Face Recognition

12 nd November 2020

Name	ID
Arsany Atef Abdo	**8**
Yomna Gamal El-Din Mahmoud	**63**

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 subjects is very simple. It has 40 subjects. Below we will show the needed steps to achieve the goal of the assignment.

Colab notebooks

- Notebook for PCA 50/50 split and LDA 50/50 split and 70/30 split.

  https://colab.research.google.com/drive/1UngNQg7VDqGpys_LHpMXaGCw7GRzRLlj?usp=sharing

- Notebook for PCA 70/30 split

  https://colab.research.google.com/drive/11unvvde9a_9zzWgrNzlwJ9D5VWNsFtT2?usp=sharing

Generate the Data Matrix

We define a function list_files which take directory and return list with all files in that directory. Each file is an image reshaped to 10304 and they are 400 image which mean that list is 400 x 10304

```
def list_files(dir):
    r = []
    subdirs = [x[0] for x in os.walk(dir)]
    for subdir in subdirs:
        files = os.walk(subdir).__next__()[2]
        if (len(files) > 0):
            for file in files:
                r.append(np.reshape(np.resize(cv2.imread(os.path.join(subdir, file)),(92,112)),(10304)))
    return r
```

```
#data matrix
x = list_files(filname)
X_array = np.asarray(x)
print(X_array.shape)
```

```
(400, 10304)
```

Split the Dataset into Training and Test sets

In this part we split data 2 times first as 50-50 of size 200 x 10304 for both and second as 70-30 of size 280 x 10304 & 120 x 10304 **“Bonus”** . And create the label vectors.

```
]  # split the data
   train_set = []
   test_set = []
   train_label = []
   test_label = []
   train7_set = []
   test3_set = []
   train7_label = []
   test3_label = []
   for i in range(400):
     if i % 2 == 0:
       test_set.append(X_array[i])
       test_label.append(np.floor(i/10) + 1.)
     else:
       train_set.append(X_array[i])
       train_label.append(np.floor(i/10) + 1.)
     # another split ratio of data
     if i % 10 in range(0,3):
       test3_set.append(X_array[i])
       test3_label.append(np.floor(i/10) + 1.)
     else:
       train7_set.append(X_array[i])
       train7_label.append(np.floor(i/10) + 1.)
```

Classification

PCA

- In this part we implement the PCA algorithm by defining a new function that takes the training matrix as an input with the alpha which is needed for truncation of the projection matrix.
- Returns the projection matrix used to project training and test sets.

```
# PCA algorithm
def PCA(D, alpha):
  # Compute the mean vector (10304x1)
  mu = np.mean(D, axis = 0).reshape(10304,1)
  print("\n the mean is:\n", mu)

  # center the data
  z = D - mu.T
  print("\n the centered data is:\n",z)

  # compute COV matrix
  COV = np.dot(z.T, z) / D.shape[0]
  print("\n the covariance matrix is:\n",COV)

  # find eigenvalues and eigenvectors
  values, vectors = np.linalg.eigh(COV)
  sorted_eig  = np.argsort(-values)
  values = values[sorted_eig]
  # values = np.diag(values)
  vectors = vectors[:, sorted_eig]
  print("\n the eigenvalues are:\n", values)
  print("\n the eigenvectors are:\n", vectors)

  # fraction of total variance
  tot = np.sum(values)
  var_exp = [(i / tot) for i in (values)]
  # var_exp = np.diag(var_exp)
  print("\n the fraction of total variances:\n", var_exp)
```

```
  # calculate the cumulative sum
  fr = np.cumsum(var_exp)
  print("\n the cumulative sum:\n",fr)

  # choose dimentionality
  index = 0
  for i in range(len(fr)):
    if fr[i] >= alpha:
      # smallest r so that f(r) >= alpha
      item_value = fr[i]
      index = i
      break
  print("\n item is:\n",item_value)
  print("\n we take till index: \n", index)

  # Compute projection matrix U
  U = vectors[:, :index]
  print("\n the projection matrix is:\n",U)

  return U
```

- Get the projection matrix from the training matrix from each alpha.

```
U1 = PCA(train_set_array, 0.8)
```

```
U2 = PCA(train_set_array, 0.85)
```

```
U3 = PCA(train_set_array, 0.9)
```

```
U4 = PCA(train_set_array, 0.95)
```

- Project the training and test sets for each alpha.

```
# Project the training set and test sets
P1_test = np.dot(test_set_array, U1)
print("\n the projection of test is:\n", P1_test)
print(P1_test.shape)

P1_train = np.dot(train_set_array, U1)
print("\n the projection of train is:\n", P1_train
print(P1_train.shape)
```

```
P2_test = np.dot(test_set_array, U2)
print("\n the projection of test is:\n", P2_test)
print(P2_test.shape)

P2_train = np.dot(train_set_array, U2)
print("\n the projection of train is:\n", P2_train)
print(P2_train.shape)
```

```
P3_test = np.dot(test_set_array, U3)
print("\n the projection of test is:\n", P3_test)
print(P3_test.shape)

P3_train = np.dot(train_set_array, U3)
print("\n the projection of train is:\n", P3_train)
print(P3_train.shape)
```

```
P4_test = np.dot(test_set_array, U4)
print("\n the projection of test is:\n", P4_test)
print(P4_test.shape)

P4_train = np.dot(train_set_array, U4)
print("\n the projection of train is:\n", P4_train)
print(P4_train.shape)
```

LDA

We define a new function called LDA which takes a training matrix and test matrix as an input and returns the new matrices after projection.

```
[8]  def LDA(D,Dtest):
       X = []
       M = []
       for i in range(40):
           subarray = []
           for j in range(5):
               subarray.append(D[i*5+j])
           M.append(np.mean(subarray,axis = 0))
           X.append(subarray)
       X = np.asarray(X)  # (40, 5, 10304)
       M = np.mean(X,axis = 1)  # (40, 10304)
       overall_M = np.mean(M,axis = 0)  # 10304
       S = np.zeros([10304,10304])
       S_b = np.zeros([10304,10304])
       for i in range (40):
         Zi = X[i]-M[i]
         S_b += 5*np.dot((M-overall_M).T,(M-overall_M))
         S += np.dot(Zi.T,Zi)
       # print(S_b.shape)
       product = np.matmul(np.linalg.pinv(S),S_b) # S inv
       vals,vecs = np.linalg.eigh(product)
       idx = vals.argsort()[::-1]
       vals = vals[idx]
       vecs = vecs[:,idx]
       Train_LDA = np.dot(D , vecs)
       Test_LDA = np.dot(Dtest , vecs)
       return Train_LDA,Test_LDA
```

Classifier Tuning

PCA

For 50/50 splitting

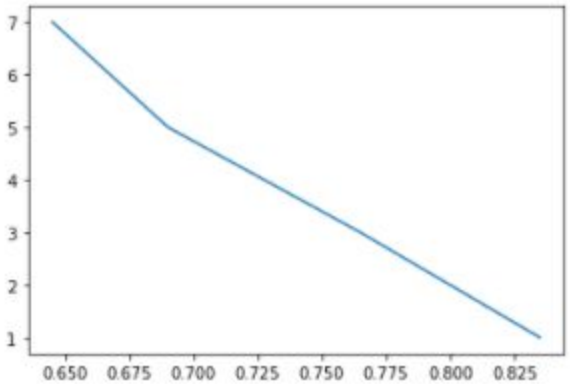
```
# Use a simple classifier (first Nearest Neighbor to determine the class labels).
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

```
# @ alpha = 0.8
accuracy_alpha1 = []
for i in [1,3,5,7]:
  classifier = KNeighborsClassifier(n_neighbors=i)
  classifier.fit(P1_train, train_label_array)
  y_pred = classifier.predict(P1_test)

  print("\n classifier:\n", classifier)
  print("\n prediction:\n", y_pred)
  print("\n prediction shape:\n", y_pred.shape)

  # Report Accuracy for every value of alpha separately
  accuracy_alpha1.append(accuracy_score(test_label_array, y_pred))

print("\n accuracy for alpha = 0.8 is:\n",accuracy_alpha1)
  # accuracy_score(y_pred, test_label, normalize=False)

print("\n accuracy for PCA when 50/50 split graph is:\n")
plt.plot(accuracy_alpha1,[1,3,5,7])
plt.show()
```

This gives us that result for 50-50 split data for neighbours of [1, 3, 5, 7].

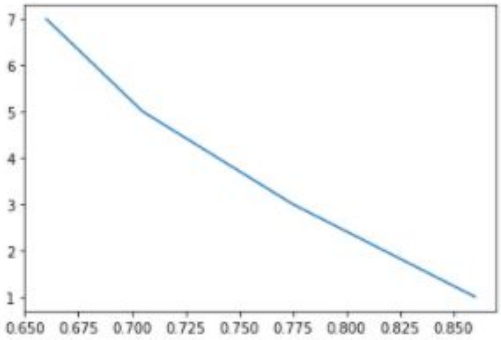
@ alpha = 0.8

```
accuracy for alpha = 0.8 is:
[0.835, 0.765, 0.69, 0.645]

accuracy for PCA when 50/50 split graph is:
```



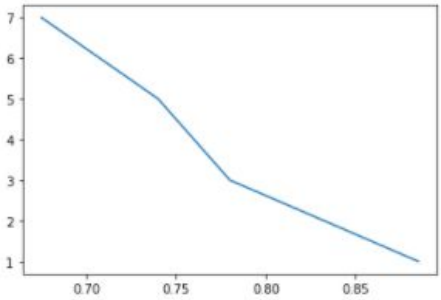
@ alpha = 0.85

```
accuracy for alpha = 0.85 is:
[0.86, 0.775, 0.705, 0.66]

accuracy for PCA when 50/50 split graph is:
```



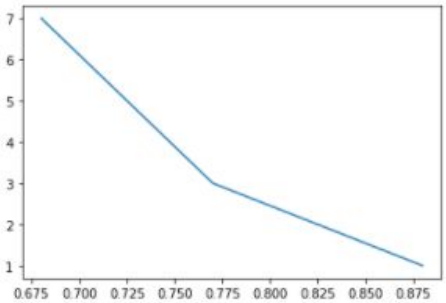
@ alpha = 0.9

```
accuracy for alpha = 0.9 is:
[0.885, 0.78, 0.74, 0.675]

accuracy for PCA when 50/50 split graph is:
```



@ alpha = 0.95

```
accuracy for alpha = 0.95 is:
[0.88, 0.77, 0.725, 0.68]

accuracy for PCA when 50/50 split graph is:
```

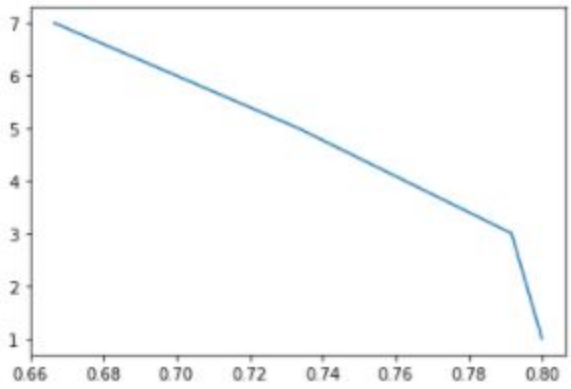


For 70/30 splitting

This gives us that result for 70-30 split data for neighbours of [1, 3, 5, 7].

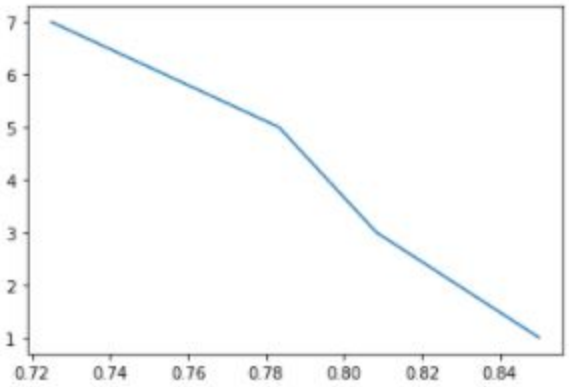
@ alpha = 0.8

```
accuracy for alpha = 0.8 is:
[0.8, 0.7916666666666666, 0.7333333333333333, 0.6666666666666666]

accuracy for PCA when 70/30 split graph is:
```



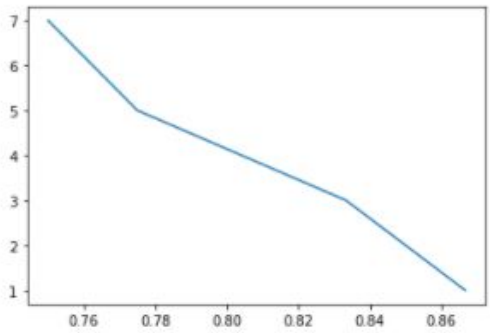
@ alpha = 0.85

```
accuracy for alpha = 0.85 is:
[0.85, 0.8083333333333333, 0.7833333333333333, 0.725]

accuracy for PCA when 70/30 split graph is:
```



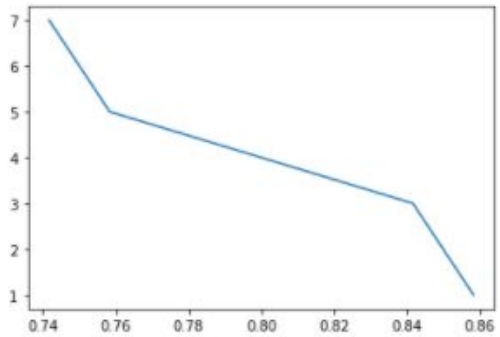
@ alpha = 0.9

```
accuracy for alpha = 0.9 is:
[0.8666666666666667, 0.8333333333333334, 0.775, 0.75]

accuracy for PCA when 70/30 split graph is:
```



@ alpha = 0.95

```
accuracy for alpha = 0.95 is:
[0.8583333333333333, 0.8416666666666667, 0.7583333333333333, 0.7416666666666667]

accuracy for PCA when 70/30 split graph is:
```



Relation between alpha and classification accuracy

When alpha increases with decreasing the neighbours the accuracy decreases.

@ 50/50 split

knn	*PCA (alpha = 0.8)*	*PCA (alpha = 0.85)*	*PCA (alpha = 0.9)*	*PCA (alpha = 0.95)*
1	*0.835*	*0.86*	*0.885*	*0.88*
3	*0.765*	*0.775*	*0.78*	*0.77*
5	*0.69*	*0.705*	*0.74*	*0.725*
7	*0.645*	*0.66*	*0.675*	*0.68*

@ 70/30 split

knn	*PCA (alpha = 0.8)*	*PCA (alpha = 0.85)*	*PCA (alpha = 0.9)*	*PCA (alpha = 0.95)*
1	*0.8*	*0.85*	*0.8666666667*	*0.8583333333*
3	*0.79166666*	*0.8083333333*	*0.83333334*	*0.84166666667*
5	*0.73333333*	*0.7833333333*	*0.775*	*0.7583333333*
7	*0.6666666*	*0.725*	*0.75*	*0.74166666667*

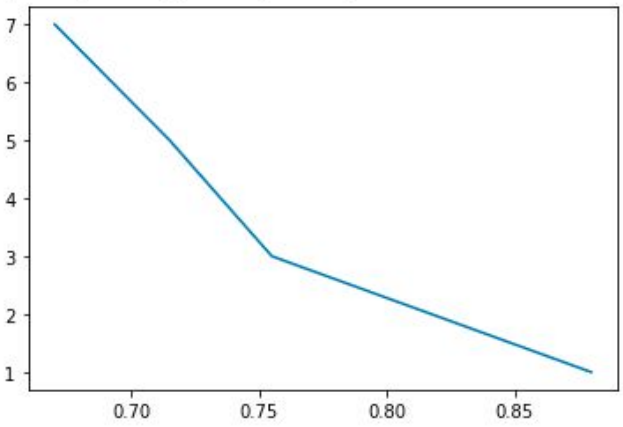
LDA

```
[9]  Train_LDA,Test_LDA = LDA(train_set_array,test_set_array)
```

```
[12] from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score
     from matplotlib import pyplot as plt
     accuracy_LDA = []
     for i in [1,3,5,7]:
       classifier = KNeighborsClassifier(n_neighbors=i)
       classifier.fit(Train_LDA, train_label_array)
       y_pred = classifier.predict(Test_LDA)

       print("\n classifier:\n", classifier)
       print("\n prediction:\n", y_pred)
       print("\n prediction shape:\n", y_pred.shape)

       # Report Accuracy for every value of alpha separately
       accuracy_LDA.append(accuracy_score(test_label_array, y_pred))

     print("\n accuracy for LDA when 50/50 split is:\n",accuracy_LDA)
     plt.plot(accuracy_LDA,[1,3,5,7])
     plt.show()
```

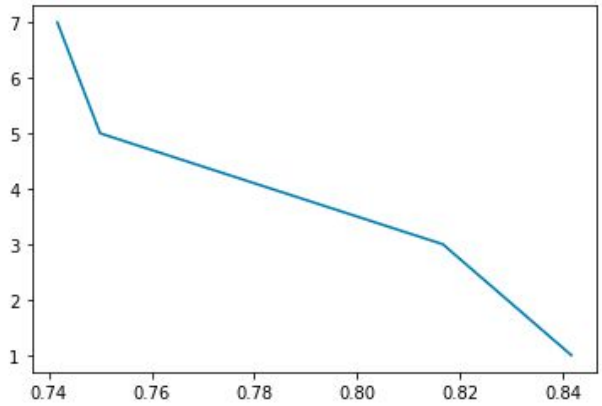
This gives us that result for 50-50 split data for neighbours of [1, 3, 5, 7].

```
accuracy for LDA when 50/50 split is:
[0.88, 0.755, 0.715, 0.67]
```



This gives us that result for 70-30 split data for neighbours of [1, 3, 5, 7].

```
[0.8416666666666667, 0.8166666666666667, 0.75, 0.7416666666666667]
```



Comparison of accuracy results between PCA and LDA

@ 50/50 split

The result from the LDA is nearest to the result of the PCA when alpha is 0.95.

The result from the LDA is almost better than PCA when alpha is smaller than or equal 0.85.

The result from the PCA is almost better than LDA when alpha is greater than or equal 0.9.

knn	LDA	PCA (alpha = 0.8)	PCA (alpha = 0.85)	PCA (alpha = 0.9)	PCA (alpha = 0.95)
1	0.88	0.835	0.86	0.885	0.88
3	0.755	0.765	0.775	0.78	0.77
5	0.715	0.69	0.705	0.74	0.725
7	0.67	0.645	0.66	0.675	0.68

@ 70/30 split

The result from the LDA is nearest to the result of the PCA when alpha is 0.95.

The result from the LDA is almost better than PCA when alpha is smaller than or equal 0.85.

The result from the PCA is almost better than LDA when alpha is greater than or equal 0.9.

knn	LDA	PCA (alpha = 0.8)	PCA (alpha = 0.85)	PCA (alpha = 0.9)	PCA (alpha = 0.95)
1	0.841666667	0.8	0.85	0.8666666667	0.8583333333
3	0.81666667	0.79166666	0.8083333333	0.83333334	0.84166666667
5	0.75	0.73333333	0.7833333333	0.775	0.7583333333
7	0.741666667	0.6666666	0.725	0.75	0.74166666667