ASSIGNMENT 1 PR

Face Recognition using LDA &PCA

Anas Emad	7048
Mazen Ahmed Ghanem	6896
Mohab Ayman	7127

2. Generate the Data Matrix and the Label vector

```python
def generate_data():
    g=input("Press 1 to generate  faces problem and 2 to generate non faces problem")
    if g=="1":
        Data = []
        label = []
        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)
                img_flat = np.array(img).ravel()

                subject = int(i)
                Data.append(img_flat)
                label.append(subject)

        test_data = []
        train_data = []

        test_label = []
        train_label = []
```

We have a for loop to get the list of files in our directory which will be from s1 to s40 each having 10 images for 1 of the 40 persons In the dataset.

For each image in the directory, the code uses the cv2.imread() function from the OpenCV library to read the image as a NumPy array.

We then flatten the images and append them to our data matrix and we take the label which is a number from 1 to 40 and we append it to our label vector

Split the Dataset into Training and Test sets

```python
    for x in range(400):

        if x % 2 == 0:
            test_data.append(Data[x])
            test_label.append(label[x])
        else:
            train_data.append(Data[x])
            train_label.append(label[x])
    test_data = np.array(test_data)
    train_data = np.array(train_data)

    test_label = np.array(test_label)
    train_label = np.array(train_label)

    return train_data, train_label, test_data,test_label
```

We split the data and labels of each person to have 5 images in the test dataset and 5 in the training dataset by splitting them into even and odd images and adding each image to the respective dataset by using the above for loop

```
Enter 1 to run PCA
Enter 2 to run LDA
Enter 3 to run Classifier Tuning
Enter 4 to run the bonus part
1
RUNING PCA :
dimension of cenetered data (200, 10304) (10304, 36)
alpha = 0.8
accuracy = 0.94


dimension of cenetered data (200, 10304) (10304, 52)
alpha = 0.85
accuracy = 0.945


dimension of cenetered data (200, 10304) (10304, 76)
alpha = 0.9
accuracy = 0.94


dimension of cenetered data (200, 10304) (10304, 117)
alpha = 0.95
accuracy = 0.935
```

4. Classification using PCA

ALGORITHM 7.1. Principal Component Analysis

PCA $(\mathbf{D}, \alpha)$:

$\mu = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ // compute mean

$\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \mu^T$ // center the data

$\Sigma = \frac{1}{n}(\mathbf{Z}^T\mathbf{Z})$ // compute covariance matrix

$(\lambda_1, \lambda_2, \ldots, \lambda_d) = \text{eigenvalues}(\Sigma)$ // compute eigenvalues

$\mathbf{U} = (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_d) = \text{eigenvectors}(\Sigma)$ // compute eigenvectors

$f(r) = \frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$, for all $r = 1, 2, \ldots, d$ // fraction of total variance

Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality

$\mathbf{U}_r = (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_r)$ // reduced basis

$\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{U}_r^T\mathbf{x}_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

```python
def pca(X, alpha):

    # Compute the mean of the data matrix
    X_mean = np.mean(X, axis=0)

    # Compute the centered data matrix
    X_centered = X - X_mean

    # Compute the covariance matrix
    C = np.cov(X_centered.T)

    # Compute the eigenvalues and eigenvectors of the covariance matrix
    eigenvalues, eigenvectors = np.linalg.eigh(C)

    # Sort the eigenvectors in descending order of the eigenvalues
    idx = np.absolute(eigenvalues).argsort()[::-1]
    eigenvectors = eigenvectors[:,idx]

    count = 0
    counter_iterator = 0
    for i in idx:
        count += eigenvalues[i] / sum(eigenvalues)
        counter_iterator += 1
```

We have a function pca(X, alpha): which takes X the data matrix and alpha :the explained variance. we calculate the mean for our data

Then we center the data by subtracting the mean from it.The covariance matrix is computed using **C = np.cov(X_centered.T)**

The eigen values and eigen vectors are calculated using eigenvalues, **eigenvectors = np.linalg.eigh(C)**

We sort the eigen values in descending order to know the eigen vectors with maximal variance in data and we capture their indexes.

We then sort eigen vectors according to the indexes

idx = np.absolute(eigenvalues).argsort()[::-1]

eigenvectors = eigenvectors[:,idx]

we create a for loop to take enough eigen values to be equal to or surpass alpha

we store the number of components in n_components .

U = eigenvectors[:, :n_components]

print("dim",X_centered.shape,U.shape)

X_reduced = np.matmul(X,U)

return X_reduced, U

we take the first n_components eigen vectors to create our projection matrix we project the data by multiplying the data matrix with the projection matrix and the function returns the projected data and the projection matrix

5. Classification Using LDA

ALGORITHM 20.1. Linear Discriminant Analysis

LINEARDISCRIMINANT ($\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$):

$\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n\}, i = 1, 2$ // class-specific subsets

$\boldsymbol{\mu}_i \leftarrow \text{mean}(\mathbf{D}_i), i = 1, 2$ // class means

$\mathbf{B} \leftarrow (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ // between-class scatter matrix

$\mathbf{Z}_i \leftarrow \mathbf{D}_i - \mathbf{1}_{n_i} \boldsymbol{\mu}_i^T, i = 1, 2$ // center class matrices

$\mathbf{S}_i \leftarrow \mathbf{Z}_i^T \mathbf{Z}_i, i = 1, 2$ // class scatter matrices

$\mathbf{S} \leftarrow \mathbf{S}_1 + \mathbf{S}_2$ // within-class scatter matrix

$\lambda_1, \mathbf{w} \leftarrow \text{eigen}(\mathbf{S}^{-1}\mathbf{B})$ // compute dominant eigenvector

```
def lda_org(data_matrix,train_label):

    y = 0
    mean_arr=[]
    centered_data=[]
    mean_vector = np.mean(data_matrix, axis=0)

    for i in range(5,205,5):
        data_matrix_modified=[]
        while(y<i):
            data_matrix_modified.append(data_matrix[y]) # collect every 5 rows together
            y=y+1
        mean_arr.append(np.mean(data_matrix_modified, axis=0))
        centered_data.append(np.subtract(data_matrix_modified,mean_arr[i//5 -1]))


    centered_data=np.reshape(centered_data, (200,10304))
    centered_data=np.array(centered_data)
```

```
mean_vector=np.array(mean_vector)
mean_arr = np.array(mean_arr)
class_label = train_label
class_label = np.array(class_label)
iet = 0
S_total = np.zeros([10304, 10304])
x_a = np.zeros([5, 10304])

for c in range(1,41):
    print("ietration",iet)
    iet=iet+1
    x_a = centered_data[class_label == c]
    S_total += np.matmul(x_a.T,x_a)
    print("B matrix",S_total)
```

```
data_matrix=np.array(data_matrix)
n_features = data_matrix.shape[1]
class_label=train_label
class_label=np.array(class_label)
s_b=np.zeros((n_features,n_features))
iet=0
for c in range(1,41):
    print("iteration",iet)
    iet=iet+1
    x_c = data_matrix[class_label == c]
    mean_c=np.mean(x_c,axis=0)
    mean_c=np.array(mean_c)
    n_c=x_c.shape[0]
    mean_diff=(mean_c-mean_vector.reshape(n_features,1)) #1*10304.reshape-->10304*1
    s_b += n_c * np.matmul(mean_diff,mean_diff.T)
    print("s_b",s_b)
```

```
# Use 39 eigenvectors
eigen_values,eigen_vectors = np.linalg.eigh( np.matmul(np.linalg.inv(S_total),s_b)  )
print("eigen vectors",eigen_vectors.shape)

idx = np.absolute(eigen_values).argsort()[::-1]
eigenvectors = eigen_vectors[:,idx]

u=eigenvectors[:,:39] # Projection matrix
print("U =",u.shape)


projection_data = np.matmul(data_matrix,u)
print("projection data",projection_data)

return projection_data,u
```

We have the function **lda_org(data_matrix,train_label):** which takes the data matrix and the test data we calculate the total mean .we then calculate the mean for each class(5 rows) .and we center the data for each class by subtracting its man from the original data.

We calculate the class scatter matrices for each class and add them together in S_total.

We calculate Sb by the equation given above.

We use 39 eigen vectors for the projection matrix and we calculate the projected data and we return the projected data and the projection matrix

```
   5.38552247e+00  1.83333843e+02]]
accuracy = 0.955
Enter 1 to run PCA
Enter 2 to run LDA
Enter 3 to run Classifier Tuning
Enter 4 to run the bonus part
```

Lda gave us an accuracy of 0.955 which is slightly better than pca

6) classifier Tuning:

a)After running PCA,LDA algorithm we get the projected trained data and project the test data using the same projection matrix ,then use K-NN classifier with values

[i= 1,3,5,7] to get the k with best accuracy.

knn_pca = KNeighborsClassifier(n_neighbors=i)

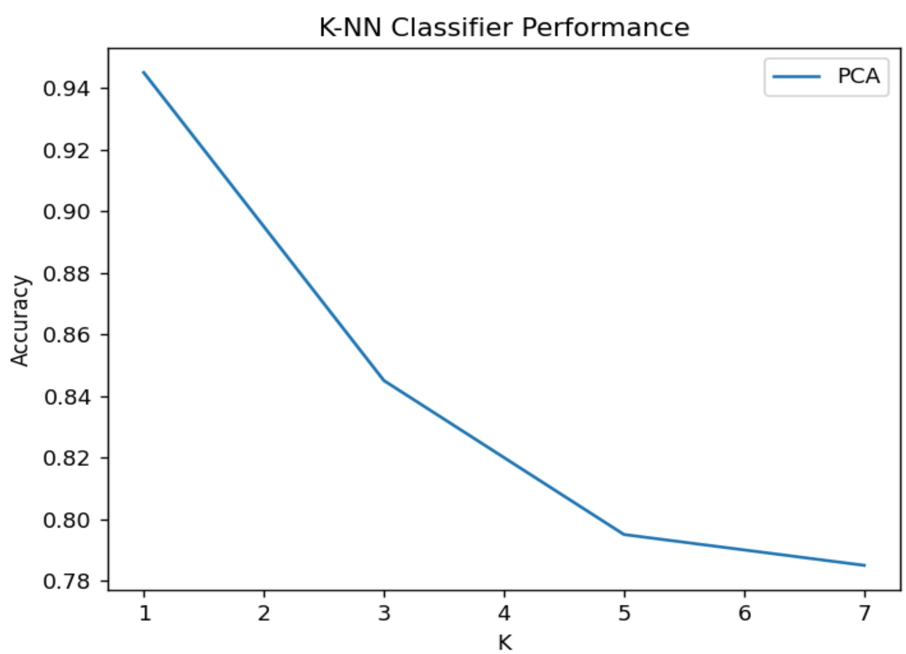
b)there are multiple opinions in this part as in the scikit-learn the function chooses the class with the largest num of samples

another opinion is that the function chooses it randomly

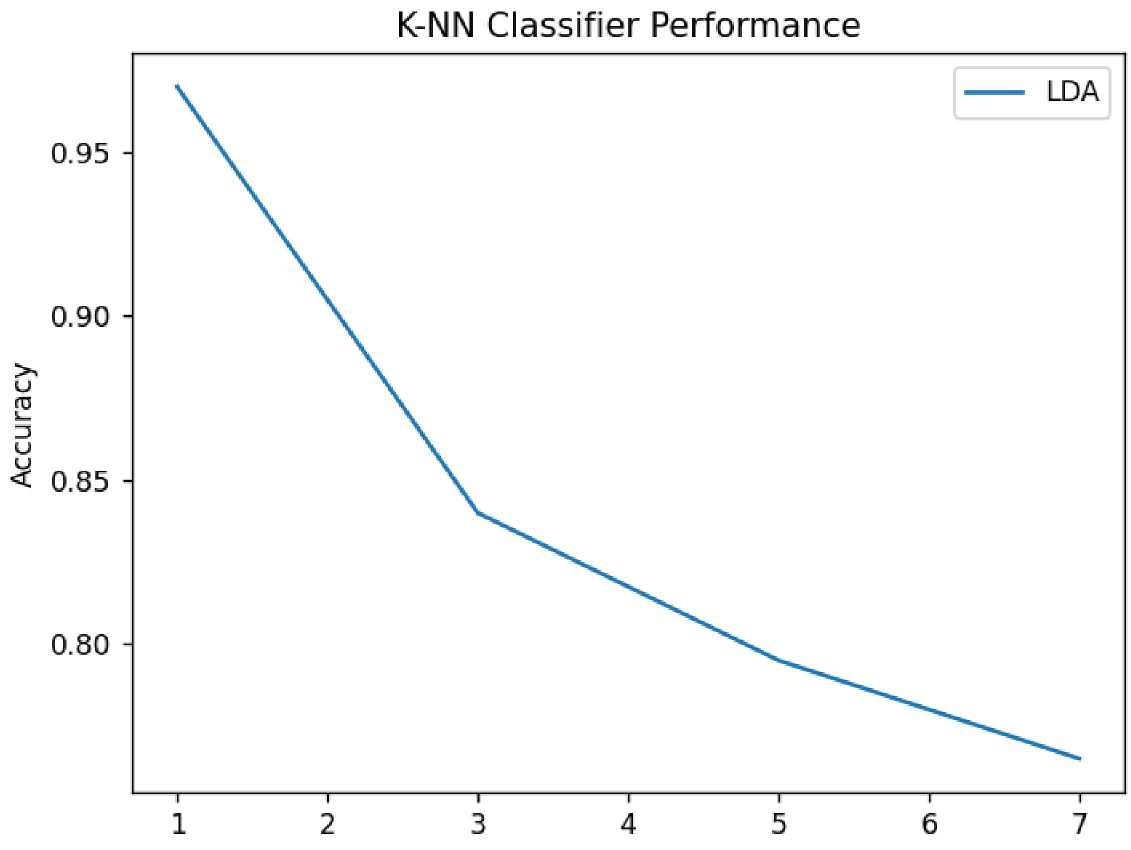
and the third opinion is to choose the least weight of the distances.

c)PLOTS:

1)PCA:



2)LDA:



7)compare non-faces and faces:

a)first we loaded a dataset contains photos of flowers, dogs and cars ,then converted them to grayscale and resized the images to 92x112 then made the same operations as the original faces dataset to flatten the images and stack them into 1 data matrix and assign labels for the samples but in this case we have only 2 classes as it is a binary classification problem either face or non face .

next we split the data into train and test each will be 200*10304 matrix.

i) we project the test and train data then get the confusion the matrix to get the number of failure and success cases after the prediction of the test data using K-NN using k=1.

```
  1391.524   ]
 ...
 [1442.182    1507.065    1436.54     ... 1883.56     1860.558
  1824.536   ]
 [1424.5701   1488.66075  1418.997    ... 1860.558    1837.8369
  1802.2548  ]
 [1396.9892   1459.839    1391.524    ... 1824.536    1802.2548
  1767.3616  ]]
eigen values 10304
eigen vectors (10304, 10304)
U = (10304, 1)
accuracy = 0.78
True Positive(TP)  =  153
False Positive(FP) =  41
True Negative(TN)  =  159
False Negative(FN) =  47
Accuracy of the binary classifier = 0.780
```

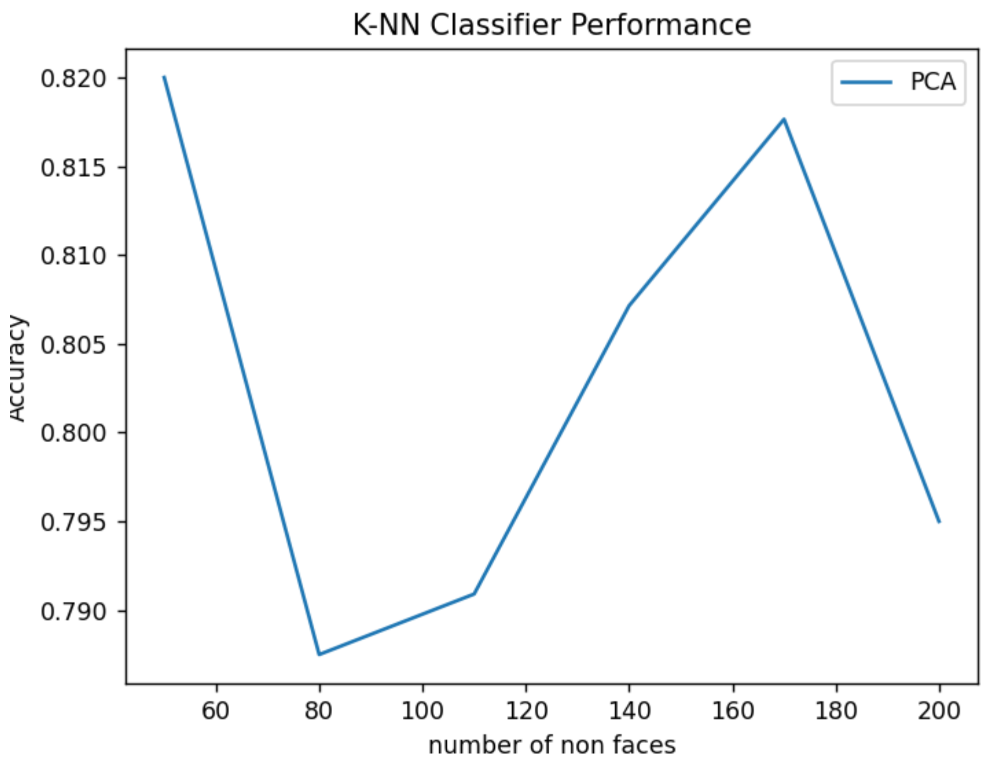
ii) the number of eigen vectors used in the projection matrix is equal to the number of [classes - 1] so in our case the number of classes is 2 and the number of the eigen vectors is 1 keeping in mind in case of increasing the number of eigen vectors the accuracy will increase.

iii)Plot fixed num of faces and variable number of non faces:

here we change the number of the nonfaces in test projection starting from 50 and every iteration we add more 30 non faces.

```
  1802.2548  ]
 [1396.9892   1459.839    1391.524    ... 1824.536    1802.2548
  1767.3616  ]]
eigen values 10304
eigen vectors (10304, 10304)
U = (10304, 1)
Trained data projected successfully

accuracy = 0.82
accuracy = 0.7875
accuracy = 0.7909090909090909
accuracy = 0.8071428571428572
accuracy = 0.8176470588235294
accuracy = 0.795
```



iv)

the accuracy of LDA in case of large number of non faces in training set 50 samples of faces and 200 samples of nonfaces:

```
 [1053.23125  1089.5125   1012.38125  ... 2889.0625   2833.96875
  2763.825   ]
 [1033.146375 1068.73575   993.075375 ... 2833.96875  2779.925625
  2711.1195  ]
 [1007.5749   1042.2834    968.4957   ... 2763.825    2711.1195
  2644.0164  ]]
eigen values 10304
eigen vectors (10304, 10304)
U = (10304, 1)
data projected successfully !

accuracy = 0.888
```

Non_faces accuracy:

```
eigen values 10304
eigen vectors (10304, 10304)
U = (10304, 1)
accuracy = 0.78
True Positive(TP)  =  153
False Positive(FP) =  41
True Negative(TN)  =  159
False Negative(FN) =  47
Accuracy of the binary classifier = 0.780
```

If you increased the number of non faces images in the training sample, the model will overfit or prefer to classify the data as non faces due to the large number of nonfaces and the small number faces data therefore this will cause that the faces can be classified as non faces too.

The accuracy of the LDA in case of large number of non faces in training set is higher than the nonfaces equal to faces.

Bonus Part:

a)splitting data into train and test with 70:30

the only change is in the original PCA and LDA we split data according to the index even or odd here we use [mod 10] to decide either the sample index is from 0->2 or from 3->9 to split it as 70% train and 30% test.

Accuracy comparison:

PCA: after new split

```
dimension of cenetered data (280, 10304) (10304, 40)
alpha = 0.8
accuracy = 0.975


dimension of cenetered data (280, 10304) (10304, 60)
alpha = 0.85
accuracy = 0.975


dimension of cenetered data (280, 10304) (10304, 92)
alpha = 0.9
accuracy = 0.975


dimension of cenetered data (280, 10304) (10304, 149)
alpha = 0.95
accuracy = 0.975
```

LDA:

```
 ...
 [-1845.81149075  1837.80317807  4876.90832878 ...    40.7995411
    127.30225437  -459.2691537 ]
 [-1477.34591236  1475.18036816  4887.1890861  ...   -40.84144367
    117.01174363  -337.39915844]
 [-1693.25074836  1704.51517822  4744.72691831 ...    17.63342215
    116.26325624  -395.61807524]]
accuracy = 0.9166666666666666
```

b)variations of PCA and LDA:

There are many variations of LDA and PCA implementation, one of them is in sklearn library (**from sklearn.discriminant_analysis import** LinearDiscriminantAnalysis, **from sklearn.decomposition import** PCA).We used it and calculated the accuracy, confusion matrix, and time complexity of each approach to compare between them.

```
Enter 4 to run the bonus part
4
To run different training and tst splits press 1
To run other variations of PCA & LDA press 2:
2
(200, 10304)
[[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 4 0]
 [0 0 0 ... 0 0 5]]
Accuracy : 0.955
LDA takes 0.658273 secs
[[3 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 3 0]
 [0 0 0 ... 0 0 3]]
Accuracy 0.915
PCA takes 0.283077 secs
```