Alexandria University
Faculty of Engineering
Computer and Communication Program
Spring 2021
CC484 Pattern Recognition

Assignment "1"
Face Recognition

Name	ID
Assem Salama	5669
Mostafa Talaat	5401
Ahmed Swidan	5588
Youssef Nader	5332

Introduction

Face verification involves a one-to-one match that compares a query face image against a template face image whose identity is being claimed. Face identification involves one-to-many matches that compare a query face image against all the template images in the database to determine the identity of the query face. Another face recognition scenario involves a watch-list check, where a query face is matched to a list of suspects (one-to-few matches). As per Hietmeyer, face recognition is one of the most effective biometric techniques for travel documents and scored higher on several evaluation parameters.

Computational models of face recognition must address several difficult problems. This difficulty arises from the fact that faces must be represented in a way that best utilizes the available face information to distinguish a particular face from all other faces. The problem of dimensionality reduction arises in face recognition because an m X n face image is reconstructed to form a column vector of mn components, for computational purposes. As the number of images in the data set increases, the complexity of representing data sets increases. Analysis with a large number of variables generally consumes a large amount of memory and computation power.

Dimensionality Reduction

Efforts are on for efficient storage and retrieval of images. Considerable progress has happened in face recognition with newer models especially with the development of powerful models of face appearance. These models represent faces as points in high-dimensional image spaces and employ dimensionality reduction to find a more meaningful representation, therefore, addressing the issue of the “curse of dimensionality”. Dimension reduction is a process of reducing the number of variables under observation. The need for dimension reduction arises when there is a large number of univariate data points or when the data points themselves are observations of a high dimensional variable. The key observation is that although face images can be regarded as points in a high-dimensional space, they often lie on a manifold (i.e., subspace) of much lower dimensionality, embedded in the high-dimensional image space. The main issue is how to properly define and determine a low-dimensional subspace of face appearance in a high-dimensional image space.

Dimensionality reduction techniques using linear transformations have been very popular in determining the intrinsic dimensionality of the manifold as well as extracting its principal directions. Dimensionality reduction is an effective approach to downsizing data. In statistics, dimension reduction is the process of reducing the number of random variables under consideration, $RN \rightarrow RM$ $(M<N)$ and can be divided into feature selection and feature extraction.

Feature selection is choosing a subset of all the features

[x1 x2 … xn] Feature selection [xi1 xi2 … xim]

Feature extraction is creating new features from existing ones

[x1 x2 … xn] Feature extraction [y1 y2 … ym]

In either case, the goal is to find a low-dimensional representation of the data while still describing the data with sufficient accuracy.

For reasons of computational and conceptual simplicity, the representation is often sought as a linear transformation of the original data. In other words, each component of the representation is a linear combination of the original variables. Well-known linear transformation methods include principal component analysis, factor analysis, and projection pursuit. Independent component analysis (ICA) is a recently developed method in which the goal is to find a linear representation of non-gaussian data so that the components are statistically independent, or as independent as possible. Such a representation seems to capture the essential structure of the data in many applications, including feature extraction and signal separation.

Several techniques exist to tackle the curse of dimensionality out of which some are linear methods and others are nonlinear. PCA, LDA, LPP are some popular linear methods and nonlinear methods include ISOMAP & Eigenmaps. PCA and LDA are the two most widely used subspace learning techniques for face recognition. These methods project the training sample faces to a low dimensional representation space where the recognition is carried out. The main supposition behind this procedure is that the face space (given by the feature vectors) has a lower dimension than the image space (given by the number of pixels in the image), and that the recognition of the faces can be performed in this reduced space. PCA has the advantage of capturing holistic features but ignore the localized features. Fisher faces from LDA technique extracts discriminating features between classes and is found to perform better for large data sets. Its shortcoming is that of Small Sample Space (SSS) problem. LPPs are linear projective maps that arise by solving variational problem that optimally preserves the neighborhood structure of the data set.

In many cases, face images may be visualized as points drawn on a low-dimensional manifold hidden in a high-dimensional ambient space. Specially, we can consider that a sheet of rubber is crumpled into a (high-dimensional) ball. The objective of a dimensionality-reducing mapping is to unfold the sheet and to make its low-dimensional structure explicit. If the sheet is not torn in the process, the mapping is topology-preserving. Moreover, if the rubber is not stretched or compressed, the mapping preserves the metric structure of the original space.

PCA is guaranteed to discover the dimensionality of the manifold and produces a compact representation. Turk and Pentland use Principal Component Analysis to describe face images in terms of a set of basis functions, or “eigenfaces”. LDA is a supervised learning algorithm. LDA searches for the project axes on which the data points of different classes are far from each other while requiring data points of the same class to be close to each other. Unlike PCA which encodes information in an orthogonal linear space, LDA encodes discriminating information in a linear separable space using bases are not necessarily orthogonal. It is generally believed that algorithms based on LDA are superior to those based on PCA. However, some recent work shows that, when the training dataset is small, PCA can outperform LDA, and also that PCA is less sensitive to different training datasets.

Recently, a number of research efforts have shown that the face images possibly reside on a nonlinear submanifold. However, both PCA and LDA effectively see only the Euclidean structure. They fail to discover the underlying structure, if the face images lie on a nonlinear submanifold hidden in the image space. Some nonlinear techniques have been proposed to discover the nonlinear structure of the manifold, e.g. Isomap, LLE and Laplacian Eigenmap. These nonlinear methods do yield impressive results on some benchmark artificial data sets. However, they yield maps that are defined only on the training data points and how to evaluate the maps on novel test data points remains unclear.

Code Samples and Explanation

First, the code starts by collecting data from the “at&t face dataset” file resulting an array of D400x10304 and labeling it.

```
for i in range(1, 41):
    images = os.listdir('./att_faces/s'+str(i))
    for image in images:
        img =
cv2.imread('./att_faces/s'+str(i)+"/"+image, 0)
        height1, width1 = img.shape[:2]
        img_col = np.array(img,
dtype='float64').flatten()
        subject = int(i)
        x.append(img_col)
        y.append(subject)

x = np.array(x)
y = np.array(y)
```

Second, Splitting the data into training and testing data.

```
trainingSet = []
testingSet = []
test_training_labels = np.repeat(np.arange(1, 41), 5)
for i in range(0, 400):
    if i % 2 != 0:
        trainingSet.append(x[i])
    else:
        testingSet.append(x[i])
trainingSet = np.array(trainingSet)
testingSet = np.array(testingSet)
```

Third, performing the PCA with the collected data as below.

```
knn = 1
clf = KNN.KNN(knn)
print("Using PCA with k-nearest neighbour of value ",
knn, ' : ')
for alpha in [0.8, 0.85, 0.9, 0.95]:
    pca = PCA.PCA(alpha)
    pca.fit(x)
    projected_training_data =
pca.transform(trainingSet)
    projected_testing_data = pca.transform(testingSet)
    clf.fit(projected_training_data,
test_training_labels)
    predictions = clf.predict(projected_testing_data)
    accuracy = np.sum(test_training_labels ==
predictions) / len(test_training_labels) * 100
    print("    - accuracy for alpha = ", alpha, " is ",
accuracy, "%")
```

PCA Algorithm:

```
import numpy as np

class PCA:

    def __init__(self, alpha):
        self.alpha = alpha
        self.components = None
        self.mean = None
        self.n_components = None

    def fit(self, X):
        # Mean centering
        self.mean = np.mean(X, axis=0)
        X = X - self.mean
        # covariance, function needs samples as columns
        cov = np.cov(X.T)
        # eigenvalues, eigenvectors
        eigenvalues, eigenvectors = np.linalg.eigh(cov)
        # -> eigenvector v = [:,i] column vector,
transpose for easier calculations
        # sort eigenvectors
        eigenvectors = eigenvectors.T
        idxs = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]
        # find number of n_components satisfying alpha
        self.n_components = 1
        while(self.n_components):
            f =
sum(eigenvalues[:self.n_components])/sum(eigenvalues)
            if f < self.alpha:
                self.n_components += 1
            else:
                break
        # store first n eigenvectors (projection
matrix)
        self.components =
eigenvectors[0:self.n_components]

    def transform(self, X):
        # project data
        X = X - self.mean
        return np.dot(X, self.components.T)
```

Fourth, performing the LDA with the collected data as below.

```
knn = 1
clf = KNN.KNN(knn)
print("Using LDA with k-nearest neighbour of value ",
knn, ' : ')
lda = LDA.LDA(39)
lda.fit(x,y)
projected_training_data = lda.transform(trainingSet)
projected_testing_data = lda.transform(testingSet)
clf.fit(projected_training_data, test_training_labels)
predictions = clf.predict(projected_testing_data)
accuracy = np.sum(test_training_labels == predictions)
```

```
/ len(test_training_labels) * 100
print("    - accuracy is ", accuracy, "%")
```

LDA Algorithm:

```
import numpy as np

class LDA:

    def __init__(self, n_components):
        self.n_components = n_components
        self.linear_discriminants = None

    def fit(self, X, y):
        n_features = X.shape[1]
        class_labels = np.unique(y)

        # Within class scatter matrix:
        # SW = sum((X_c - mean_X_c)^2 )

        # Between class scatter:
        # SB = sum( n_c * (mean_X_c - mean_overall)^2 )

        mean_overall = np.mean(X, axis=0)
        SW = np.zeros((n_features, n_features))
        SB = np.zeros((n_features, n_features))
        for c in class_labels:
            X_c = X[y == c]
            mean_c = np.mean(X_c, axis=0)
            # (4, n_c) * (n_c, 4) = (4,4) -> transpose
            SW += (X_c - mean_c).T.dot((X_c - mean_c))

            # (4, 1) * (1, 4) = (4,4) -> reshape
            n_c = X_c.shape[0]
            mean_diff = (mean_c -
mean_overall).reshape(n_features, 1)
            SB += n_c * (mean_diff).dot(mean_diff.T)

        # Determine SW^-1 * SB
        A = np.linalg.pinv(SW).dot(SB)
        # Get eigenvalues and eigenvectors of SW^-1 *
SB
        eigenvalues, eigenvectors = np.linalg.eigh(A)
        # -> eigenvector v = [:,i] column vector,
transpose for easier calculations
        # sort eigenvalues high to low
        eigenvectors = eigenvectors.T
        idxs = np.argsort(abs(eigenvalues))[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]
        # store first n eigenvectors
        self.linear_discriminants =
eigenvectors[0:self.n_components]
        return self.linear_discriminants

    def transform(self, X):
        # project data
        X = X - self.mean
        return np.dot(X, self.linear_discriminants.T)
```

Classifier Tuning (K-NN) Class:

```
import numpy as np
from collections import Counter


def euclidean_distance(x1, x2):
        return np.sqrt(np.sum((x1 - x2)**2))


class KNN:

    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        # Compute distances between x and all examples
in the training set
```

```
        distances = [euclidean_distance(x, x_train) for
x_train in self.X_train]
        # Sort by distance and return indices of the
first k neighbors
        k_idx = np.argsort(distances)[:self.k]
        # Extract the labels of the k nearest neighbor
training samples
        k_neighbor_labels = [self.y_train[i] for i in
k_idx]
        # return the most common class label
        most_common =
Counter(k_neighbor_labels).most_common(1)
        return most_common[0][0]
```

Last, Comparing the non-face images with the Collected faces data.

```
import sys
import numpy as np
np.set_printoptions(threshold=sys.maxsize)
import os
import cv2
import pandas as pd
import PCA
import KNN
import LDA

testData = []
trainingSet = []
testingSet = []


for i in range(1,41):
    im2 = cv2.imread('./faces/'+str(i)+'.pgm', 0)
    img_col = np.array(im2, dtype='float64').flatten()
    testData.append(img_col)

for i in range(1,41):
    im2 = cv2.imread('./nonfaces/'+str(i)+'.jpg', 0)
    img_col = np.array(im2, dtype='float64').flatten()
    testData.append(img_col)
test_data_labels = np.repeat(np.arange(1, 3), 40)
```

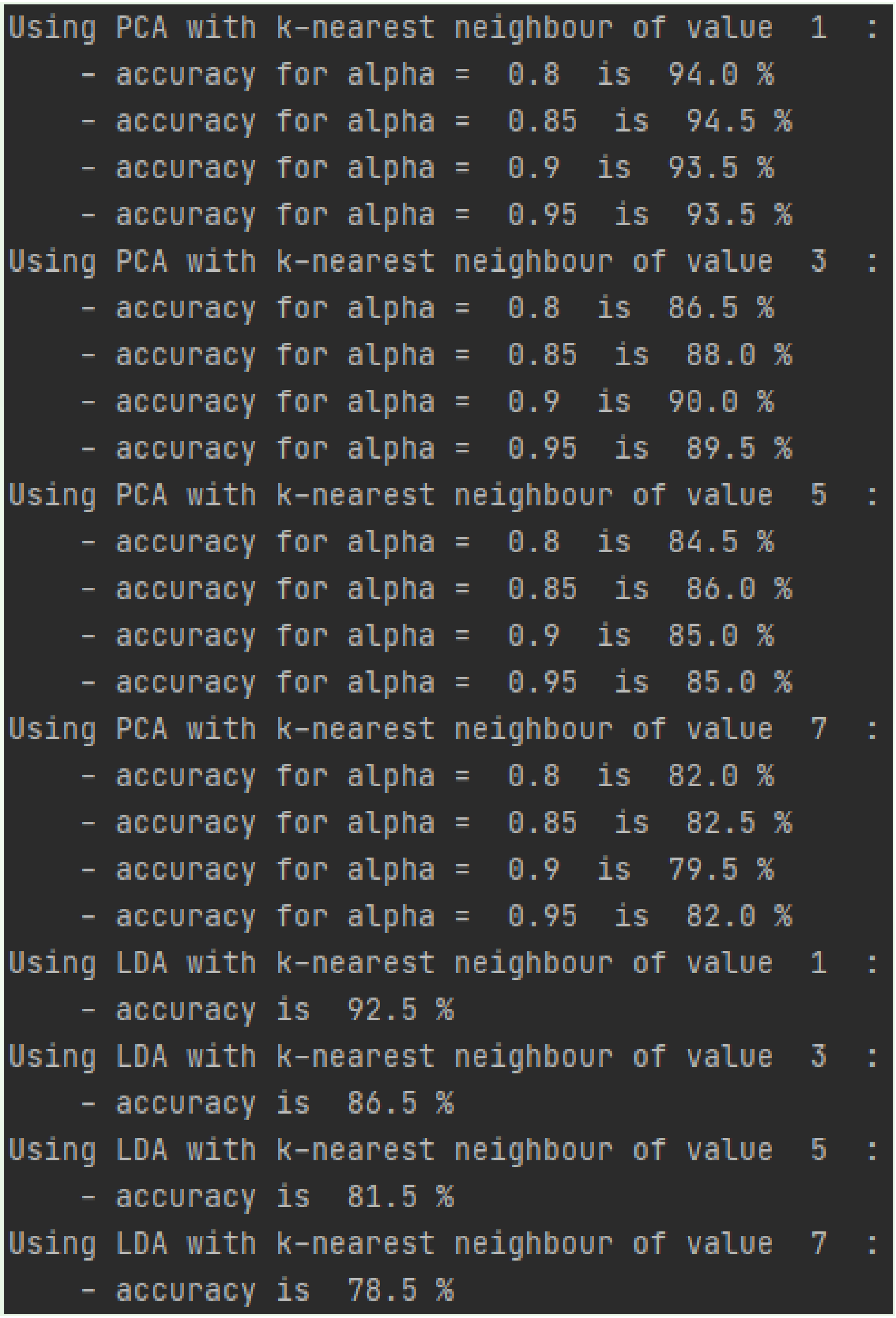
```
for i in range(0, 80):
    if i % 2 != 0:
        trainingSet.append(testData[i])
    else:
        testingSet.append(testData[i])

test_training_labels = np.repeat(np.arange(1, 3), 20)

testData = np.array(testData)
trainingSet = np.array(trainingSet)
testingSet = np.array(testingSet)

knn = 1
clf = KNN.KNN(knn)
print("Using LDA with k-nearest neighbour of value ",
knn, ' : ')
lda = LDA.LDA(1)
lda.fit(testData,test_data_labels)
projected_training_data = lda.transform(trainingSet)
projected_testing_data = lda.transform(testingSet)
clf.fit(projected_training_data, test_training_labels)
predictions = clf.predict(projected_testing_data)
accuracy = np.sum(test_training_labels == predictions)
/ len(test_training_labels) * 100
print("    - accuracy is ", accuracy, "%")
```

Test runs

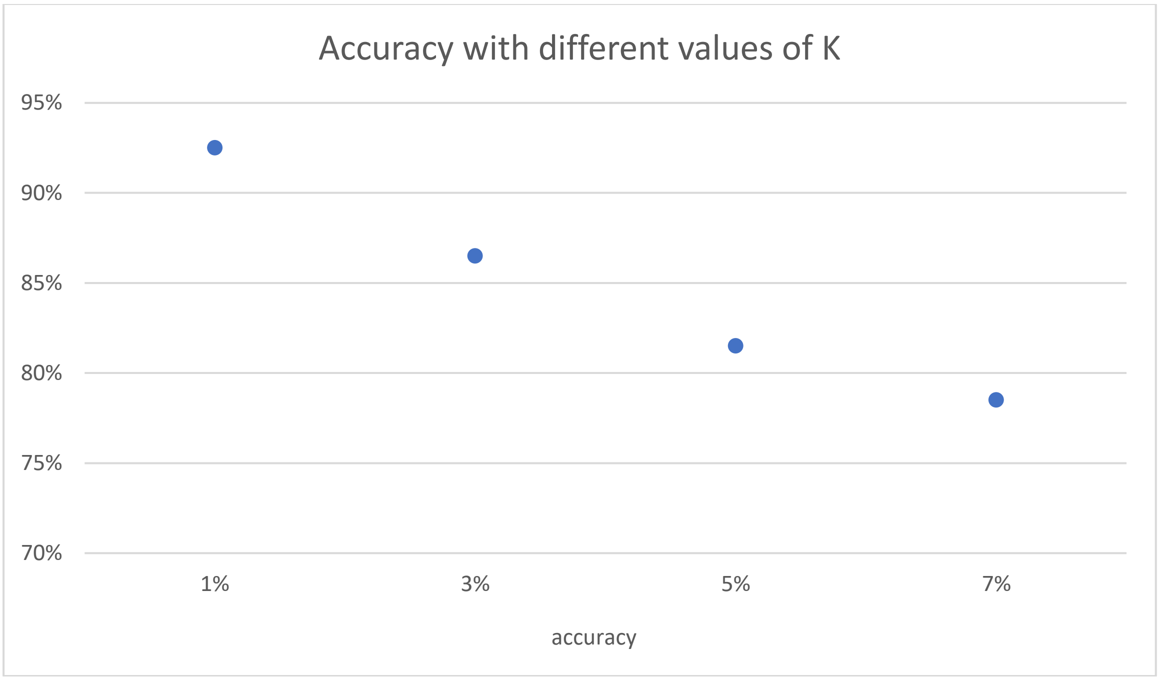
It was noticed in the PCA and LDA that when alpha is increased the accuracy of recognition increases as shown in the following runs.

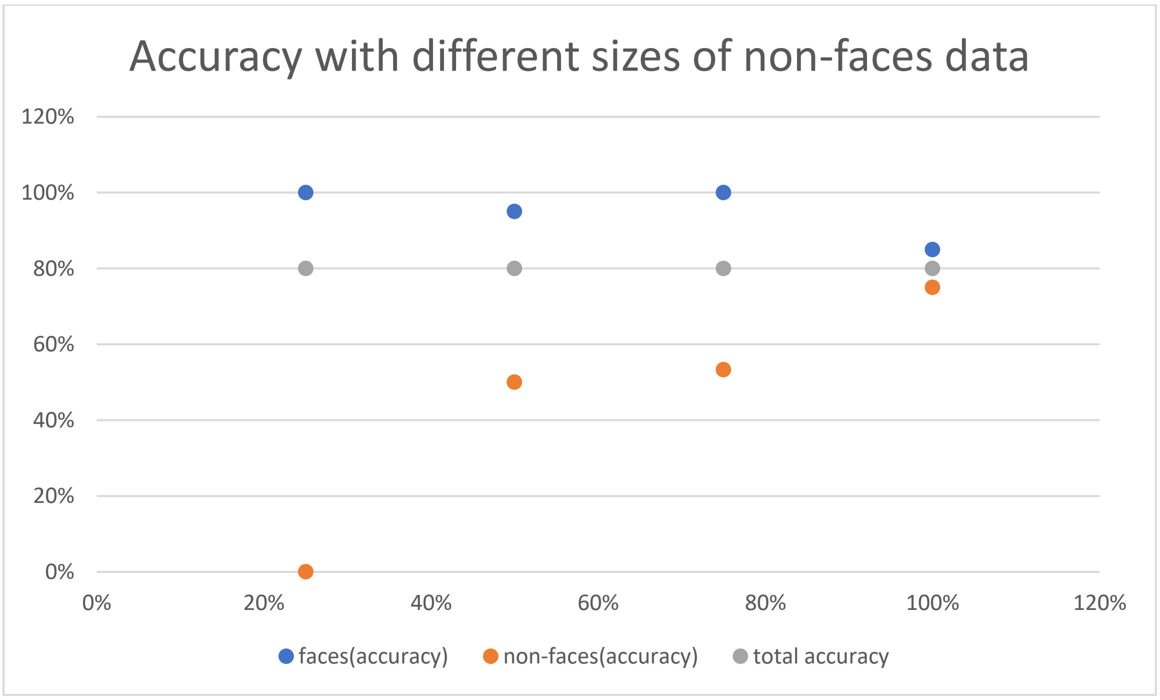
It was also noticed that when increasing the number of neighbors in KNN algorithm accuracy decreases overall.

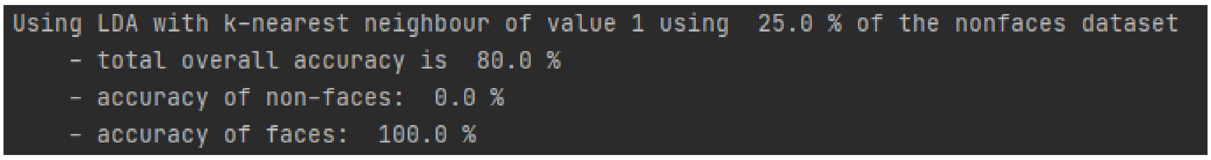
As for the non faces detection program we trained the program using 25, 50, 75 and 100% of the nonfaces test data while always using 100% of the faces data set. It was noticed that as we increase the number of non-faces data given to the program the accuracy increases.



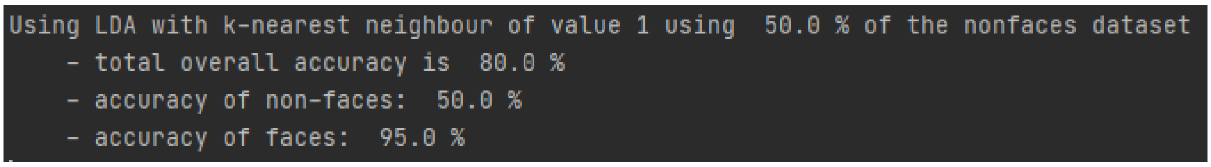
```
Using PCA with k-nearest neighbour of value  1  :
    - accuracy for alpha =  0.8  is  94.0 %
    - accuracy for alpha =  0.85  is  94.5 %
    - accuracy for alpha =  0.9  is  93.5 %
    - accuracy for alpha =  0.95  is  93.5 %
Using PCA with k-nearest neighbour of value  3  :
    - accuracy for alpha =  0.8  is  86.5 %
    - accuracy for alpha =  0.85  is  88.0 %
    - accuracy for alpha =  0.9  is  90.0 %
    - accuracy for alpha =  0.95  is  89.5 %
Using PCA with k-nearest neighbour of value  5  :
    - accuracy for alpha =  0.8  is  84.5 %
    - accuracy for alpha =  0.85  is  86.0 %
    - accuracy for alpha =  0.9  is  85.0 %
    - accuracy for alpha =  0.95  is  85.0 %
Using PCA with k-nearest neighbour of value  7  :
    - accuracy for alpha =  0.8  is  82.0 %
    - accuracy for alpha =  0.85  is  82.5 %
    - accuracy for alpha =  0.9  is  79.5 %
    - accuracy for alpha =  0.95  is  82.0 %
Using LDA with k-nearest neighbour of value  1  :
    - accuracy is  92.5 %
Using LDA with k-nearest neighbour of value  3  :
    - accuracy is  86.5 %
Using LDA with k-nearest neighbour of value  5  :
    - accuracy is  81.5 %
Using LDA with k-nearest neighbour of value  7  :
    - accuracy is  78.5 %
```


Accuracy with different values of K
95%
90%
85%
80%
75%
70%
1%
3%
5%
7%
accuracy

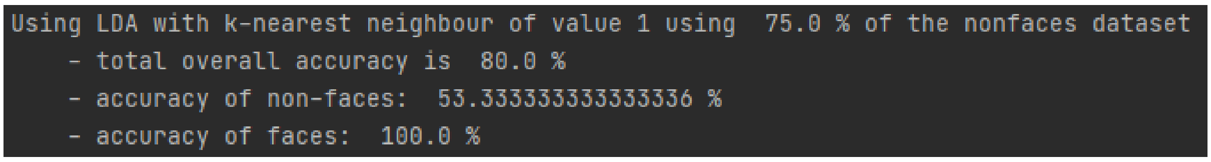

Accuracy with different sizes of non-faces data
120%
100%
80%
60%
40%
20%
0%
0%
20%
40%
60%
80%
100%
120%
● faces(accuracy)
● non-faces(accuracy)
● total accuracy



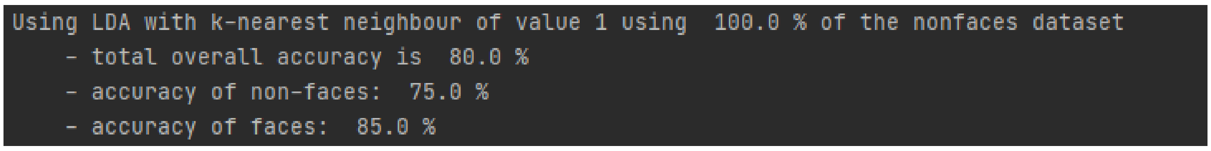
```
Using LDA with k-nearest neighbour of value 1 using  25.0 % of the nonfaces dataset
    - total overall accuracy is  80.0 %
    - accuracy of non-faces:  0.0 %
    - accuracy of faces:  100.0 %
```



```
Using LDA with k-nearest neighbour of value 1 using  50.0 % of the nonfaces dataset
    - total overall accuracy is  80.0 %
    - accuracy of non-faces:  50.0 %
    - accuracy of faces:  95.0 %
```



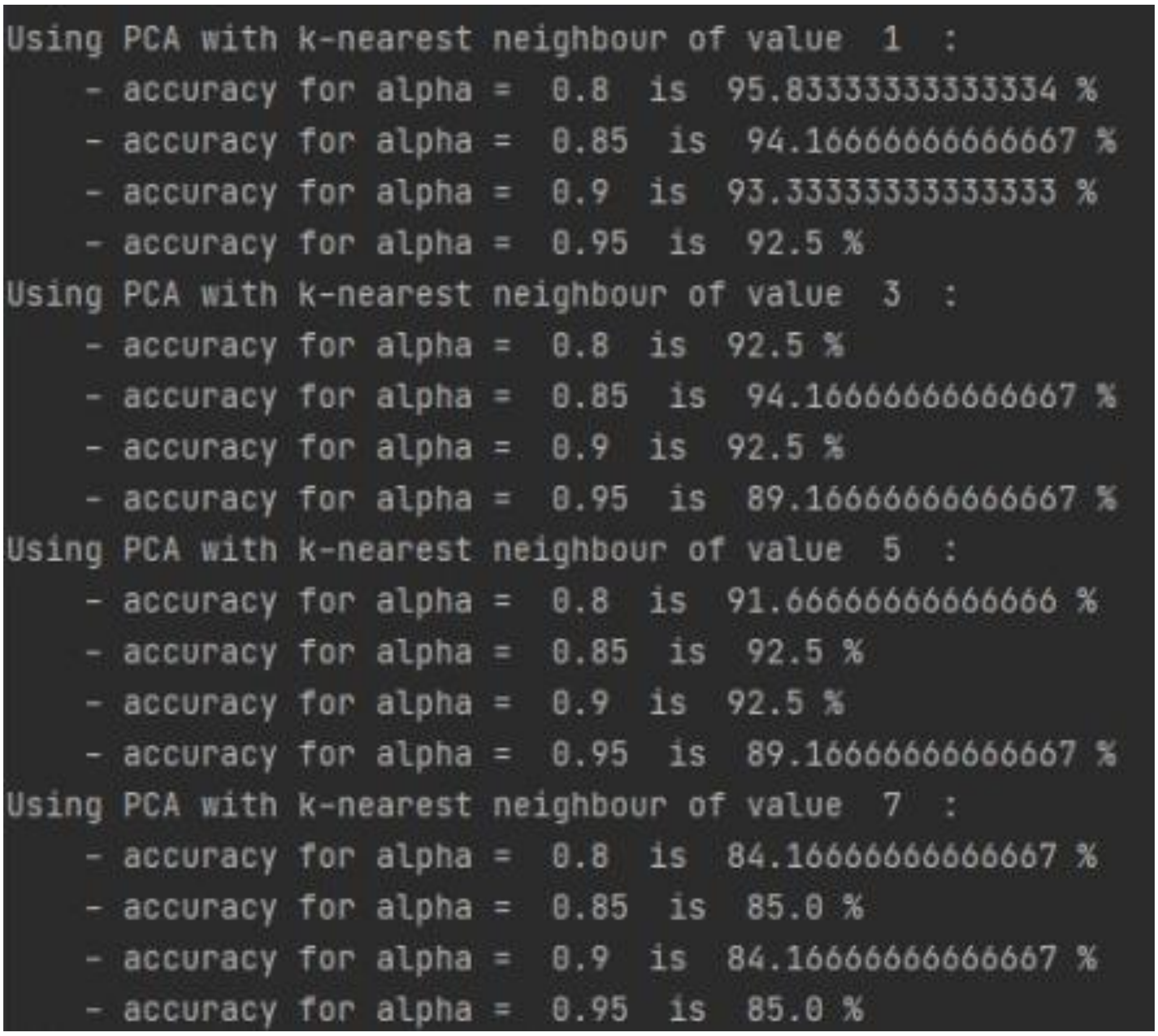
```
Using LDA with k-nearest neighbour of value 1 using  75.0 % of the nonfaces dataset
    - total overall accuracy is  80.0 %
    - accuracy of non-faces:  53.333333333333336 %
    - accuracy of faces:  100.0 %
```



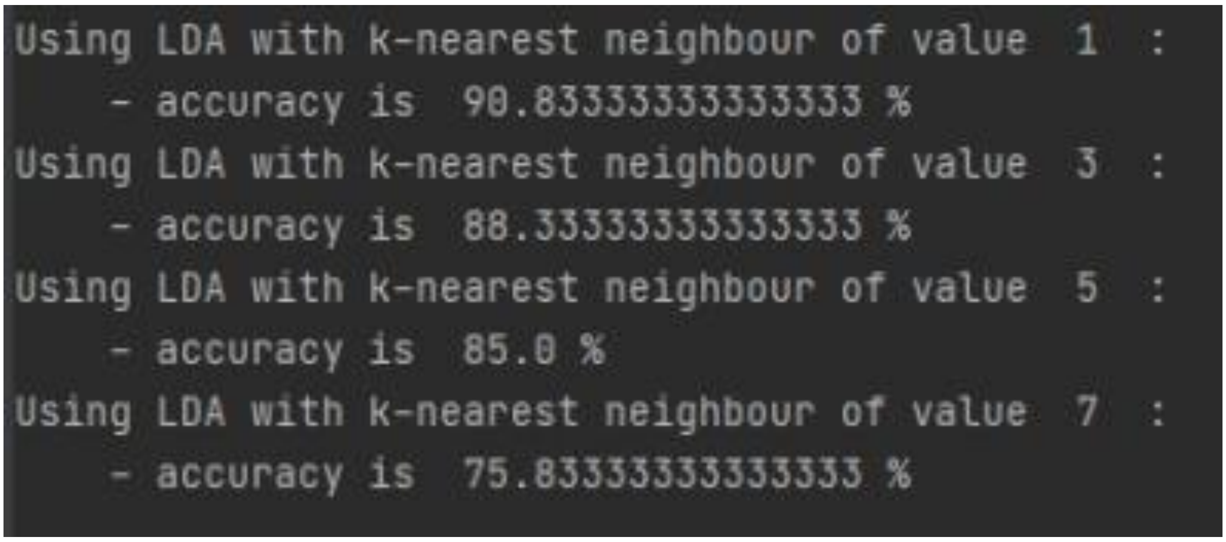
```
Using LDA with k-nearest neighbour of value 1 using  100.0 % of the nonfaces dataset
    - total overall accuracy is  80.0 %
    - accuracy of non-faces:  75.0 %
    - accuracy of faces:  85.0 %
```

Bonus

When we changed the training test ratio to 70-30 instead of 50-50 there was a very good spike in accuracy as the program was refined more by the training data and it had less chance of failing in the test data



```
Using PCA with k-nearest neighbour of value  1  :
    - accuracy for alpha =  0.8  is  95.83333333333334 %
    - accuracy for alpha =  0.85  is  94.16666666666667 %
    - accuracy for alpha =  0.9  is  93.33333333333333 %
    - accuracy for alpha =  0.95  is  92.5 %
Using PCA with k-nearest neighbour of value  3  :
    - accuracy for alpha =  0.8  is  92.5 %
    - accuracy for alpha =  0.85  is  94.16666666666667 %
    - accuracy for alpha =  0.9  is  92.5 %
    - accuracy for alpha =  0.95  is  89.16666666666667 %
Using PCA with k-nearest neighbour of value  5  :
    - accuracy for alpha =  0.8  is  91.66666666666666 %
    - accuracy for alpha =  0.85  is  92.5 %
    - accuracy for alpha =  0.9  is  92.5 %
    - accuracy for alpha =  0.95  is  89.16666666666667 %
Using PCA with k-nearest neighbour of value  7  :
    - accuracy for alpha =  0.8  is  84.16666666666667 %
    - accuracy for alpha =  0.85  is  85.0 %
    - accuracy for alpha =  0.9  is  84.16666666666667 %
    - accuracy for alpha =  0.95  is  85.0 %
```



```
Using LDA with k-nearest neighbour of value  1  :
    - accuracy is  90.83333333333333 %
Using LDA with k-nearest neighbour of value  3  :
    - accuracy is  88.33333333333333 %
Using LDA with k-nearest neighbour of value  5  :
    - accuracy is  85.0 %
Using LDA with k-nearest neighbour of value  7  :
    - accuracy is  75.83333333333333 %
```

Conclusion

Facial recognition is a way of identifying or confirming an individual’s identity using their face. Facial recognition systems can be used to identify people in photos, videos, or in real-time.

Facial recognition is a category of biometric security. Other forms of biometric software include voice recognition, fingerprint recognition, and eye retina or iris recognition. The technology is mostly used for security and law enforcement, though there is increasing interest in other areas of use.