Pattern Recognition
Lab 1 Report

Abdelrhman Ahmed Yousry Elnainay	19015889
Mahmoud Ibrahim Gad Abou Alwafa	19016532
Gamal Abdul Hameed Nasef Newysar	19015550

1. **Download the Dataset and Understand the Format (10Points)**

a.ORL dataset is available at the following link .https://www.kaggle.com/kasikrit/att-database-of-faces/
b.The dataset has 10 images per 40 subjects. Every image is grayscale image of size 92x112.

```
In [4]: x_features = []
        y_labels = []
        path  = "E:\year3_term2\pattern_recognition\Assigments\Assigment1"

        folder_path = path + '\pca_lda_dataset'

        def import_data(folder_path):
            for i in tqdm(os.listdir(folder_path)) :
                class_path = folder_path + '/' + i
                for j in os.listdir(class_path) :
                    img = plt.imread(os.path.join(class_path, j))
                    x_features.append(img)
                    y_labels.append(i)
            return x_features, y_labels


        x_features, y_labels = import_data(folder_path)

100%|████████████████████████████████████████████████████| 40/40 [00:05<00:00,  7.80it/s]
```

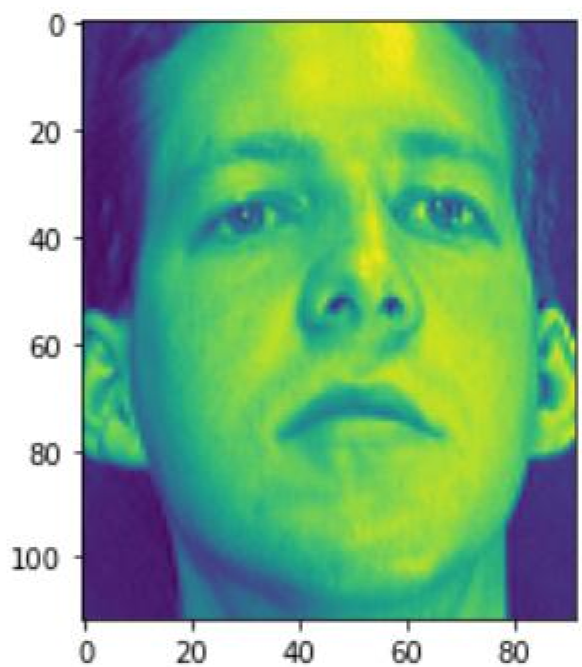
**The data consists of 40 classes each class contains 10 samples .
The data size is small and loaded in no time**

```
In [5]: x_features[0]

Out[5]: array([[48, 49, 45, ..., 56, 56, 54],
               [45, 52, 39, ..., 52, 50, 51],
               [45, 50, 42, ..., 48, 53, 50],
               ...,
               [50, 48, 50, ..., 45, 46, 46],
               [45, 54, 49, ..., 46, 47, 47],
               [51, 51, 51, ..., 47, 46, 46]], dtype=u

In [6]: plt.imshow(x_features[9])
        print(y_labels[9])

s1
```



Sample from the data

```
In [7]: x_features[0].shape
Out[7]: (112, 92)

In [8]: type(x_features[0])
Out[8]: numpy.ndarray
```

Each image has shape (112,92)
Each image is of type ndarray (n dimensional array)

2. Generate the Data Matrix and the Label vector (10 Points)

a.Convert every image into a vector of 10304 values corresponding to the image size.
b.Stack the 400 vectors into a single Data Matrix D and generate the label vector y

```
In [10]: def flatten_images(x_features):
             for i in range(len(x_features)) :
                 x_features[i] = x_features[i].reshape(-1) # we can use .ravel() also
             return x_features

         x_features = flatten_images(x_features)

In [11]: x_features[0].shape
Out[11]: (10304,)

In [12]: len(x_features)
Out[12]: 400
```

.The labels are integers from 1:40 corresponding to the subject id.

```
In [13]: def encode_labels(y_labels):
             le = LabelEncoder()
             y_label_coded = le.fit_transform(y_labels)
             return y_label_coded, le

         y_label_coded, le = encode_labels(y_labels)
         y_label_coded
```

We used label encoder to encode the data subjects

3. Split the Dataset into Training and Test sets (10 Points)

a.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.
b.Split the labels vector accordingly.

```
In [14]: def split_data_even_odd(x_features, y_label_coded):
             x_train = []
             y_train = []
             x_test = []
             y_test = []
             for i in range(len(x_features)) :
                 if i % 2 != 0 :
                     x_train.append(x_features[i])
                     y_train.append(y_label_coded[i])
                 else :
                     x_test.append(x_features[i])
                     y_test.append(y_label_coded[i])
             return x_train, x_test, y_train, y_test

         x_train, x_test, y_train, y_test = split_data_even_odd(x_features, y_label_coded)
```

4. Classification using PCA(30 points)

a.Use the pseudo code below for computing the projection matrix U. Define the alpha = {0.8,0.85,0.9,0.95}

```
In [18]: def get_eigens_and_meanVector(x_train): #must be numpy matrix
             #compute mean
             mean_vector = x_train.mean(axis = 0)
             #center_data
             x_train_cen = x_train - mean_vector
             #compute covariance
             cov = 1 / len(x_train_cen) * (np.dot(x_train_cen.T, x_train_cen))
             #compute eigen vals and eigen vec
             eig_val, eig_vec = np.linalg.eigh(cov)
             return eig_val, eig_vec, mean_vector
```

```
In [19]: def get_proj_mat(eig_val, eig_vec, alpha):
             #fraction of total variance
             fraction = []
             eig_sum=0
             eig_val_rev = eig_val[::-1]
             eig_vec_rev = eig_vec[::-1]

             for i in eig_val_rev :
                 eig_sum += i
                 fraction.append(eig_sum/eig_val_rev.sum())
             fraction = np.array(fraction)
             #choose dimensionality
             fraction_trimed = fraction > alpha
             smallest_ind = 0
             for i in range(len(fraction_trimed)) :
                 if fraction_trimed[i] == True :
                     smallest_ind = i
                     break;
             #reduced basis
             proj_mat = eig_vec_rev[:smallest_ind+1]
             return proj_mat
```

b.Project the training set, and test sets separately using the same projection matrix.

```
In [20]: def center_data(x_train, x_test, mean_vector):
             return x_train-mean_vector, x_test-mean_vector
```

```
In [21]: def get_projected_mat(x_train, x_test, proj_mat):
             x_train_proj = np.dot(x_train, proj_mat.T)
             #x_test_cen = x_test- x_train_proj.mean(axis=1)
             x_test_proj = np.dot(x_test, proj_mat.T)
             return x_train_proj, x_test_proj
```

c .Use a simple classifier (first Nearest Neighbor to determine the class labels).

```
In [22]: def classify_KNN(x_train_proj, y_train, x_test_proj, y_test, alpha, neighbors=1):
             knn = KNN(n_neighbors = neighbors)
             knn.fit(x_train_proj, y_train)
             y_pred = knn.predict(x_test_proj)
             print("accuracy of alpha",alpha,"and neighbours =", neighbors,"is", accuracy_score(y_pred, y_test))
             print("precision of alpha",alpha,"and neighbours =", neighbors,"is", precision_score(y_pred, y_test, average = 'weighted'))
             print("recall of alpha",alpha,"and neighbours =", neighbors,"is", recall_score(y_pred, y_test, average = 'macro'))
             print("f1 score of alpha",alpha,"and neighbours =", neighbors,"is", f1_score(y_pred, y_test, average = 'weighted'))
```

```
In [*]: x_train = np.matrix(x_train)
        x_test = np.matrix(x_test)
        y_train = np.array(y_train).reshape(200,1)
        y_test = np.array(y_test).reshape(200,1)
        eig_val, eig_vec, mean_vector = get_eigens_and_meanVector(x_train)
```

```
In [*]: alphas = [0.8, 0.85, 0.9, 0.95]
        for alpha in alphas:
            proj_mat = get_proj_mat(eig_val, eig_vec, alpha)
            x_train_cen, x_test_cen = center_data(x_train, x_test, mean_vector)
            x_train_proj, x_test_proj = get_projected_mat(x_train_cen, x_test_cen, proj_mat)
            classify_KNN(x_train_proj, y_train, x_test_proj, y_test, alpha, neighbors = 1)
            print("________________________________________________________")
```

d. Report Accuracy for every value of alpha separately.

```
accuracy of alpha 0.8 and neighbours = 1 is 0.87
precision of alpha 0.8 and neighbours = 1 is 0.907
recall of alpha 0.8 and neighbours = 1 is 0.8921428571428572
f1 score of alpha 0.8 and neighbours = 1 is 0.8764500777000777
________________________________________________________
accuracy of alpha 0.85 and neighbours = 1 is 0.915
precision of alpha 0.85 and neighbours = 1 is 0.9490000000000001
recall of alpha 0.85 and neighbours = 1 is 0.9386507936507936
f1 score of alpha 0.85 and neighbours = 1 is 0.9191901154401154
________________________________________________________
accuracy of alpha 0.9 and neighbours = 1 is 0.92
precision of alpha 0.9 and neighbours = 1 is 0.9620000000000001
recall of alpha 0.9 and neighbours = 1 is 0.9440476190476191
f1 score of alpha 0.9 and neighbours = 1 is 0.9256060606060607
________________________________________________________
accuracy of alpha 0.95 and neighbours = 1 is 0.945
precision of alpha 0.95 and neighbours = 1 is 0.966
recall of alpha 0.95 and neighbours = 1 is 0.9577380952380953
f1 score of alpha 0.95 and neighbours = 1 is 0.9481890331890331
```

e. Can you find a relation between alpha and classification accuracy?

From the results it's obvious that by increasing the alpha the accuracy of the model increases

This happens because the percentage of data loss from our data decrease so the accuracy of the model increases

i.e. when increasing the alpha the new dims (eigen vectors) become more and more representative for the data

5. Classification Using LDA (30 Points)

a. Use the pseudo code below for LDA. We will modify few lines in pseudocode to handle multiclass LDA.

i.Calculate the mean vector for every class Mu1,Mu2,..,Mu40.

Calculate the mean vector for every class Mu1,Mu2,...,Mu40.

In [29]:
```
meanVectors = []
eachClassIndices = []
for i in tqdm(range(len(uniqueClasses))):
    indices = np.where(y_train == uniqueClasses[i])
    array = np.asarray(x_train[indices[0], :])
    meanVectors.append(np.mean(array, axis=0))
    eachClassIndices.append( indices[0])

meanVectors = np.asarray(meanVectors)
print(meanVectors.shape)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 40/40 [00:00<00:00, 6681.49it/s]
(40, 10304)
```

Calculate the mean vector for every class Mu1, Mu2, ..., Mu40.

In [ ]:
```
overAllMean = np.mean(x_train, axis=0)
print(overAllMean)
print(overAllMean.shape)
```

```
[[84.97  84.79  85.055 ... 76.39  73.815 72.215]]
(1, 10304)
```

ii.Replace B matrix by Sb .$S_b=\sum n_k(\mu_k-\mu)(\mu_k-\mu)^T$

In [*]:
```
sb = np.zeros([10304,10304])
nk = len(eachClassIndices[0])
for i in range(len(uniqueClasses)):
    centered = np.asmatrix(meanVectors[i] - overAllMean)
    sb += (nk * np.matmul(centered.T , centered) )

print(sb)
```

iii.S matrix remains the same, but it sums S1,S2,S3,...S40.

In [*]:
```
sb = np.zeros([10304,10304])
nk = len(eachClassIndices[0])
for i in range(len(uniqueClasses)):
    centered = np.asmatrix(meanVectors[i] - overAllMean)
    sb += (nk * np.matmul(centered.T , centered) )

print(sb)
```

Calculating S matrix

In [32]:
```
s = np.zeros([10304,10304])
for j in tqdm(range(len(eachClassIndices))):
    D = np.asarray(x_train[eachClassIndices[j], :])
    z = np.asarray(D) - meanVectors[j]
    s += np.matmul(z.T , z)

print(s)
```

```
100%|████████████████████████████████████████████████████████████████| 40/40 [00:38<00:00,  1.04it/s]
```

Calculating S^{-1}

In [*]:
```
s = np.asmatrix(s)
sInv = np.linalg.inv(s)
```

Calculating eignValues and eignVectors $of\ S^{-1}B$

using eigh

In [*]:
```
eignValues, eignVectors = np.linalg.eigh(np.matmul(sInv, sb))
```

iv.Use 39 dominant eigenvectors instead of just one. You will have a projection matrix $U_{39x10304}$.

In [*]:
```
idx = np.asmatrix(eignValues.argsort())
idx = np.flip(idx)

eignValues = eignValues[idx]
eignVectors = eignVectors[idx,:].reshape(10304, 10304).T
```

In [*]:
```
projectionMatrix  = np.asmatrix(eignVectors[0:39,:])
print(projectionMatrix)
```

b.Project the training set, and test sets separately using the same projection matrix U. You will have 39 dimensions in the new space.

In [40]:
```
print(projectionMatrix.shape)
print(x_train.shape)
x_trainProj = np.matmul(projectionMatrix, x_train.T).T
x_testProj = np.matmul(projectionMatrix, x_test.T).T
print(x_trainProj.shape)
```

```
(39, 10304)
(200, 10304)
(200, 39)
```

c.Use a simple classifier (first Nearest Neighbor to determine the class labels).

d. Report accuracy for the multiclass LDA on the face recognition dataset.

In [41]:
```
neighbours = [1,3,5,7]
def LDATuning():
    for i in neighbours:
        knn = KNN(n_neighbors = i)
        knn.fit(np.asarray(x_trainProj), y_train.ravel())
        y_pred = knn.predict(np.asarray(x_testProj))
        print("accuracy for nerigbour",i,"is",accuracy_score(y_pred, y_test.ravel()))
```

In [42]:
```
LDATuning()
```

```
accuracy for nerigbour 1 is 0.93
accuracy for nerigbour 3 is 0.85
accuracy for nerigbour 5 is 0.775
accuracy for nerigbour 7 is 0.745
```

6. Classifier Tuning (20 Points)

Plot (or tabulate) the performance measure (accuracy) against the K value. This is to be done for PCA and LDA as well.

K	PCA	LDA
1	94.5	93
3	83.5	85
5	78	77.5
7	73.5	74.5

---

7. Compare vs Non-Face Images (15 Points)

Download non-face images and make them of the same size 92x112.

The data contains images of airplanes, cars, cats, dogs, fruits, flowers and motorbikes.

i. Show failure and success cases.

```
accuracy of neighbours = 1 is 0.96
precision of neighbours = 1 is 0.9596
recall of neighbours = 1 is 0.96
f1 score of neighbours = 1 is 0.9597083456498176
Success case:
Actual Classification: [0.]
Pred Classification: 0.0
Failure case:
Actual Classification: [0.]
Pred Classification: 1.0
---------------------------------------------------
accuracy of neighbours = 3 is 0.94
precision of neighbours = 3 is 0.9455999999999999
recall of neighbours = 3 is 0.94
f1 score of neighbours = 3 is 0.9417420940001584
Success case:
Actual Classification: [0.]
Pred Classification: 0.0
Failure case:
Actual Classification: [0.]
Pred Classification: 1.0
---------------------------------------------------
```

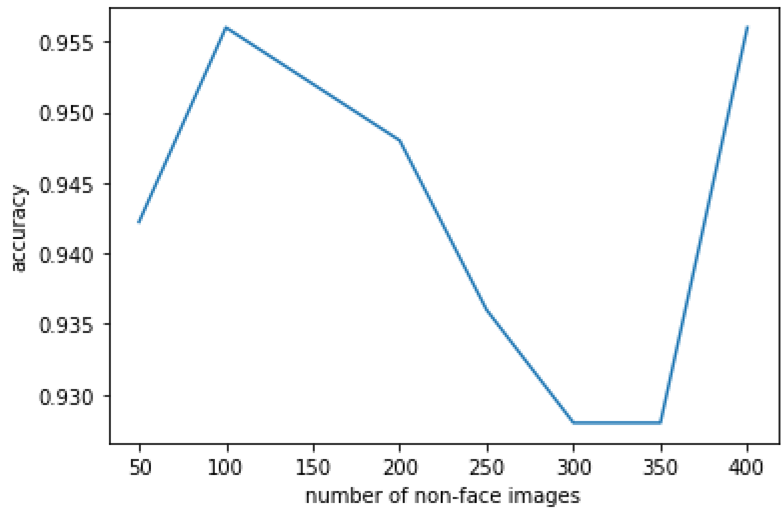
ii. How many dominant eigenvectors will you use for the LDA solution?

Number of dominant eigenvectors = 10, as it has the highest accuracy, recall, precision and F1 score between the other numbers of dominant eigenvectors.

```
accuracy of neighbours = 1 is 0.96
precision of neighbours = 1 is 0.9596
recall of neighbours = 1 is 0.96
f1 score of neighbours = 1 is 0.9597083456498176
Success case:
Actual Classification: [0.]
Pred Classification: 0.0
Failure case:
Actual Classification: [0.]
Pred Classification: 1.0
--------------------------------------------------
accuracy of neighbours = 3 is 0.94
precision of neighbours = 3 is 0.9455999999999999
recall of neighbours = 3 is 0.94
f1 score of neighbours = 3 is 0.9417420940001584
Success case:
Actual Classification: [0.]
Pred Classification: 0.0
Failure case:
Actual Classification: [0.]
Pred Classification: 1.0
--------------------------------------------------
```

```
-------------------------------------------------------
accuracy of neighbours = 5 is 0.912
precision of neighbours = 5 is 0.91776
recall of neighbours = 5 is 0.912
f1 score of neighbours = 5 is 0.9141582643328792
Success case:
Actual Classification: [0.]
Pred Classification: 0.0
Failure case:
Actual Classification: [0.]
Pred Classification: 1.0
-------------------------------------------------------
accuracy of neighbours = 7 is 0.9
precision of neighbours = 7 is 0.9181999999999999
recall of neighbours = 7 is 0.9
f1 score of neighbours = 7 is 0.9058974144888814
Success case:
Actual Classification: [0.]
Pred Classification: 0.0
Failure case:
Actual Classification: [0.]
Pred Classification: 1.0
-------------------------------------------------------
```

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



iv. Criticize the accuracy measure for large numbers of non-faces images in the training data.

As we saw from the plot, the accuracy decreases when we increase the size of the non-face images whlile fixing the number of face images.

---

8. Bonus

a. **[5 points] Use different Training and Test splits. 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.**

Algorithm	Accuracy
PCA (50%)	94.5
PCA (70%)	92.4
LDA (50%)	94.5
LDA (70%)	95.8

b. **[10 points] There are other variations of PCA and LDA beyond the original algorithms. Please use one of the variations of PCA and one variations of LDA other than the original ones. Compare the time complexity and accuracy between the 2 different PCA and LDA models.**

Algorithm	Accuracy	Time(seconds)
PCA	94.5	0.11 (without calculating eign vectors and values) 136.83 (total time including all steps)
LDA	93	0.01 (without calculating eign vectors and values) 363.1 (total time including all steps)
sklearn.PCA	97.5	0.24
sklearn.LDA	95	0.45