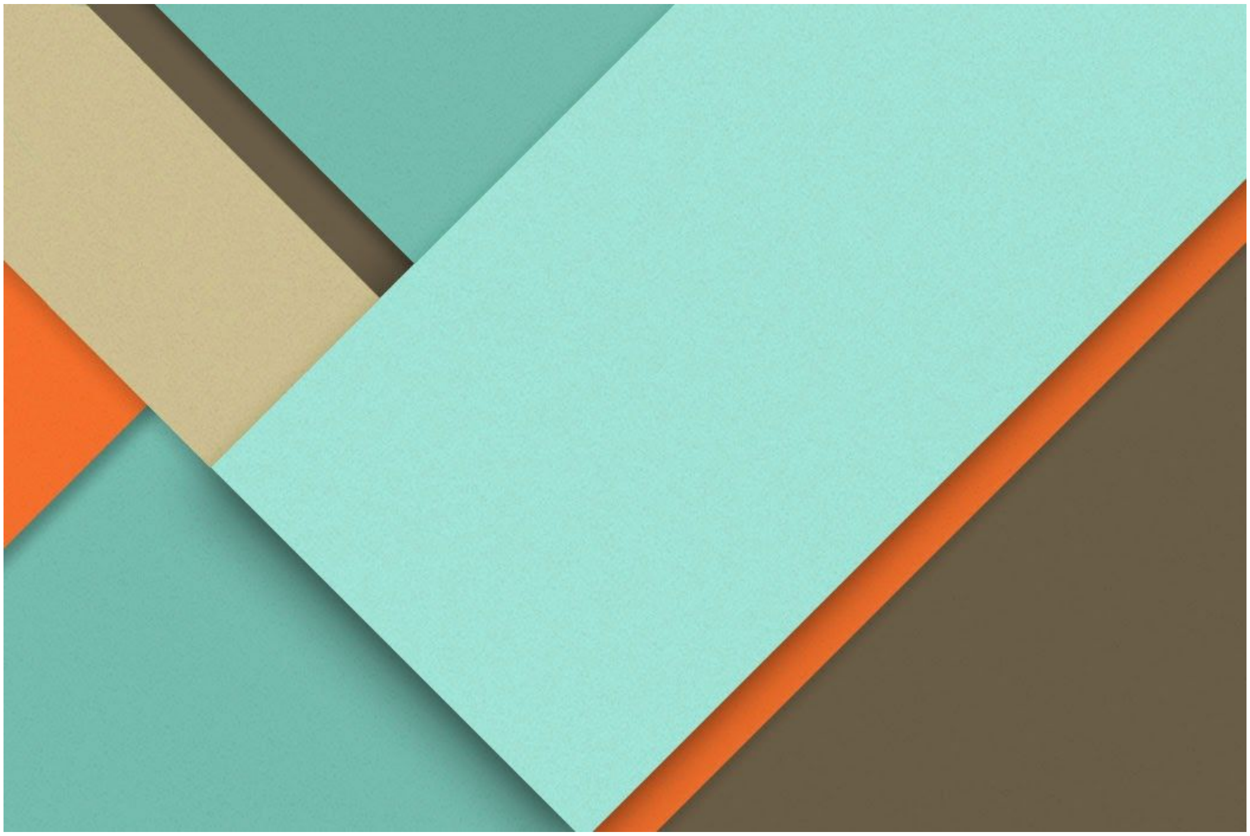


Assignment 1:Face Recognition

—

Members Names:
Toka Alaa Ahmed (14)
Nada Salama Mohammed (58)

2. Generate the Data Matrix

```
def read_pgm(filename):
    with open(filename, 'rb') as f:
        buffer = f.read()
    try:
        header, width, height, maxval = re.search(
            b"(^P5\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n]\s)*)", buffer).groups()
    except AttributeError:
        raise ValueError("Not a raw PGM file: '%s'" % filename)

    return numpy.frombuffer(buffer,
                            dtype='u1',
                            count=int(width)*int(height),
                            offset=len(header)
                            ).flatten()
Data_Matrix = []
label = []
for j in range(40):
  for i in range(10):
    image = read_pgm("/content/drive/My Drive/Face_Recognition/orl_dataset/s"+ str(j+1)+ "/" + str(i+1) + ".pgm")
    label.append(j+1)
    Data_Matrix.append(image)
```

- Data_Matrix.shape : (400, 10304)
- Label.shape : (400,)

3. Split the Dataset into Training and Test sets

```
def split_data(Data_Matrix, label,way = 1):
    if way == 1:
      Train_Data, Test_Data = Data_Matrix[::2], Data_Matrix[1::2]
      Train_Label, Test_Label = label[::2], label[1::2]
    else:
      Train_Data, Test_Data, Train_Label, Test_Label = train_test_split(Data_Matrix, label,
                                                        train_size=0.7)
    return Train_Data, Test_Data, Train_Label, Test_Label
```

```
Train_Data, Test_Data, Train_Label, Test_Label = split_data(Data_Matrix, label,way = 1)

print(Train_Data.shape)
print(Train_Label.shape)
```

```
(200, 10304)
```

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`

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

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

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

1- calculate mean of data by (mean = numpy.mean(Train_Data, axis=0))

2- then our data by subtract mean from our data

3- then calculate covariance matrix, eigenvalues, and eigenvectors

```
mean = numpy.mean(Train_Data, axis=0)
Z = Train_Data - mean
COV = (np.dot(Z.T,Z)) / Z.shape[0]
eigvals, eigvecs = np.linalg.eigh(COV)
eigvals = np.diag(eigvals)
```

4- then to calculate smallest r :

- I calculate explained_variances then reversed it and also reversed eigenvectors
- Take index number when sum of explained_variances of this index and before it greater than alpha
- Then take from eigenvectors from start to this index to be reduced basis

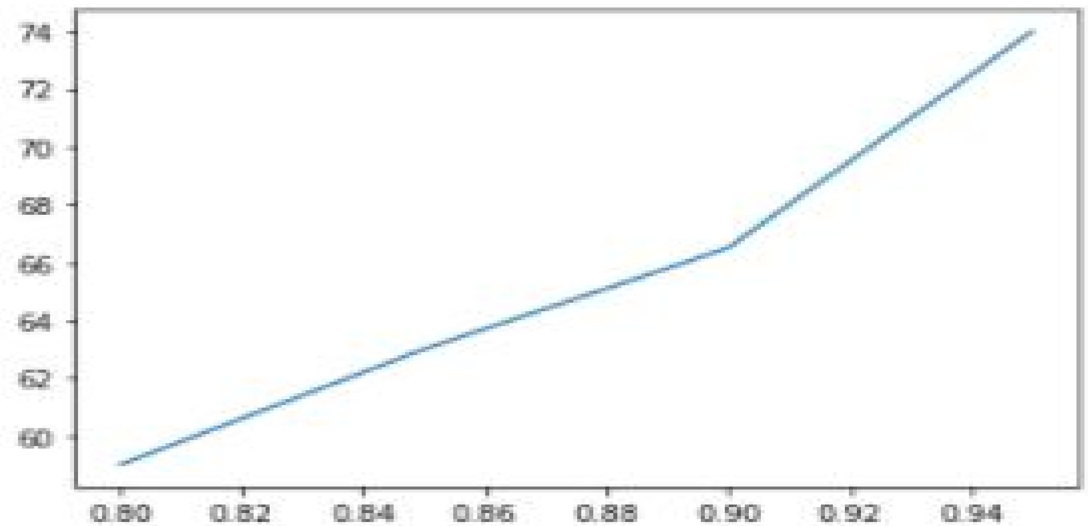
5- then calculate reduced dimensionality of train and test set by using same projection matrix (reduced basis)

d. Accuracy for every value of alpha separately

alpha	0.8	0.85	0.9	0.95
Accuracy	59%	63%	66.5%	74%

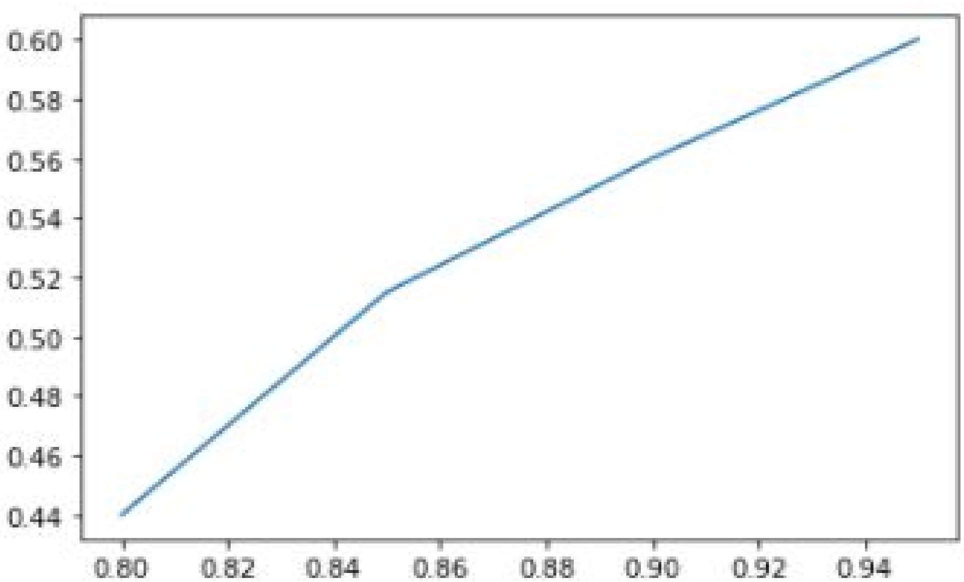
e- relation between alpha and the classification accuracy

At K = 1 (a simple classifier (first Nearest Neighbor to determine the class labels))

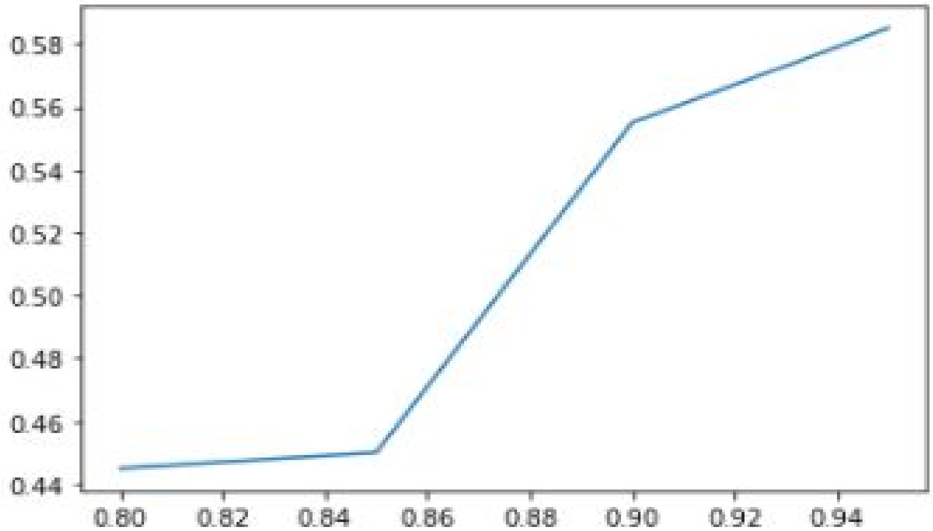


- When alpha increases,also accuracy increases

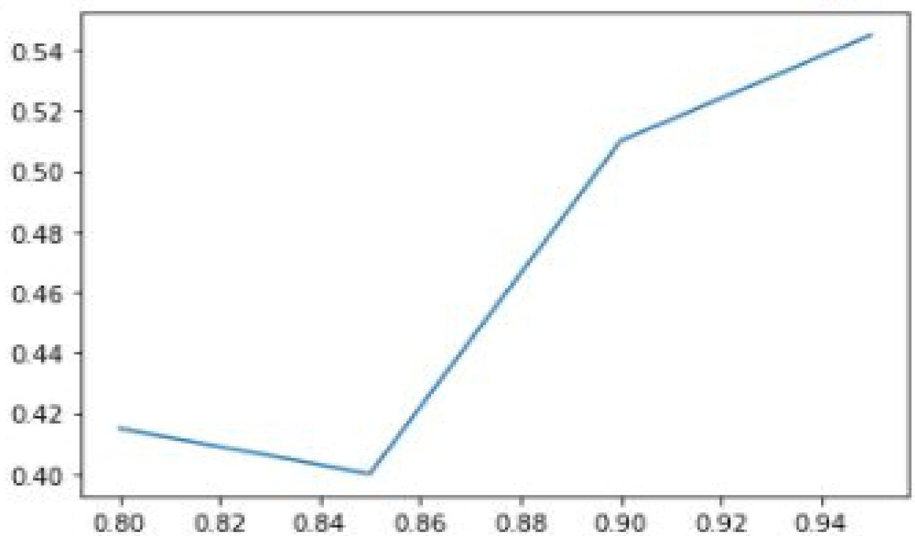
At K = 3



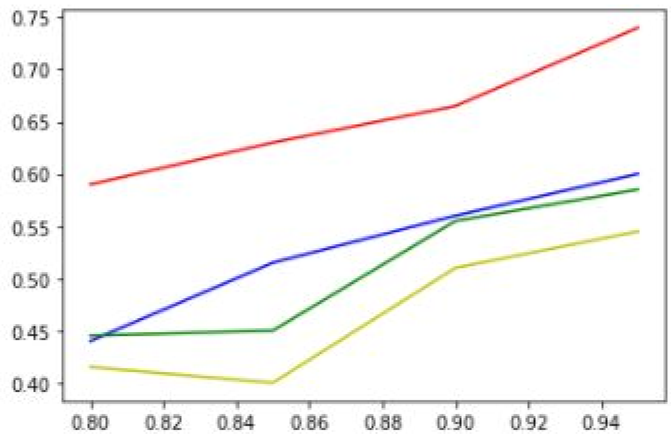
At K = 5


0.58
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.80
0.82
0.84
0.86
0.88
0.90
0.92
0.94

At K = 5


0.54
0.52
0.50
0.48
0.46
0.44
0.42
0.40
0.80
0.82
0.84
0.86
0.88
0.90
0.92
0.94

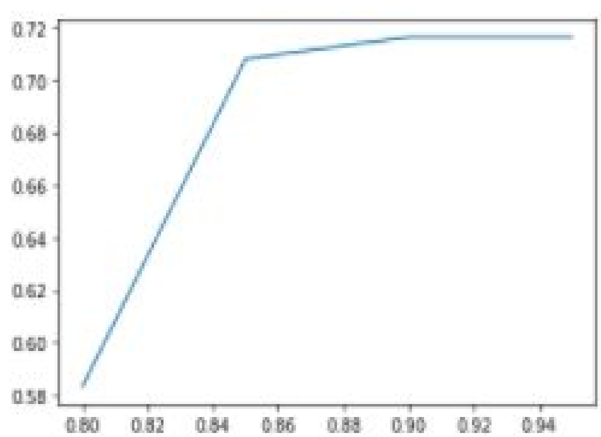
6. Plot accuracy against the K value

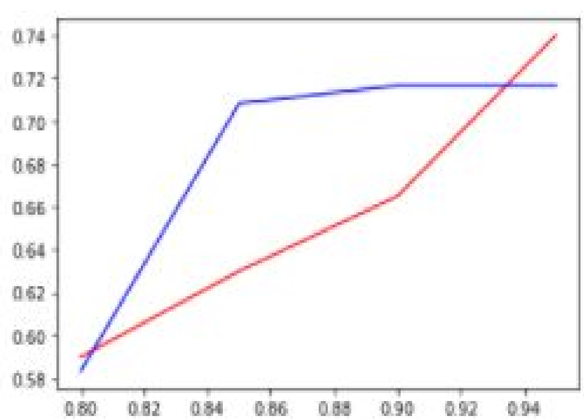


K / alpha	0.8	0.85	0.9	0.95
1	0.59	0.63	0.665	0.74
3	0.44	0.515	0.56	0.6
5	0.445	0.45	0.555	0.585
7	0.415	0.4	0.51	0.545

Bouns : For split with 70%

```
([0.8,0.85,0.9,0.95],[0.59,0.63,0.665,0.74],'r')#for50%
[0.8,0.85,0.9,0.95],[0.583,0.708,0.717,0.717],'b')#for70%
```





LDA

Implementation of LDA algorithm:

1- reshape training data to (num of classes, Samples per class, num of features)

2- compute mean of all samples per class (shape = num of classes, num of features)

3- compute the overall mean (shape = num of features)

4- compute Sb:

```
Sb = np.zeros((n,n))

for i in range(Nclasses):

      Sb+= NClassSamples * (mean[i] - overall_mean) @ (mean[i] -
overall_mean).T
```

5- Z = training data – mean

6- compute S:

```
S = np.zeros((n,n))

 for i in range(Nclasses):

   S+= Z[i].T@Z[i]
```

7- compute eigenvalues and eigenvectors for $S^{-1}Sb$

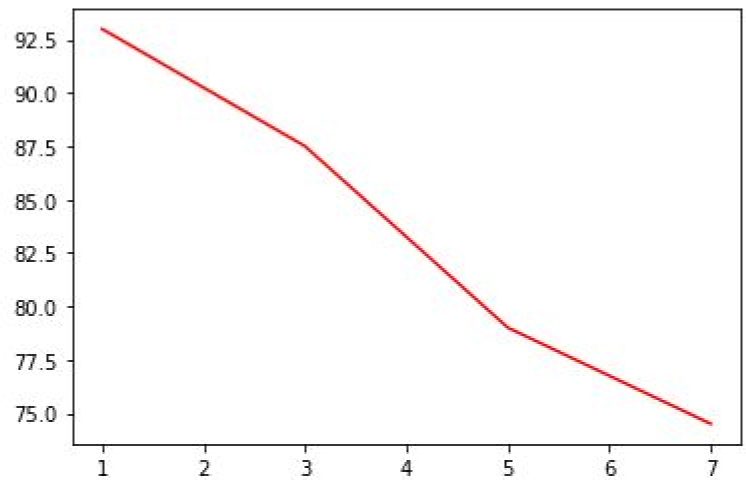
8- U = the 39 dominant eigenvectors (shape = 39 X 10304)

9- Project the training set and test sets separately using same projection matrix U

Prediction:

the K-NN classifier was used to determine the class labels for K = 1,3,5,7

(snapshot for accuracy of 50% data splitting)



(snapshot for accuracy of 70% data splitting)