question5

April 12, 2023

[23]:

```
# Question 5
# Anirudh Sathish
# CS20B1125
```

0.0.1 Fisherfaces- Face classification using LDA (40 classes)

- a) Use the following "face.csv" file to classify the faces of 40 different people using LDA.
- b) Do not use the in-built function for implementing LDA.
- c) Use appropriate classifier taught in class (any classification algorithm taught in class like Bayes classifier, minimum distance classifier, and so on)

[24]:

```
# libs
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.linalg import eig
```

[25]:

```
# get data
faces = pd.read_csv("face.csv")
faces.head()
```

[25]:

```
          0         1         2         3         4         5         6  \
0  0.309917  0.367769  0.417355  0.442149  0.528926  0.607438  0.657025
1  0.454545  0.471074  0.512397  0.557851  0.595041  0.640496  0.681818
2  0.318182  0.400826  0.491736  0.528926  0.586777  0.657025  0.681818
3  0.198347  0.194215  0.194215  0.194215  0.190083  0.190083  0.243802
4  0.500000  0.545455  0.582645  0.623967  0.648760  0.690083  0.694215

          7         8         9  …      4087      4088      4089      4090  \
0  0.677686  0.690083  0.685950  …  0.669422  0.652893  0.661157  0.475207
1  0.702479  0.710744  0.702479  …  0.157025  0.136364  0.148760  0.152893
2  0.685950  0.702479  0.698347  …  0.132231  0.181818  0.136364  0.128099
3  0.404959  0.483471  0.516529  …  0.636364  0.657025  0.685950  0.727273
4  0.714876  0.723140  0.731405  …  0.161157  0.177686  0.173554  0.177686

       4091      4092      4093      4094      4095  target
0  0.132231  0.148760  0.152893  0.161157  0.157025       0
```

```
1  0.152893  0.152893  0.152893  0.152893  0.152893       0
2  0.148760  0.144628  0.140496  0.148760  0.152893       0
3  0.743802  0.764463  0.752066  0.752066  0.739669       0
4  0.177686  0.177686  0.177686  0.173554  0.173554       0

[5 rows x 4097 columns]
```

[26]:

```
def LDA(data, target):
    # do LDA for n classes step by step
    #calculate mean for each class
    mean = []
    for i in range(len(np.unique(target))):
    # get name of class
        name = np.unique(target)[i]
        # calculate mean
        mean.append(np.mean(data[target == name], axis=0))
    mean = np.array(mean)
    # print mean
    print("Mean: ", mean)
    # compute scatter matrices
    # calculate mean of all data
    mean_all = np.mean(data, axis=0)
    # calculate within class scatter matrix
    Sw = np.zeros((data.shape[1], data.shape[1]))
    for i in range(len(np.unique(target))):
        # get name of class
        name = np.unique(target)[i]
        # calculate scatter matrix
        Sw += np.dot((data[target == name] - mean[i]).T, (data[target == name]␣
 ↪- mean[i]))

    # calculate between class scatter matrix
    Sb = np.zeros((data.shape[1], data.shape[1]))
    for i in range(len(np.unique(target))):
        # get name of class
        name = np.unique(target)[i]
        # calculate scatter matrix
        Sb += np.dot((mean[i] - mean_all).values.reshape(data.shape[1], 1),␣
 ↪(mean[i] - mean_all).values.reshape(1, data.shape[1]))

    # calculate eigen values and eigen vectors
    eigenValues, eigenVectors = np.linalg.eig(np.dot(np.linalg.inv(Sw), Sb))
    # sort eigen values and eigen vectors
    idx = eigenValues.argsort()[::-1]
    eigenValues = eigenValues[idx]
    eigenVectors = eigenVectors[:, idx]
    # reduce dimensions to 1
    d = 1
    # print d
    print("d: ", d)
    # reduce dimensions
    eigenVectors = eigenVectors[:, :d]
    # reduce data
    data = np.dot(data, eigenVectors)
    # convert to dataframe
    data = pd.DataFrame(data)
    return data
```

[27]:

```
# seperating the target and the features
y = faces["target"]
X = faces.iloc[:,:-1]
```

[28]:

```
X_ = LDA(X,y)
```

```
Mean:  [[0.34132231 0.37561984 0.41694215 … 0.27520662 0.27768596 0.27685951]
 [0.62396695 0.64586778 0.67768596 … 0.14793388 0.12561983 0.1161157 ]
 [0.37892563 0.39504133 0.41900827 … 0.29297521 0.27479339 0.26818182]
 …
 [0.14214876 0.18677686 0.26115703 … 0.31735538 0.28719008 0.32809918]
 [0.26528925 0.26322314 0.27107438 … 0.2570248  0.26735538 0.27066116]
 [0.41404959 0.43884298 0.44049587 … 0.3376033  0.33842975 0.35413223]]
d:  1
```

[29]:

```
X_reduced = pd.DataFrame(np.real(X_))
```

[30]:

```
# let's split into test and train
from sklearn.model_selection import train_test_split

# random state for reproducability
X_train, X_test , y_train , y_test = train_test_split(X_reduced,y,test_size=0.
 ↪3,random_state= 2)
```

[31]:

```
y_train = pd.DataFrame(y_train)
y_train
```

[31]:

```
     target
112      11
209      20
294      29
307      30
345      34
..      …
299      29
```

```
22         2
72         7
15         1
168       16

[280 rows x 1 columns]
```

[32]:
```
X_recomplete = pd.concat([X_train,y_train],axis = 1)
X_recomplete
```

[32]:
```
            0  target
112  0.030476      11
209  0.014997      20
294  0.036868      29
307  0.053403      30
345  0.038659      34
..        …     …
299  0.036868      29
22   0.015772       2
72   0.065397       7
15   0.034093       1
168  0.005988      16

[280 rows x 2 columns]
```

[33]:
```
meanData = X_recomplete.groupby("target").mean()
meanData
```

[33]:
```
               0
target
0       0.053665
1       0.034093
2       0.015772
3       0.043756
4       0.018576
5       0.015766
6      -0.001409
7       0.065397
8       0.033379
9       0.050969
10      0.028621
11      0.030476
12      0.015316
13      0.044778
14      0.025548
15      0.075380
16      0.005988
```

```
17      0.042496
18     -0.002886
19      0.026158
20      0.014997
21      0.043541
22      0.006569
23      0.039317
24      0.022078
25      0.034497
26      0.046660
27      0.036539
28      0.029458
29      0.036868
30      0.053403
31      0.033841
32      0.004278
33      0.040955
34      0.038659
35      0.023149
36      0.033112
37      0.055043
38     -0.000318
39      0.043456
```

[34]:

```python
def dist(X,mean):
    Y = np.sqrt(np.sum((X - mean)**2))
    return Y
```

[35]:

```python
predictions = []
for i in range(X_test.shape[0]):
    min = 10000000
    minIndex = -1
    for w in range(len(meanData)):
        dist_t = dist(X_test.iloc[i:i+1].to_numpy(),meanData.xs(w).to_numpy())
        if dist_t < min:
            min = dist_t
            minIndex = w
    predictions.append(minIndex)
```

[36]:

```python
y_test = y_test.to_frame()
```

[37]:

```python
# checking for accuracy
match = 0
for i in range(len(y_test)):
    expected = y_test["target"].iloc[i]
    prediction = predictions[i]
    if(prediction == expected):
        match+=1

accuracy = (match/len(y_test))*100

print("Accuracy of Classifier :" ,accuracy)
```

```
Accuracy of Classifier : 100.0
```