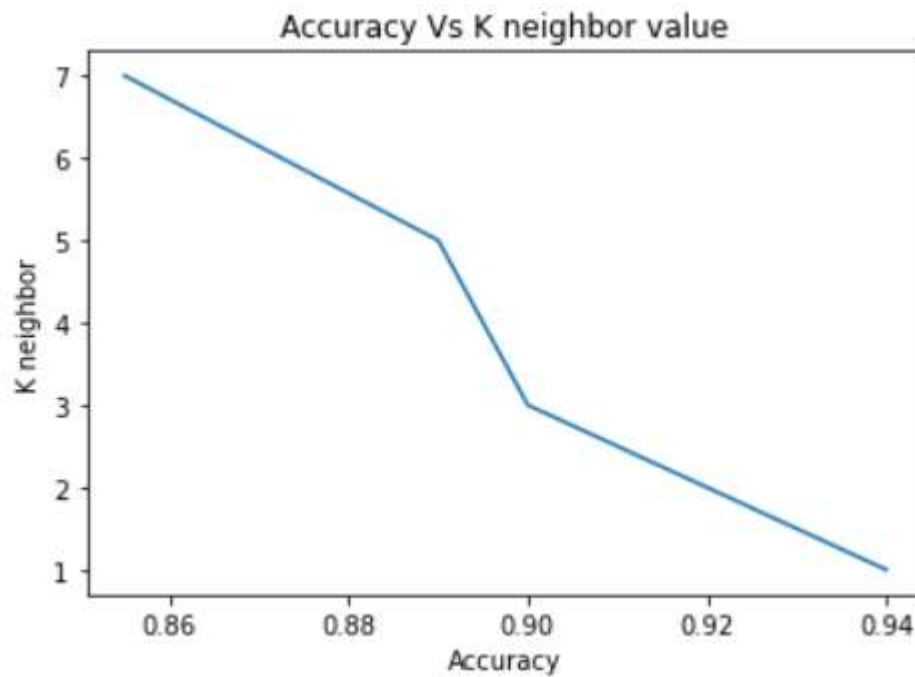
PCA results for alpha = 0.8 (different k's):



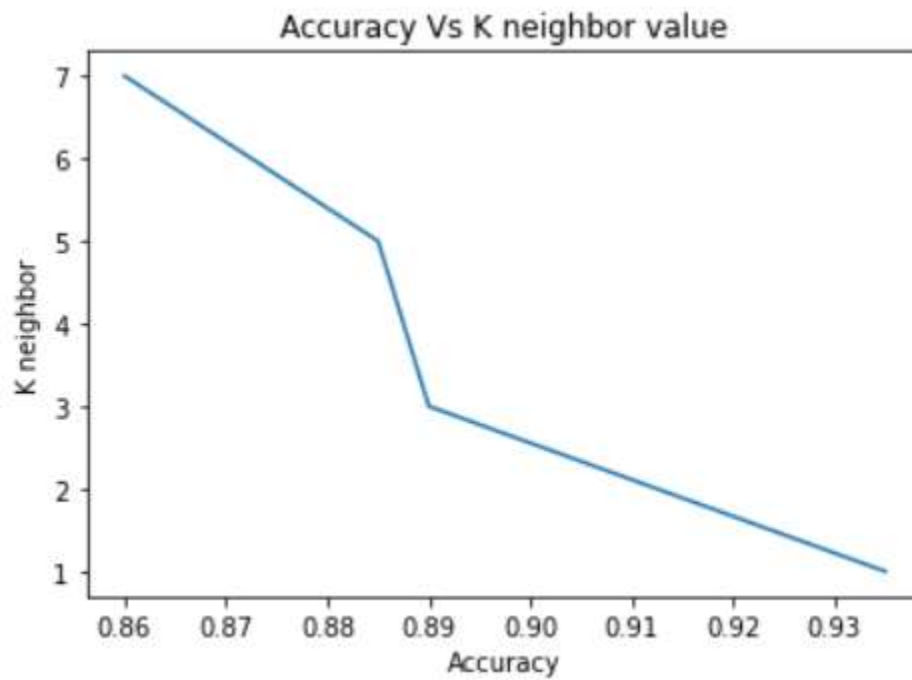
PCA results for $\alpha = 0.85$ (different k's):



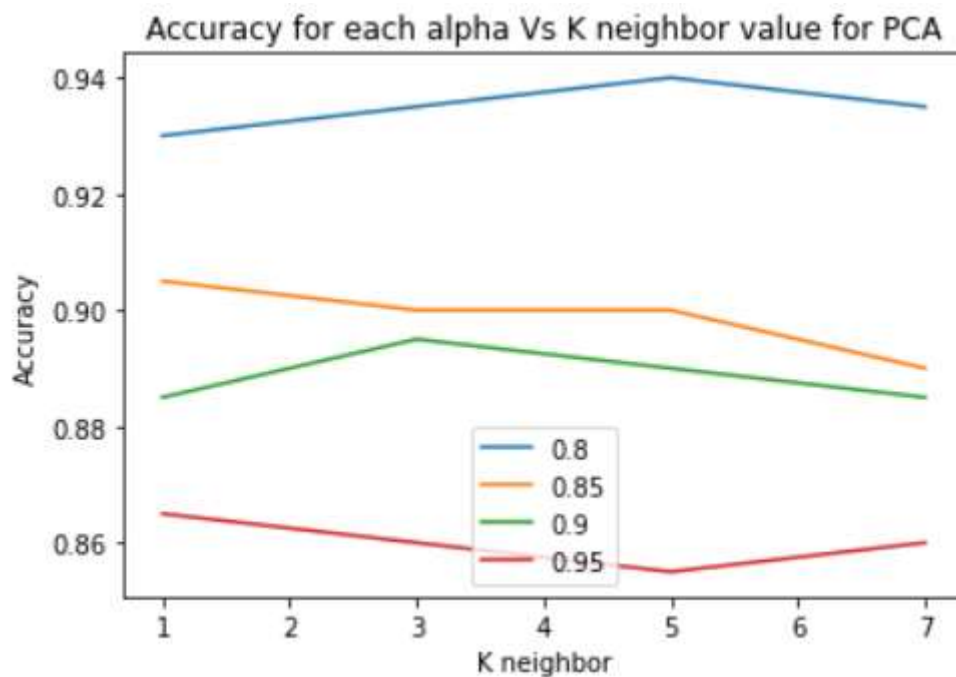
PCA results for $\alpha = 0.9$ (different k's):



PCA results for alpha = 0.95 (different k's):



Accuracy for each alpha (different k's):



5. Classification Using LDA:

The pseudocode given in the assignment pdf was followed thoroughly step by step to make sure we reach correct results.

The dataset was first split into its classes to be able to calculate the mean for each class and then we proceeded with the pseudocode until we reached the eigenvectors and made sure to use 39 dominant eigenvector like it is clearly mentioned in the assignment pdf.

5. Classification Using LDA

```
✓ [8] def LDA():  
  0s  
    #splitting the data into its classes  
    d_train2 = []  
    for i in range(40):  
        d_train2.append([])  
    j = -1  
    for i in range(200):  
        if (i % 5 == 0):  
            j = j + 1  
            d_train2[j].append(d_train[i])  
    d_train2 = np.asarray(d_train2, dtype="int32")  
  
    #initialising needed arrays  
    means = np.zeros((40, 10304))  
    between = np.zeros((10304, 10304))  
    within = np.zeros((10304, 10304))  
    center = np.zeros((40, 5, 10304))  
  
    #computing mean for each class  
    means = np.mean(d_train2, axis = 1)  
  
    #computing total mean  
    totalMean = np.mean(d_train2, axis=0)  
  
    #computing between-class scatter matrix  
    for i in range(40):  
        meanDif = means[i] - totalMean  
        between += (5*(np.dot(meanDif.T, meanDif)))
```



```

#computing centered matrix
for i in range(40):
    center[i] = d_train2[i] - means[i]

#computing within-class scatter matrix
for i in range(40):
    within += np.dot(center[i].T, center[i])

#preparing matrices for eigenvalues and eigenvectors
within = np.asarray(within)
inverse = np.linalg.inv(within)
s_inv_b = np.dot(inverse,between)

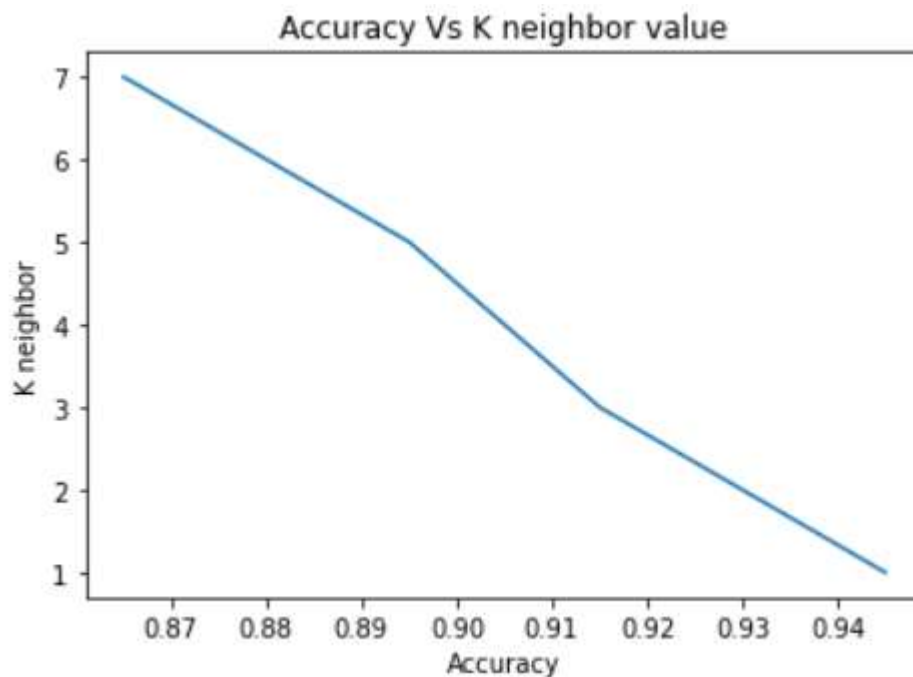
#computing, sorting and using 39 eigenvectors
eigenvalues_LDA, eigenvectors_LDA = np.linalg.eig(s_inv_b)
idx = eigenvalues_LDA.argsort()
eigenvalues_LDA.sort()
eigenvectors_LDA2 = eigenvectors_LDA[:, idx]
eigenvectors_LDA2 = eigenvectors_LDA2[:, len(eigenvalues_LDA)-39:]
eigenvectors_LDA_real=np.real(eigenvectors_LDA2)

#projecting train and test datasets
U_Train_LDA = np.dot(d_train , eigenvectors_LDA_real)
U_Test_LDA = np.dot(d_test , eigenvectors_LDA_real)

#using a simple classifier
KNN(U_Train_LDA.real.T,train_data_labels,U_Test_LDA.real.T,test_data_labels)

```

LDA results with different k's:



6. Classifier Tuning:

For the simple classifier, KNN was used with the aid of its built-in function in sklearn which we imported at the beginning of the program.

It's required to try the KNN classifier for PCA and LDA each four times with 4 different K's (the hyperparameter in KNN), k = 1,3,5,7, which was achieved using a simple for loop.

Anytime an image is misclassified, (!! Classification error !!) is printed out.

6. Classifier Tuning

```
✓ [9] def KNN(traindata, trainlabels, testdata, testlabels):  
    0s  
    n = [1,3,5,7]  
    accuracy_list = []  
    for i,nn in zip(range(len(n)),n):  
        print("-----")  
        print("For n = "+str(nn)+" :")  
        print("-----")  
        classifier = KNeighborsClassifier(n_neighbors = nn, weights = 'distance')  
        classifier.fit(traindata.T, trainlabels)  
        prediction = classifier.predict(testdata.T)  
        accuracy_list.append(accuracy_score(prediction,testlabels))  
        print("Accuracy is: " + str(accuracy_list[i]))  
        print("-----")  
        error_count = 0  
        for j in range(len(prediction)):  
            print("Picture number (" +str(j+1)+") is classified as (" +str(prediction[j])+") and is actually (" +str(testlabels[j])+").")  
            if((prediction[j]) != (testlabels[j])):  
                print("!! Classification error !!")  
        plt.plot(accuracy_list,n)  
        plt.title("Accuracy Vs K neighbor value")  
        plt.ylabel("K neighbor")  
        plt.xlabel("Accuracy")  
        plt.show()  
    return accuracy_list
```

Note: It shows that the accuracy when testing the same dataset with the same conditions (k - neighbor) that the LDA achieves better accuracies than PCA which was expected because LDA works knowing the data labels which is an advantage PCA doesn't have.

Note: Given the conditions of this dataset, that the training data is only 5 per class, 5-neighbor and 7-neighbor didn't do well regarding the accuracy of classification in comparison to 1-neighbor and 3-neighbor.

7.Compare vs Non-Face Images:

Non-face images were loaded from <https://www.kaggle.com/prasunroy/natural-images> and only 400 images of the cars were used as non-face images.

These car images were changed to grayscale to be similar to the original face images and prevent any bias results in the classification

A function is made to deal with the face-nonface problem to be able to change the number of non-face images and see its effect on the accuracy easily

Reading the nonface images and concatenating them with the face images

```
✓ [13] def face_nonface_data(size):  
0s     #reading face images  
     id = 1  
     for id in range(1,41):  
         for image_count in range (1,11):  
             image = Image.open('s'+str(id)+'/'+str(image_count)+'.pgm')  
             image_array = np.array(image)  
             image_converted = np.resize(image_array, (10304))  
             data1.append(image_converted)  
             data_label1.append("face")  
  
     #reading non-face images  
     for i in range(size):  
         if i <10:  
             image = Image.open('car_grayscale/car_000'+str(i)+'.jpg')  
             image_resized = image.resize((92,112))  
             image_array = np.array(image_resized)  
             image_converted = np.resize(image_resized,(10304))  
             data1.append(image_converted)  
             data_label1.append("non-face")  
         elif i<100:  
             image = Image.open('car_grayscale/car_00'+str(i)+'.jpg')  
             image_resized = image.resize((92,112))  
             image_array = np.array(image_resized)  
             image_converted = np.resize(image_resized,(10304))  
             data1.append(image_converted)  
             data_label1.append("non-face")
```

```

✓ [13] data_label1.append("non-face")
0s     else:
        image = Image.open('car_grayscale/car_0'+str(i)+'.jpg')
        image_resized = image.resize((92,112))
        image_array = np.array(image_resized)
        image_converted = np.resize(image_array,(10304))
        data1.append(image_converted)
        data_label1.append("non-face")

#splitting the mixed (face_nonface) dataset
for i in range(len(data1)):
    if i%2 == 0:
        face_nonface_test.append(data1[i])
        face_nonface_test_label.append(data_label1[i])
    else:
        face_nonface_training.append(data1[i])
        face_nonface_training_label.append(data_label1[i])

```

A new LDA function was made to deal with this new dataset:

Note: 39 dominant eigne vectors were used here as well.

LDA method for the face-nonface dataset

```

✓ [17] def LDA_2(train, test, train_labels, test_labels):
0s

    #counting the samples in the non-face class
    nonface_count = 0
    for i in range(len(train)):
        if train_labels[i] == "non-face":
            nonface_count += 1

    #splitting the 2 classes, face and non-face
    class1 = train[0:199]
    class2 = train[200:200+nonface_count]

    #computing mean for each class
    means = np.zeros((2,10304))
    means[0,:] = np.mean(class1,axis=0)
    means[1,:] = np.mean(class2,axis=0)

    #computing total mean
    totalMean = np.mean(means,axis=0).T

    #computing between-class scatter matrix
    between = np.zeros((means.shape[1],means.shape[1]))
    between = np.add(between,40* ((means[0] - totalMean) * (means[0] - totalMean).T))
    between = np.add(between,nonface_count* ((means[1] - totalMean) * (means[1] - totalMean).T))

```



```

#computing within-class scatter matrix
within = np.zeros((10304,10304))
center1 = class1 - means[0]
c = np.dot(center1.T,center1)
within = within + c
center2 = class2 - means[1]
c = np.dot(center2.T,center2)
within = within + c

#preparing the matrices for computing eigenvalues and eigen vectors
within = np.asarray(within)
inverse = np.linalg.inv(within)
s_inv_b = np.dot(inverse,between)

#computing, sorting and using 39 eigenvectors
eigenvalues_LDA, eigenvectors_LDA = np.linalg.eig(s_inv_b)
idx = eigenvalues_LDA.argsort()
eigenvalues_LDA.sort()
eigenvectors_LDA2 = eigenvectors_LDA[:, idx]
eigenvectors_LDA2 = eigenvectors_LDA2[:, len(eigenvalues_LDA)-39:]
eigenvectors_LDA_real=np.real(eigenvectors_LDA2)

#projecting train and test data
U_Train_LDA = np.dot(train , eigenvectors_LDA_real)
U_Test_LDA = np.dot(test , eigenvectors_LDA_real)

#using a simple classifier
accuracy = KNN(U_Train_LDA.real.T,train_labels,U_Test_LDA.real.T,test_labels)
return accuracy

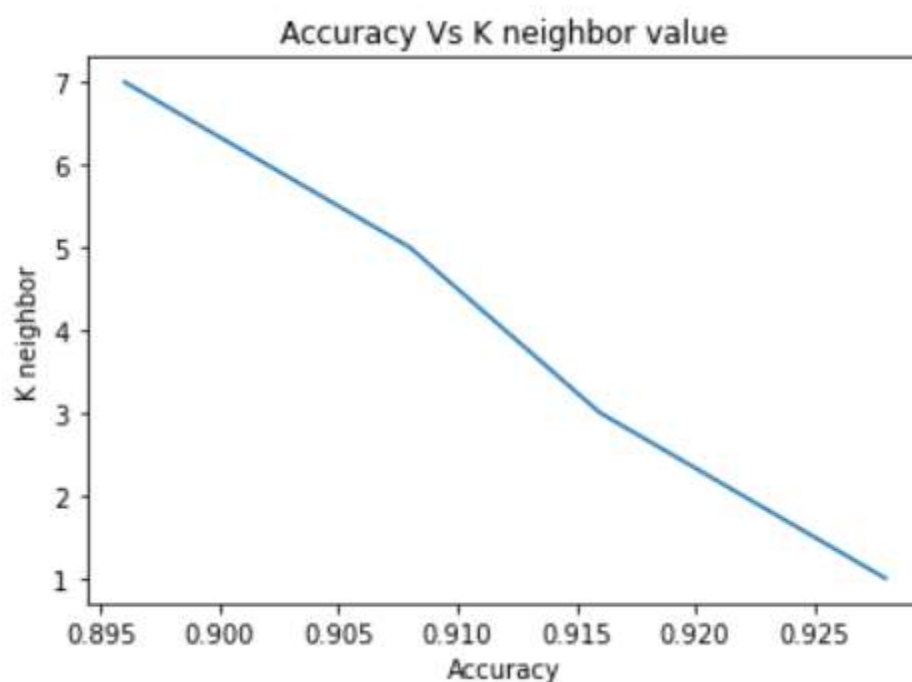
```

This new dataset is tested with PCA and LDA 3 different times to see the effect of increasing the number of non-face images in the dataset on the accuracy.

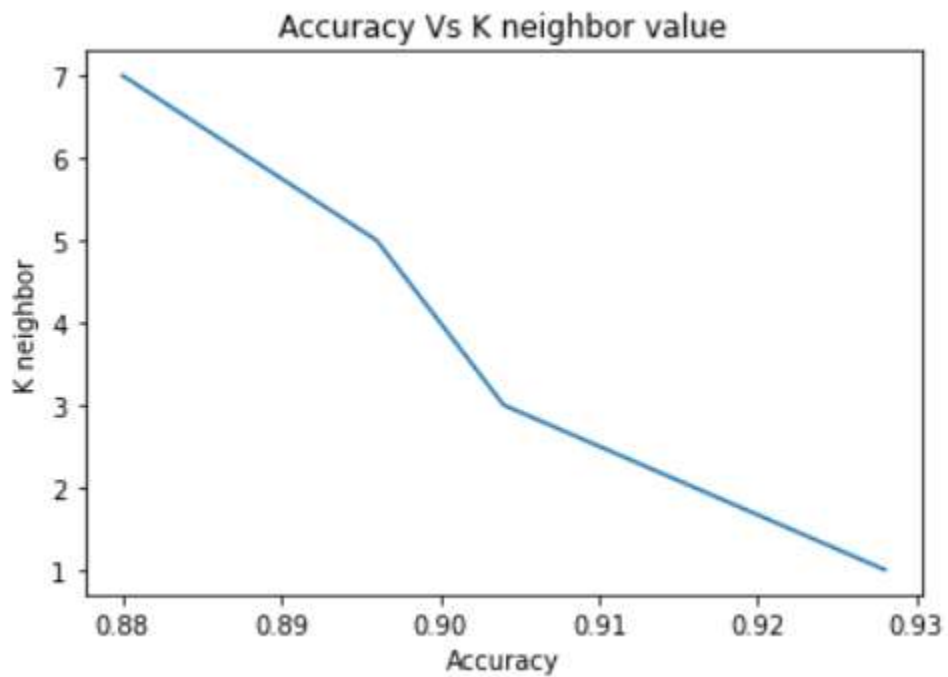
First, we started with 100, then doubled it to 200 then finally doubled it to 400 non-face image in the data set.

Tests when non-face images are 100:

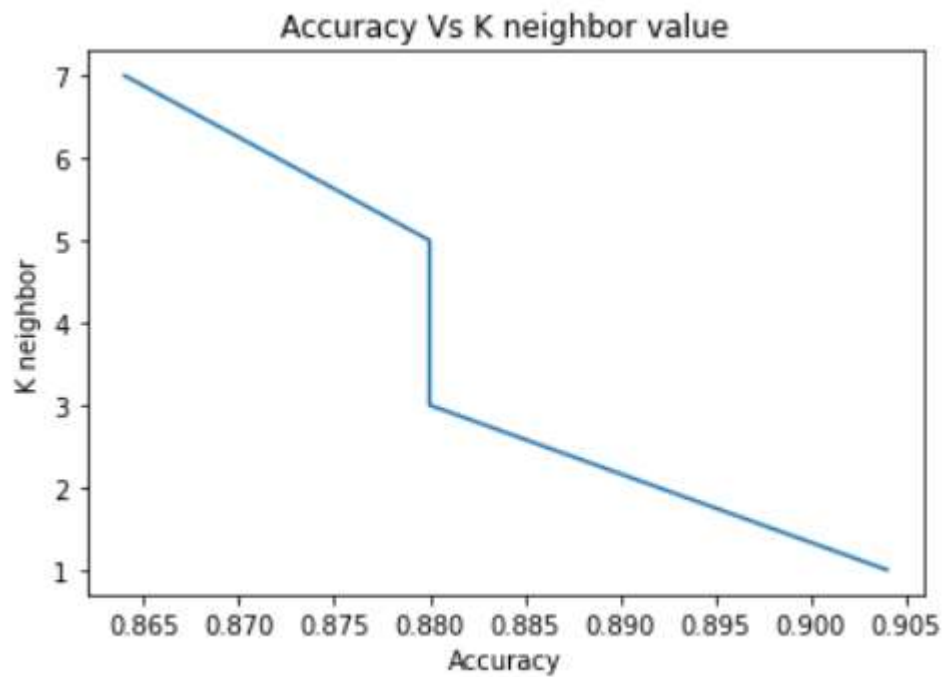
PCA result for alpha = 0.8 (different k's):



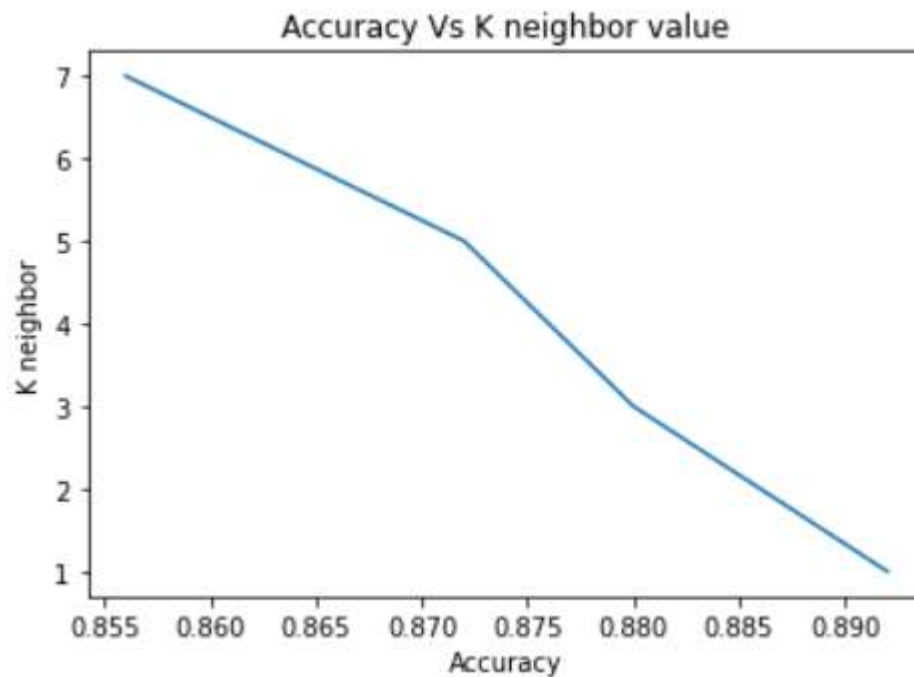
PCA result for $\alpha = 0.85$ (different k's):



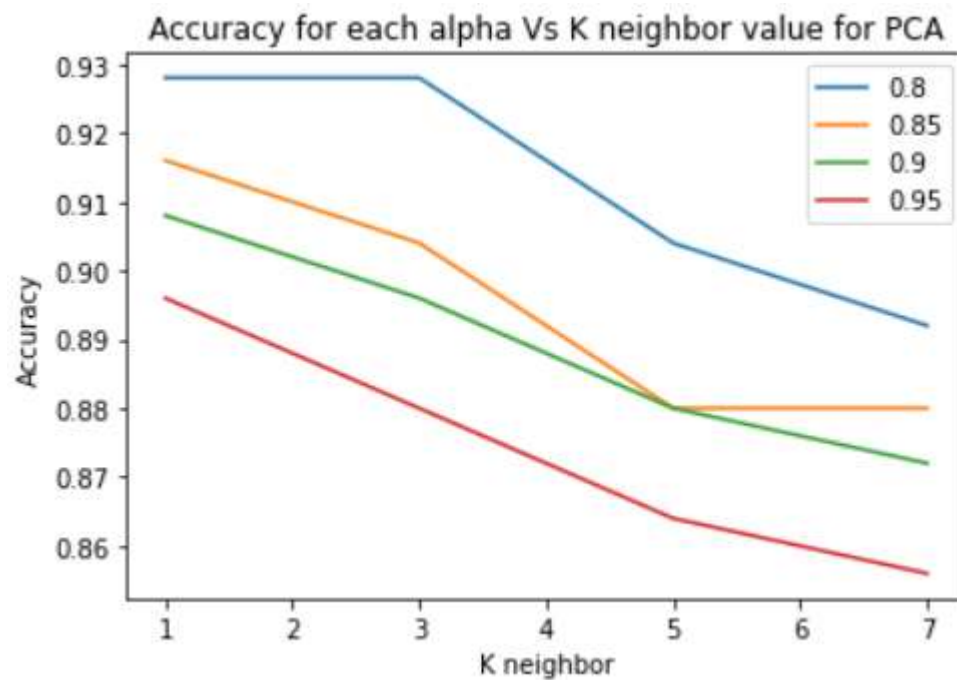
PCA result for $\alpha = 0.9$ (different k's):



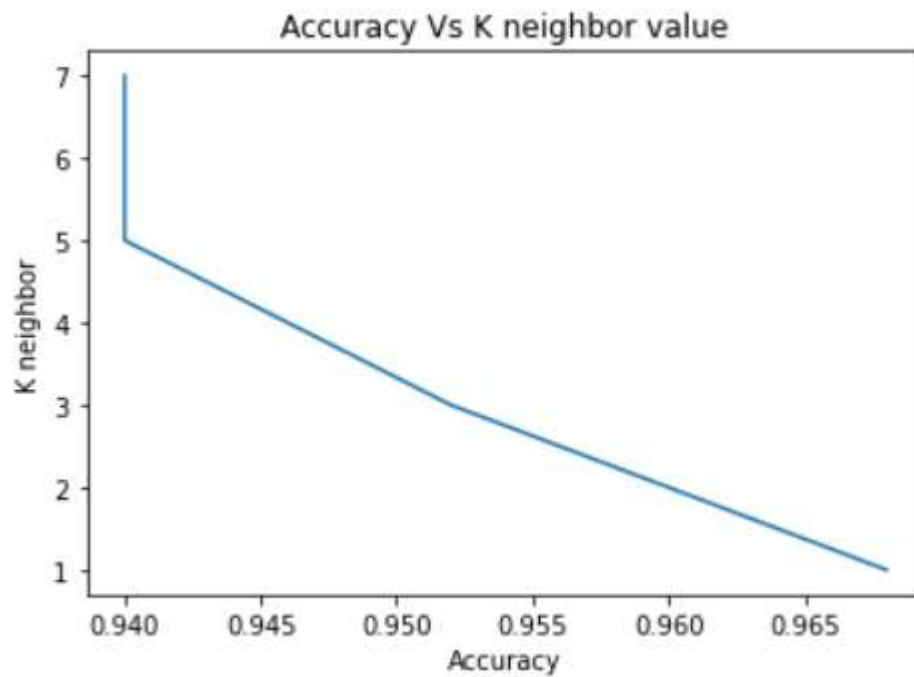
PCA result for alpha = 0.95 (different k's):



Accuracy for each alpha (different k's):

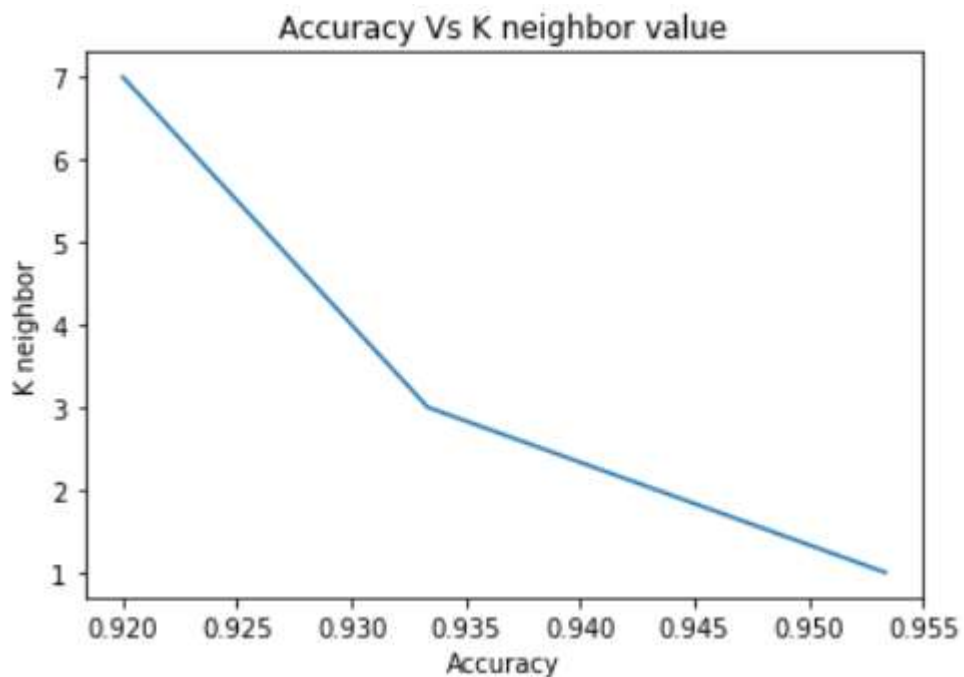


LDA results with different k's:

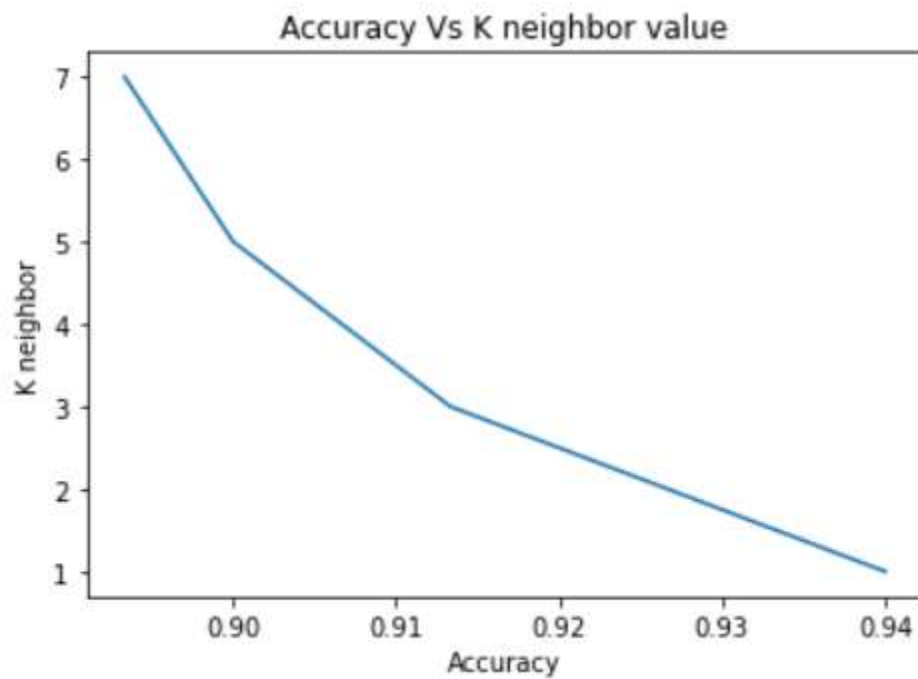


Tests when non-face images are 200:

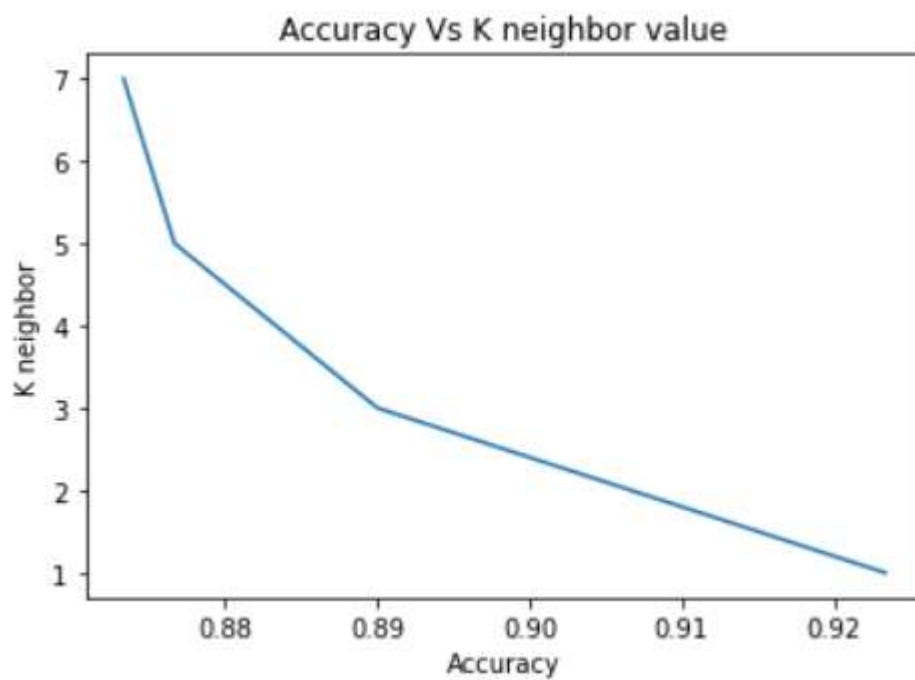
PCA result for alpha = 0.8 (different k's):



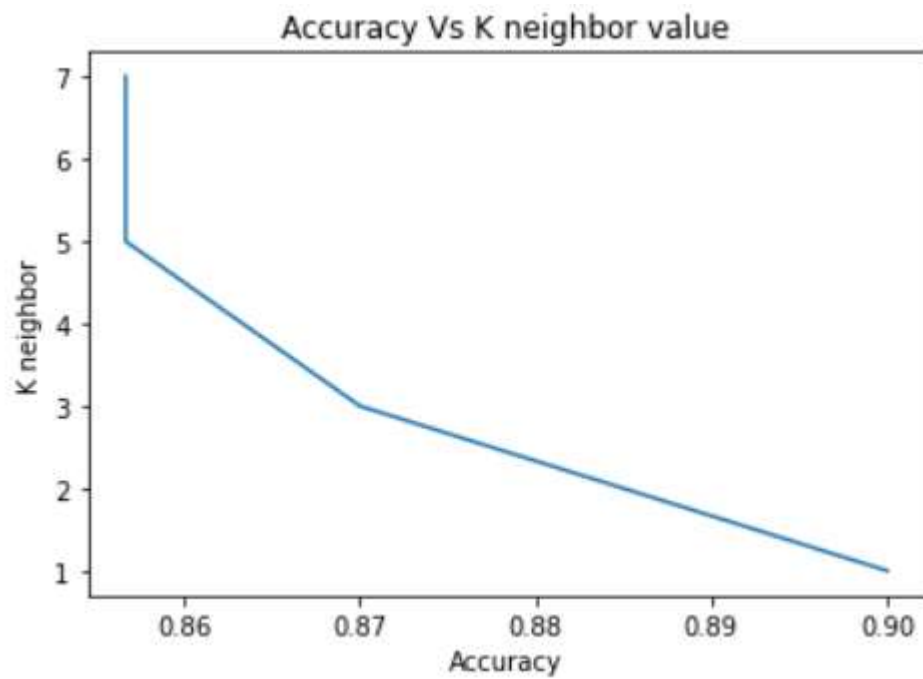
PCA result for alpha = 0.85 (different k's):



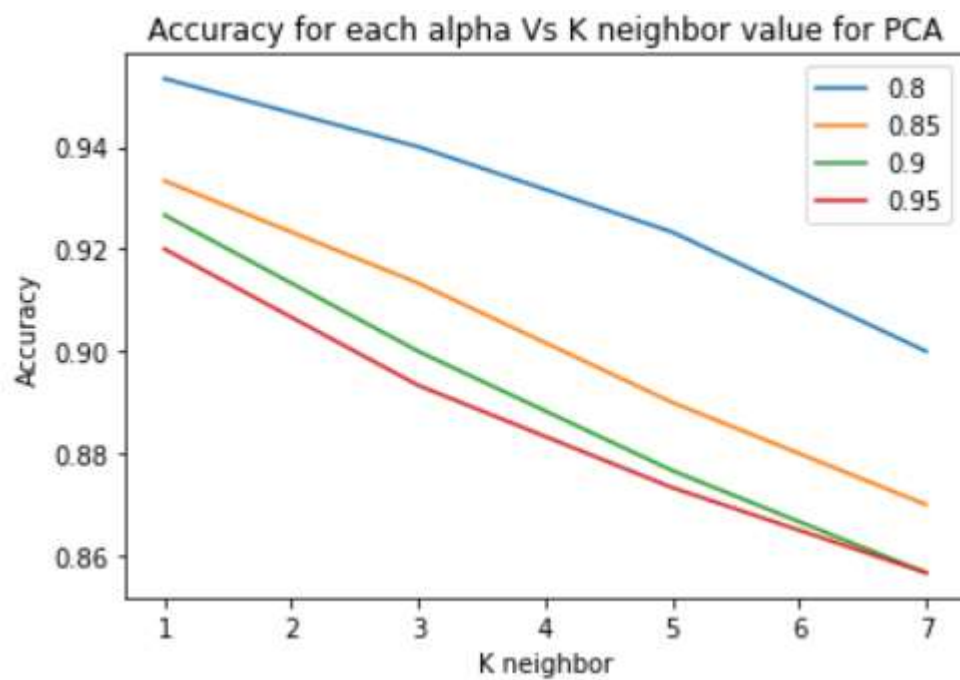
PCA result for alpha = 0.9 (different k's):



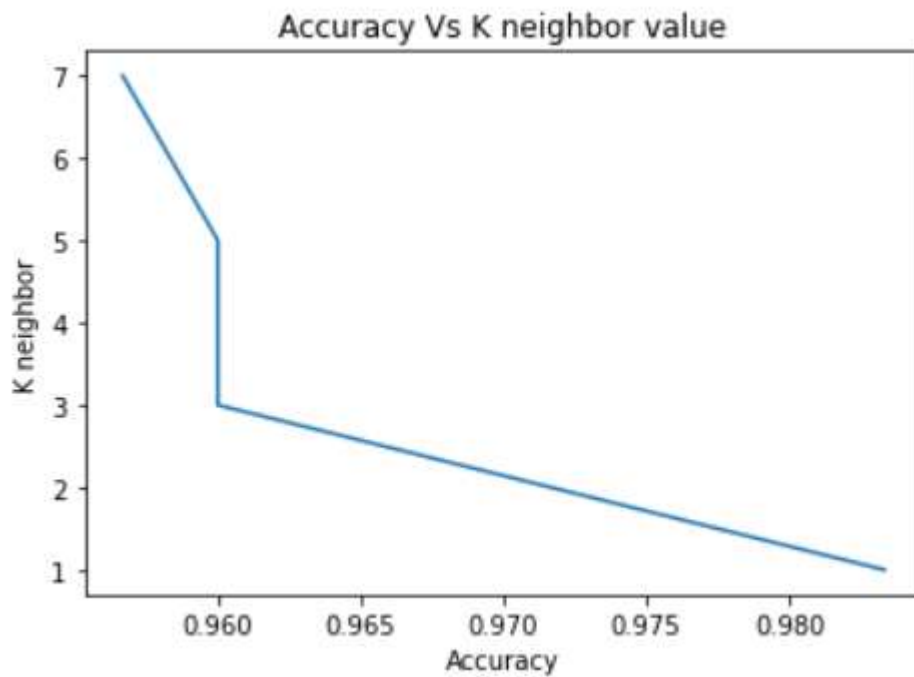
PCA result for alpha = 0.95 (different k's):



Accuracy for each alpha (different k's):

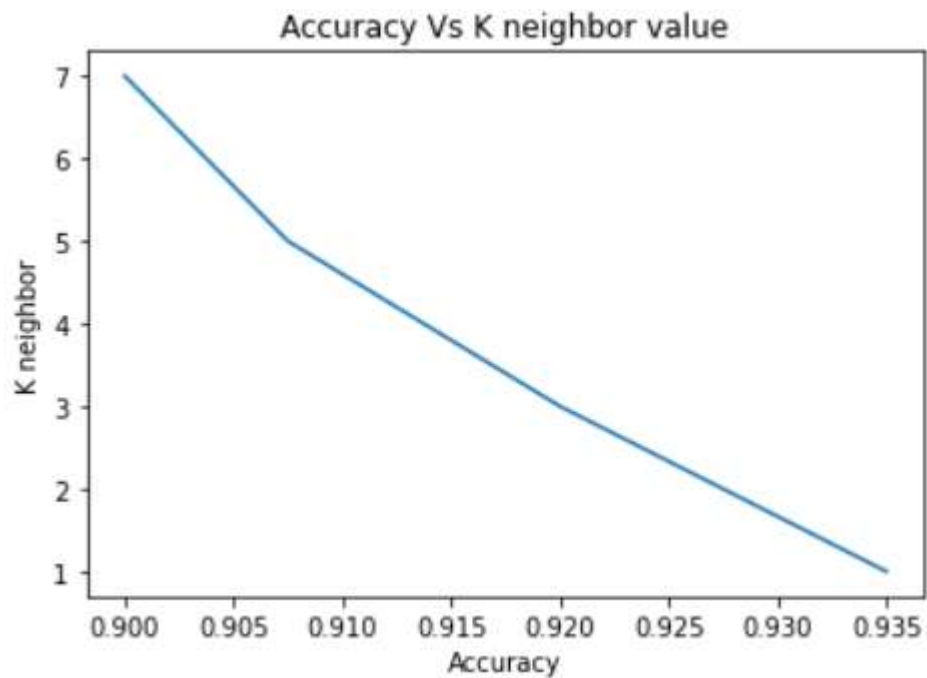


LDA results with different k's:

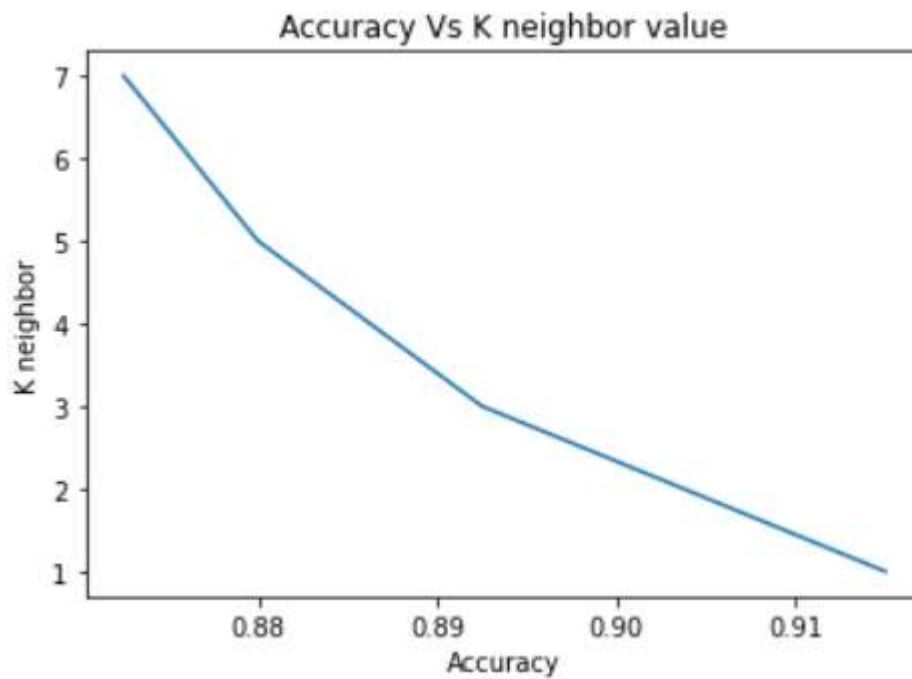


Tests when non-face images are 400:

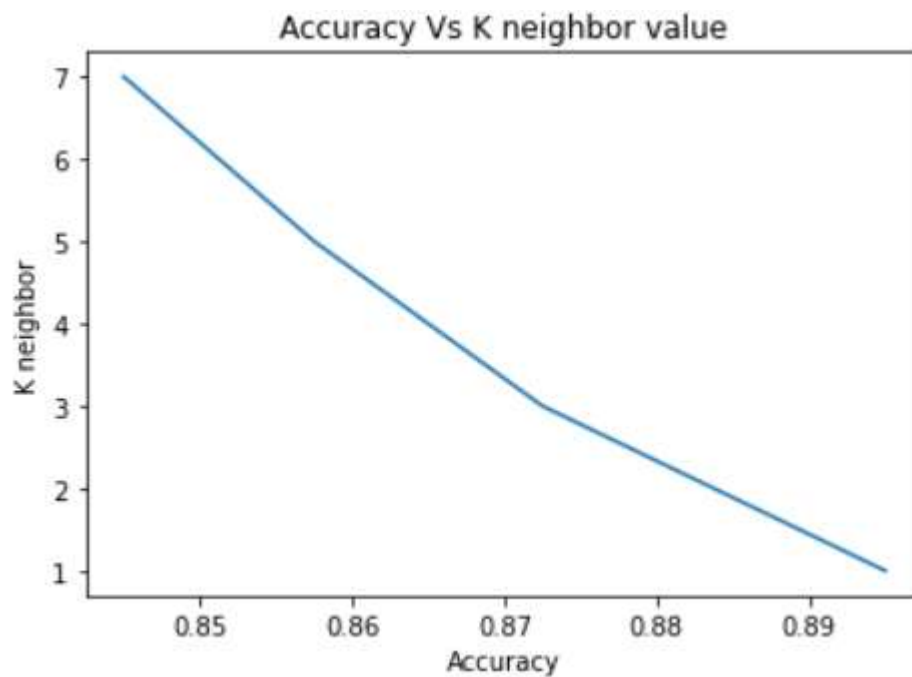
PCA result for alpha = 0.8 (different k's):



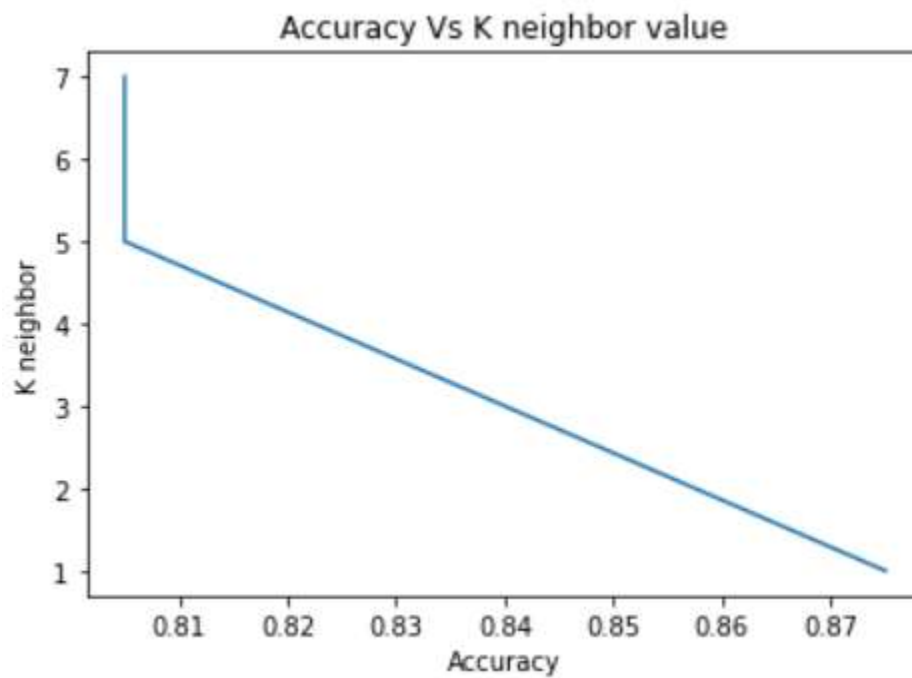
PCA result for $\alpha = 0.85$ (different k's):



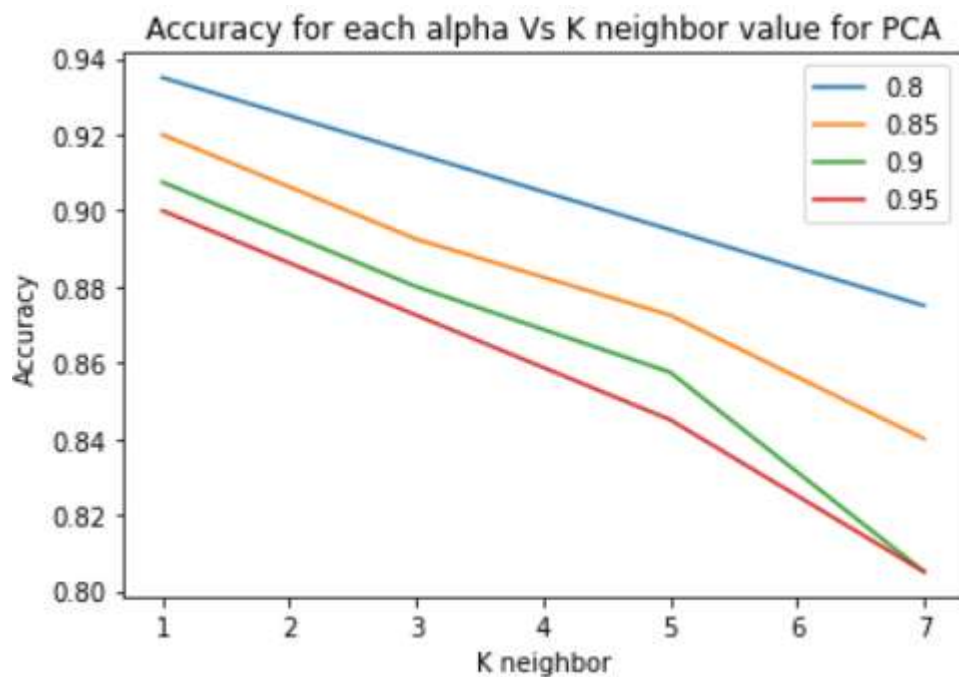
PCA result for $\alpha = 0.9$ (different k's):



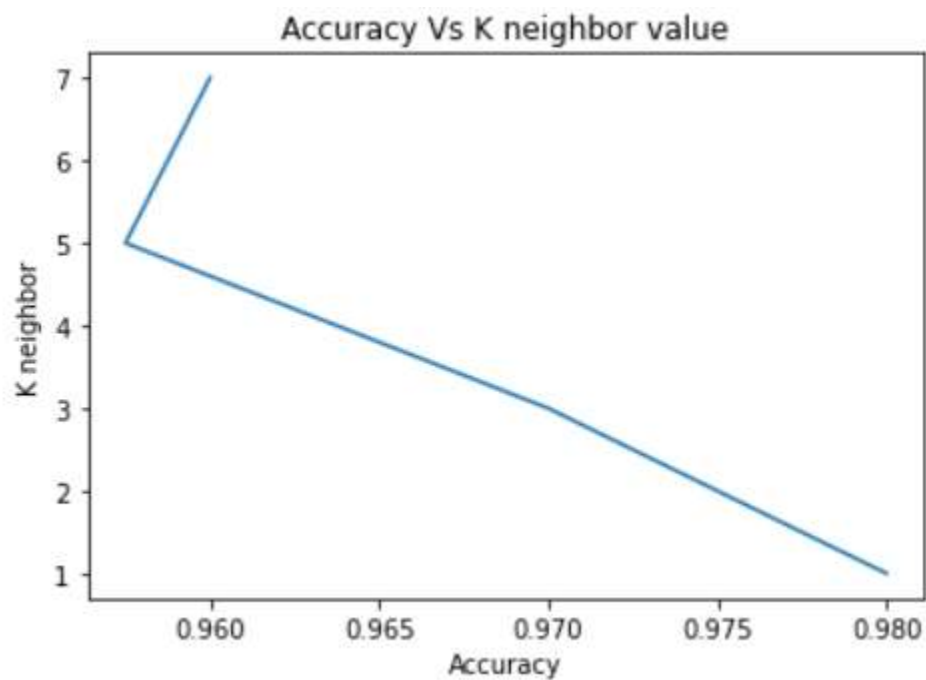
PCA result for alpha = 0.95 (different k's):



Accuracy for each alpha (different k's):

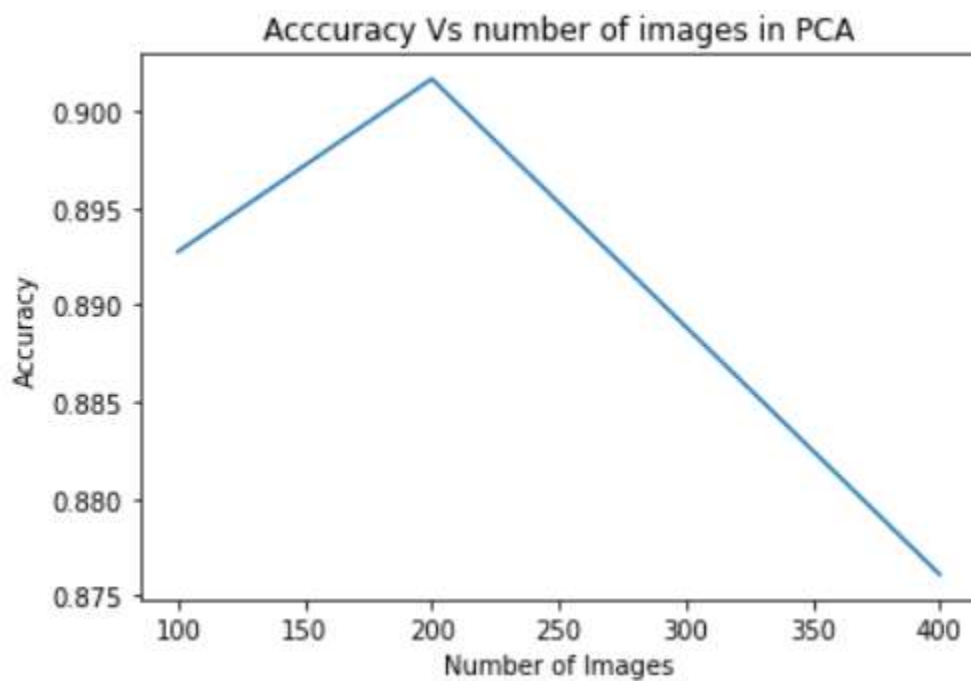


LDA results with different k's:

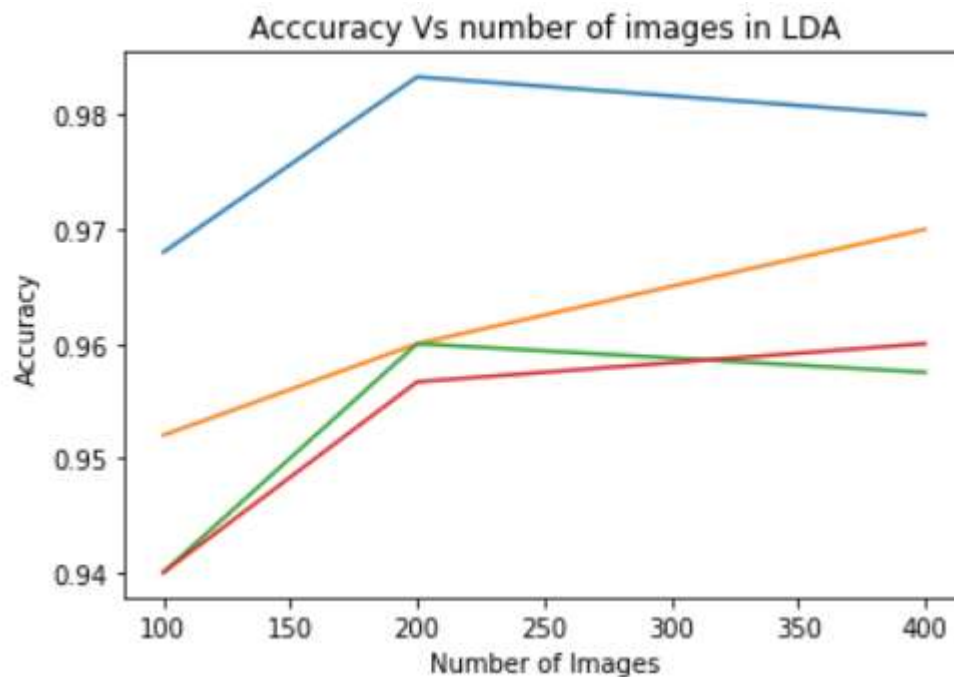


Plot the accuracy vs the number of non-faces images while fixing the number of face images.

PCA:



LDA:



Note: When we increased the number of non-face images from 100 to 200, accuracy got better, but when we doubled that to 400, accuracy decreased again but still better than 100.


Tried changing the splitting form 50% to 70% train and 30% test:

Change the number of instances per subject to be 7 and keep 3 instances per subject for testing. compare the results you have with the ones you got earlier with 50% split. This was done using this function.

A method to split the data into 7 images for training and 3 images for testing

```
def split_train_test73():  
    i = 0  
    for face, label in zip(data, data_labels):  
        if i < 7:  
            train_data73.append(face)  
            train_data_labels73.append(label)  
        elif i >= 7:  
            test_data73.append(face)  
            test_data_labels73.append(label)  
        i = i + 1  
        if(i >= 10):  
            i = 0  
  
split_train_test73()
```

Another LDA function was also made to cope with new splitting strategy:

```
 def LDA_73():
    d_train2 = []
    for i in range(40):
        d_train2.append([])
    j = -1
    for i in range(280):
        if (i % 7 == 0):
            j = j + 1
            d_train2[j].append(d_train73[i])
    d_train2 = np.asarray(d_train2, dtype="int32")

    means = np.zeros((40, 10304))
    between = np.zeros((10304, 10304))
    within = np.zeros((10304, 10304))
    center = np.zeros((40, 7, 10304))
    means = np.mean(d_train2, axis = 1)
    totalMean = np.mean(d_train2, axis=0)

    for i in range(40):
        meanDif = means[i] - totalMean
        between += (7*(np.dot(meanDif.T, meanDif)))

    for i in range(40):
        center[i] = d_train2[i] - means[i]

    for i in range(40):
        within += np.dot(center[i].T, center[i])
```

```
within = np.asarray(within)
inverse = np.linalg.inv(within)
s_inv_b = np.dot(inverse, between)

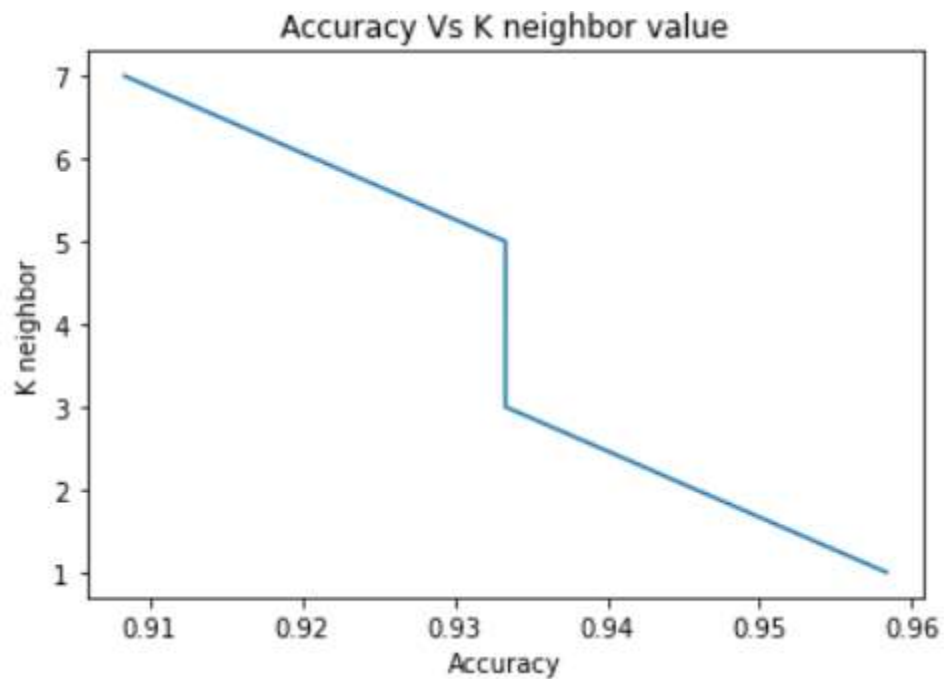
eigenvalues_LDA, eigenvectors_LDA = np.linalg.eig(s_inv_b)
idx = eigenvalues_LDA.argsort()
eigenvalues_LDA.sort()
eigenvectors_LDA2 = eigenvectors_LDA[:, idx]
eigenvectors_LDA2 = eigenvectors_LDA2[:, len(eigenvalues_LDA)-39:]
eigenvectors_LDA_real=np.real(eigenvectors_LDA2)

U_Train_LDA = np.dot(d_train73 , eigenvectors_LDA_real)
U_Test_LDA = np.dot(d_test73 , eigenvectors_LDA_real)

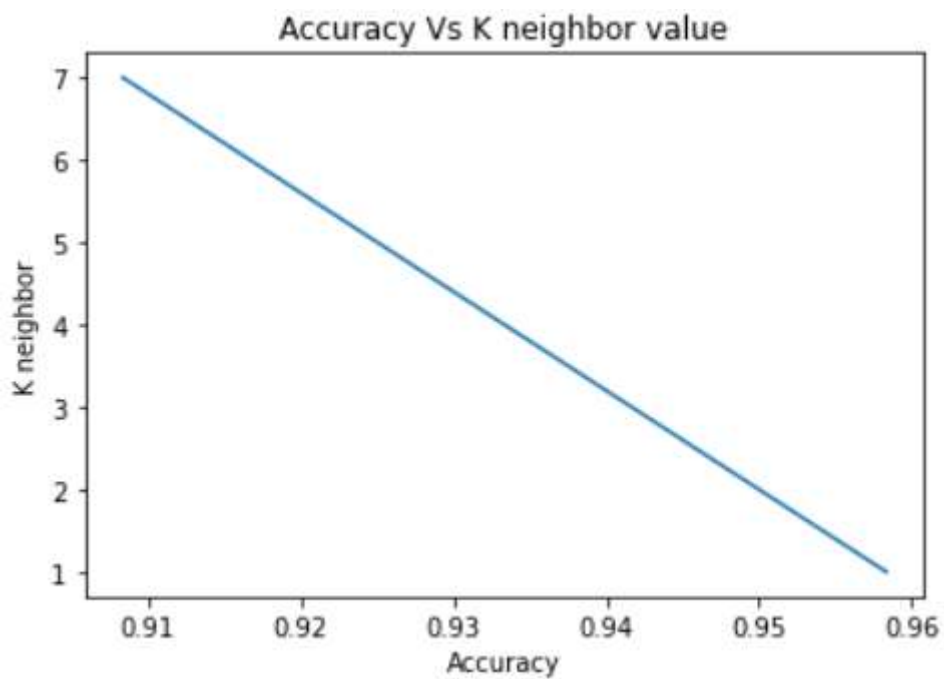
KNN(U_Train_LDA.real.T, train_data_labels73, U_Test_LDA.real.T, test_data_labels73)
```

Then the newly split data was tested using PCA and LDA

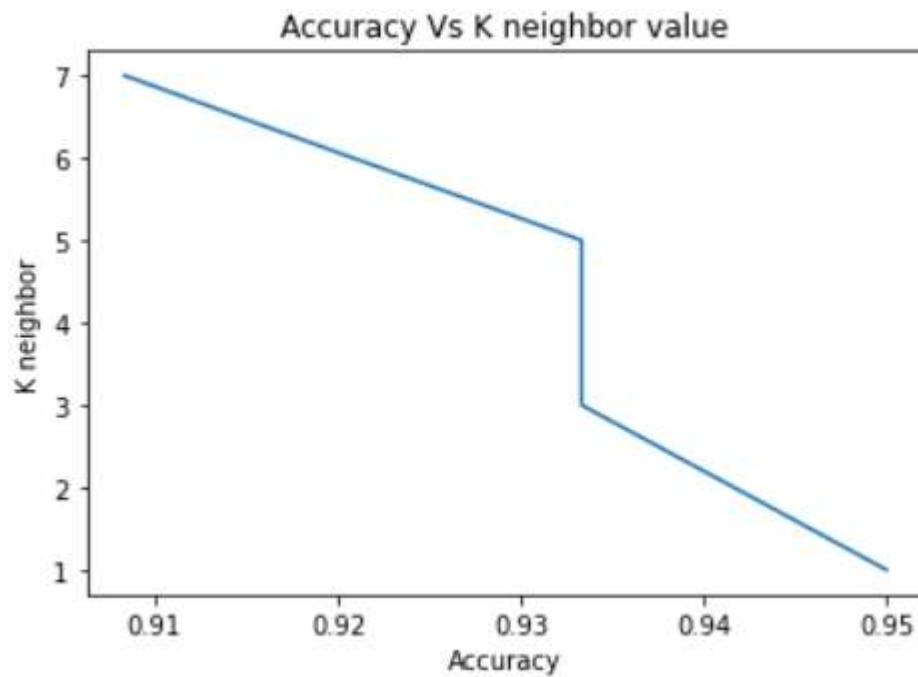
PCA result for $\alpha = 0.8$ (different k's):



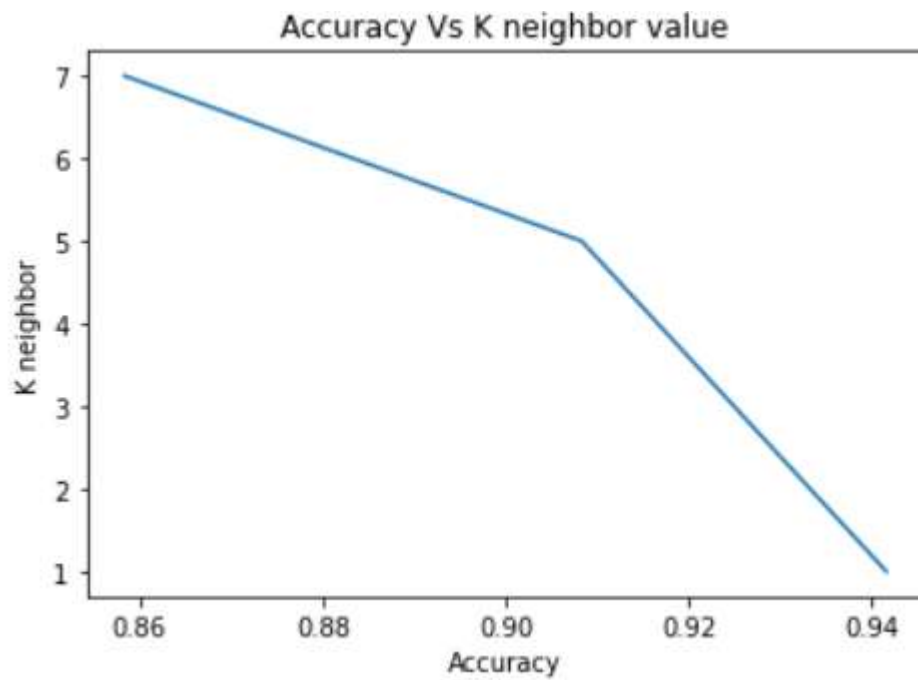
PCA result for $\alpha = 0.85$ (different k's):



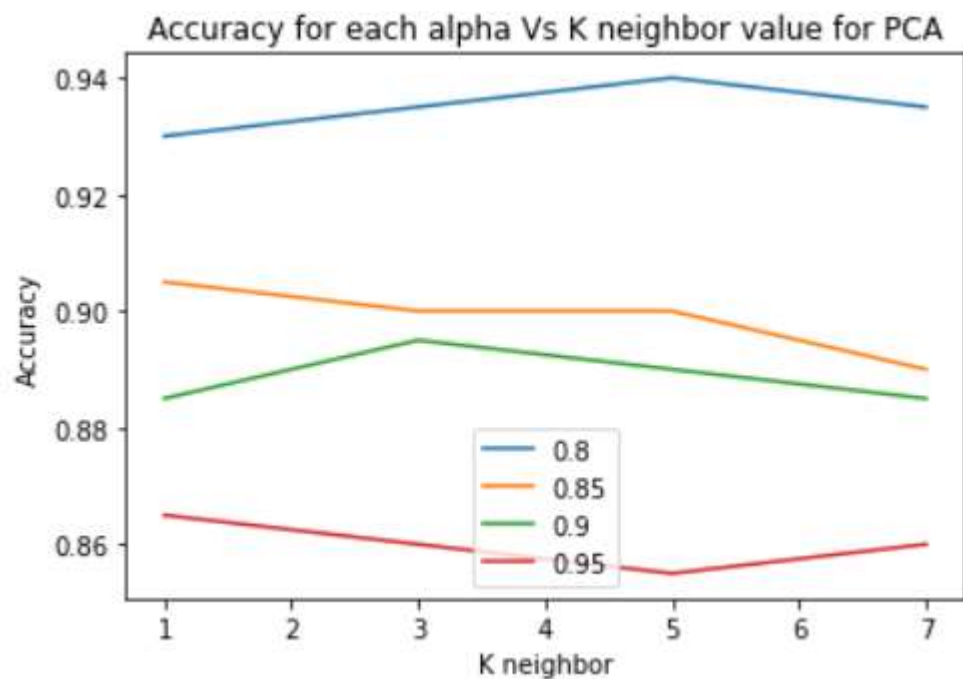
PCA result for alpha = 0.9 (different k's):



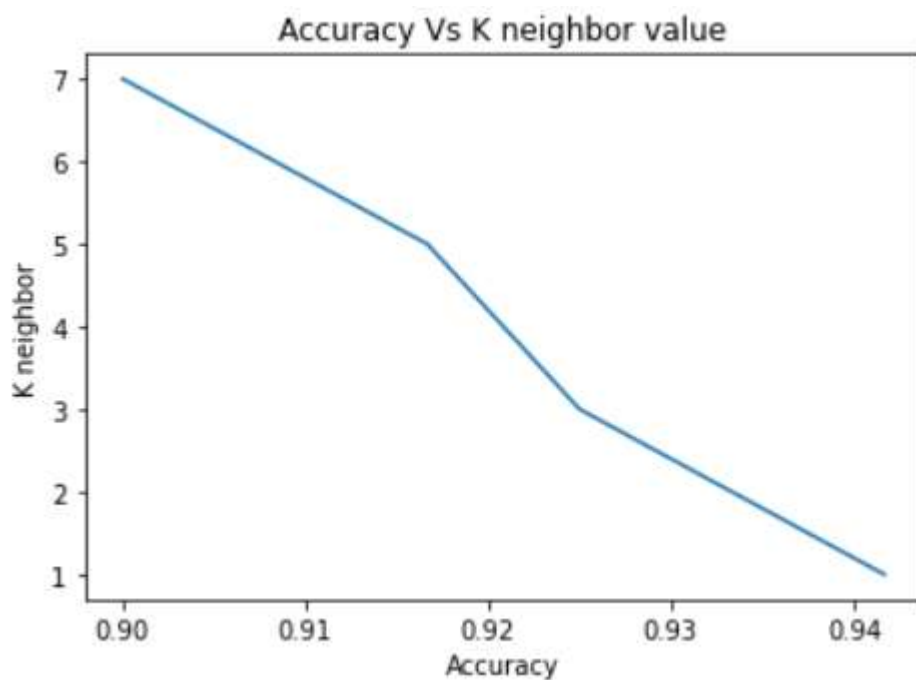
PCA result for alpha = 0.95 (different k's):



Accuracy for each alpha (different k's):



LDA results with different k's:



Note: Given the new data splitting, 7 images for training and 3 for testing instead of 5 each, it was noted that the accuracy of the 7-3 split was better as the model had more data to train on so it could easily classify the remaining unknown data.