question-123

April 12, 2023

[113]:
```
# Anirudh Sathish
# CS20B1125
# Question 1, 2 , 3
```

0.0.1 Question 1

Consider the 128- dimensional feature vectors (d=128) given in the "gender.csv" file. (2 classes, male and female)

- a) Use PCA to reduce the dimension from d to d . (Here d=128)
- b) Display the eigenvalue based on increasing order, select the d of the corresponding eigenvector which is the appropriate dimension d (select d S.T first 95% of values of the covariance matrix are considered).
- c) Use d features to classify the test cases (use any classification algorithm taught in class like Bayes classifier, minimum distance classifier, and so on) #### Dataset Specifications:

- Total number of samples = 800
- Number of classes = 2 (labeled as "male" and "female")
- Samples from "1 to 400" belongs to class "male"
- Samples from "401 to 800" belongs to class "female"
- Number of samples per class = 400
- Number of dimensions = 128
- Use the following information to design classifier:
- Number of test cases (first 10 in each class) = 20
- Number of training feature vectors (remaining 390 in each class) = 390
- Number of reduced dimensions = d (map 128 to d features vector)

[114]:
```
# import libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, classification_report
```

[115]:
```
# read data
gender = pd.read_csv("gender.csv")
gender.head()
```

[115]:
```
   Unnamed: 0 Unnamed: 1         0         1         2         3         4  \
0           1       male -0.066420  0.151611  0.027740  0.052771 -0.066105
1           2       male -0.030614  0.049667  0.008084 -0.050324  0.007649
2           3       male -0.096178  0.061127  0.035326 -0.035388 -0.090728
3           4       male -0.103057  0.085044  0.078333 -0.035873 -0.028163
4           5       male -0.125815  0.120046  0.023131 -0.042901  0.038215

          5         6         7  …       118       119       120       121  \
0 -0.041232 -0.002637 -0.158467  …  0.025989 -0.001087  0.027260 -0.046754
1 -0.063818 -0.019530 -0.119905  …  0.044229 -0.023900 -0.028108  0.040618
2 -0.018634 -0.024315 -0.139786  …  0.111141  0.059436 -0.029222  0.042115
3  0.004924  0.007829 -0.017016  …  0.100793 -0.002644 -0.023388  0.029497
4 -0.049677 -0.054258 -0.130758  …  0.090197  0.067527  0.039926  0.047469

        122       123       124       125       126       127
0 -0.118619 -0.163774 -0.000590 -0.076400  0.107497  0.001567
1 -0.146579 -0.141244  0.016162  0.017638  0.080610 -0.015930
2 -0.222173 -0.116908  0.093428  0.017391  0.057652  0.086116
3 -0.139830 -0.119243  0.005306 -0.015100  0.161575  0.062462
4 -0.056852 -0.076700  0.004966  0.028171  0.026041  0.084135

[5 rows x 130 columns]
```

[116]:
```
# Perform PCA on this

# Let us sepearte the target and only take features
df_target = gender.iloc[:,1:2]
df_target.columns = ["label"]

df_f1 = gender.iloc[:,2:]
```

[117]:
```
# Let us apply PCA on df_f1
# find mean vectors

df_f1_mean = df_f1.mean()

# calculate covariance
```

[118]:
```
# implement function for PCA
def PCA(df , k):
    X = df.to_numpy()

    # standardise
    X_std = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

    cov_matrix = np.cov(X_std.T)
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
    eig_pairs = [(np.abs(eigenvalues[i]), eigenvectors[:,i]) for i in␣
 ↪range(len(eigenvalues))]
    eig_pairs.sort(reverse=True, key=lambda x: x[0])
    matrix_w = np.hstack((eig_pairs[i][1].reshape(len(df.columns),1)) for i in␣
 ↪range(k))
    X_pca = X_std.dot(matrix_w)

    # Convert the transformed data back to a DataFrame
    cols = [f"PC{i+1}" for i in range(k)]
    df_pca = pd.DataFrame(X_pca, columns=cols)

    return df_pca
```

[119]:
```
dim = 57
df_n  = PCA(df_f1,dim)
df_n = pd.concat([df_n,df_target],axis = 1)
```

```
/tmp/ipykernel_24505/2208913414.py:12: FutureWarning: arrays to stack must be
passed as a "sequence" type such as list or tuple. Support for non-sequence
iterables such as generators is deprecated as of NumPy 1.16 and will raise an
error in the future.
  matrix_w = np.hstack((eig_pairs[i][1].reshape(len(df.columns),1)) for i in
range(k))
```

[120]:
```
# seperate the test and the train
test_m = df_n.iloc[0:10,:]
test_f = df_n.iloc[400:410,:]
test = pd.concat([test_m,test_f],axis = 0)
test_target = test.iloc[:,-1]
test_samples = test.iloc[:,:-1]
# now train
train_m = df_n.iloc[10:400,:]
train_f = df_n.iloc[410:800,:]
train = pd.concat([train_m,train_f],axis = 0)
```

[121]:
```
# let us obtain the PCA mean
mean_PCA = train.groupby("label").mean()
mean_PCA_male , mean_PCA_female = mean_PCA.xs("male") , mean_PCA.xs("female")
```

[122]:
```
# distance classifier
def dist(X,mean):
    Y = np.sqrt(np.sum((X - mean)**2))
    return Y
```

[123]:
```
# running through test data to check for predictions
dim = 57
predictionsPCA = []
predictionsLabels = []

for i in range(test_samples.shape[0]):
    maleX =  dist(test_samples.iloc[i:i+1].to_numpy(),mean_PCA_male.to_numpy())
    femaleX = dist(test_samples.iloc[i:i+1].to_numpy(),mean_PCA_female.
 ↪to_numpy())
    if maleX > femaleX:
        label = "female"
    else:
        label = "male"
    predictionsLabels.append(label)
```

[124]:
```
test_target = test_target.to_frame()
test_target
```

[124]:

	label
0	male
1	male
2	male
3	male
4	male
5	male
6	male
7	male
8	male
9	male
400	female
401	female
402	female
403	female
404	female
405	female
406	female
407	female
408	female
409	female

[125]:
```
# checking for accuracy
match = 0
```

```
for i in range(len(test_target)):
    expected = test_target["label"].iloc[i]
    prediction = predictionsLabels[i]
    if(prediction == expected):
        match+=1

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

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

```
Accuracy of Classifier : 90.0
```

0.0.2 2. For the same dataset "gender.csv" (2 classes, male and female)

- a) Use LDA to reduce the dimension from d to d . (Here d=128)
- b) Choose the direction „W to reduce the dimension d (select appropriate d).
- c) Use d features to classify the test cases (use any classification algorithm will do,Bayes classifier, minimum distance classifier, and so on).

Implementing using LDA

[126]:
```
df_f2 = pd.concat([df_f1,df_target],axis = 1)
heading = df_f2.columns[-1]
heading
```

[126]:
```
'label'
```

[127]:
```
target = pd.DataFrame(df_target.to_numpy())
target.columns = ["label"]
```

[138]:
```
def LDA(data, target):
    mean = []
    for i in range(len(np.unique(target))):
        name = np.unique(target)[i]
        mean.append(np.mean(data[target == name], axis=0))
    mean = np.array(mean)


    # compute scatter matrices

    mean_all = np.mean(data, axis=0)


    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]
```

```
        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]
        Sb += np.dot((mean[i] - mean_all).values.reshape(data.shape[1], 1),␣
↪(mean[i] - mean_all).values.reshape(1, data.shape[1]))

    # calculate e values and e vectors
    eValues, eVectors = np.linalg.eig(np.dot(np.linalg.inv(Sw), Sb))
    # sort e values and e vectors
    idx = eValues.argsort()[::-1]
    eValues = eValues[idx]
    eVectors = eVectors[:, idx]

    # reduce dimensions to 1
    d = 1
    # print d
    print("d: ", d)
    # reduce dimensions
    eVectors = eVectors[:, :d]
    data = np.dot(data, eVectors)
    data = pd.DataFrame(data)
    return data
```

[139]:
```
LDAdimRdn = LDA(df_f2.iloc[:,:-1],df_f2.iloc[:,-1])
```

```
d:  1
```

[140]:
```
LDAdimRdn = pd.concat([LDAdimRdn,target],axis = 1)
test_m_LDA = LDAdimRdn.iloc[0:10,:]
test_f_LDA = LDAdimRdn.iloc[400:410,:]
test_LDA = pd.concat([test_m_LDA,test_f_LDA],axis = 0)
test_target_LDA = test_LDA.iloc[:,-1]
test_samples_LDA = pd.DataFrame(np.real(test_LDA.iloc[:,:-1]))
# now train
train_m_LDA = LDAdimRdn.iloc[10:400,:]
train_f_LDA = LDAdimRdn.iloc[410:800,:]
train_LDA = pd.concat([train_m_LDA,train_f_LDA],axis = 0)
```

[141]:
```
train_LDA[0] = np.real(train_LDA[0])
train_LDA
```

[141]:

```
                   0   label
10   1.964650e-06    male
11   2.644062e-06    male
12   2.950064e-06    male
13   2.507864e-06    male
14   2.982156e-06    male
..            …      …
795  1.637797e-06  female
796  1.604842e-06  female
797  1.350251e-06  female
798  9.157662e-07  female
799  1.312559e-06  female

[780 rows x 2 columns]
```

[142]:

```
mean_LDA = train_LDA.groupby("label").mean()
mean_LDA_male , mean_LDA_female = mean_LDA.xs("male") , mean_LDA.xs("female")
test_target_LDAn = pd.DataFrame(test_target_LDA)
```

[143]:

```
predictionsLDA = []
predictionsLabelsLDA = []

for i in range(test_samples_LDA.shape[0]):
    maleX =  dist(test_samples_LDA.iloc[i:i+1].to_numpy(),mean_LDA_male.
 ↪to_numpy())
    femaleX = dist(test_samples_LDA.iloc[i:i+1].to_numpy(),mean_LDA_female.
 ↪to_numpy())
    if maleX > femaleX:
        label = "female"
    else:
        label = "male"
    predictionsLabelsLDA.append(label)
```

[144]:

```
testingTarget = pd.DataFrame(test_target_LDAn.to_numpy())
testingTarget.columns = ["label"]
```

[145]:

```
# checking for accuracy
match = 0
for i in range(len(testingTarget)):
    expected = testingTarget["label"].iloc[i]
    prediction = predictionsLabelsLDA[i]
    if(prediction == expected):
        match+=1

accuracy = (match/len(testingTarget))*100
print(match)
print("Accuracy of Classifier :" ,accuracy)
```

```
17
Accuracy of Classifier : 85.0
```

0.0.3 3. Give the comparative study for the above results w.r.t to classification accuracy in terms of the confusion matrix.

[146]:
```python
# Finding the confusion matrix
from sklearn.metrics import confusion_matrix

# create confusion matrix for PCA
tnPCA, fpPCA, fnPCA, tpPCA = confusion_matrix(test_target["label"],
 ↪predictionsLabels).ravel()

print("Confusion Matrix for PCA")
print("TN : ",tnPCA)
print("FN : ",fnPCA)
print("FP : ",fpPCA)
print("TP : ",tpPCA)
```

```
Confusion Matrix for PCA
TN :  9
FN :  1
FP :  1
TP :  9
```

[147]:
```python
# create confusion matrix for LDA
tnLDA, fpLDA, fnLDA, tpLDA = confusion_matrix(test_target["label"],
 ↪predictionsLabelsLDA).ravel()

print("Confusion Matrix for LDA")
print("TN : ",tnLDA)
print("FN : ",fnLDA)
print("FP : ",fpLDA)
print("TP : ",tpLDA)
```

```
Confusion Matrix for LDA
TN :  9
FN :  2
FP :  1
TP :  8
```

question4

April 12, 2023

[3]:
```
# Question 4
# Anirudh Sathish
# CS20B1125
```

0.0.1 4. Eigenfaces-Face classification using PCA (40 classes)

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

[4]:
```
# libs
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
/home/anirudh/.local/lib/python3.8/site-
packages/pandas/core/computation/expressions.py:20: UserWarning: Pandas requires
version '2.7.3' or newer of 'numexpr' (version '2.7.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
```

[5]:
```
# get data
faces = pd.read_csv("face.csv")
faces.head()
```

[5]:
```
          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]
```

Observation

- There are around 4097 columns here , let us reduce this to around 4

[6]:
```
# seperating the target and the features
y = faces["target"]
X = faces.iloc[:,:-1]
```

[7]:
```
# implement function for PCA
def PCA(df , k):
    X = df.to_numpy()

    # standardise
    X_std = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

    cov_matrix = np.cov(X_std.T)
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
    eig_pairs = [(np.abs(eigenvalues[i]), eigenvectors[:,i]) for i in␣
 ↪range(len(eigenvalues))]
    eig_pairs.sort(reverse=True, key=lambda x: x[0])
    matrix_w = np.hstack((eig_pairs[i][1].reshape(len(df.columns),1)) for i in␣
 ↪range(k))
    X_pca = X_std.dot(matrix_w)

    # Convert the transformed data back to a DataFrame
    cols = [f"PC{i+1}" for i in range(k)]
    df_pca = pd.DataFrame(X_pca, columns=cols)

    return df_pca
```

[8]:
```
dim = 32
X_ = PCA(X,dim)
```

```
/tmp/ipykernel_25105/916561548.py:12: FutureWarning: arrays to stack must be
passed as a "sequence" type such as list or tuple. Support for non-sequence
iterables such as generators is deprecated as of NumPy 1.16 and will raise an
```

```
error in the future.
  matrix_w = np.hstack((eig_pairs[i][1].reshape(len(df.columns),1)) for i in
range(k))
```

[9]:
```
X_reduced = pd.DataFrame(np.real(X_))
```

[10]:
```
# 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)
```

0.0.2 Further

Now we need to train this using a classifier

[11]:
```
# lets use the minimum dist classifier
def dist(X,mean):
    Y = np.sqrt(np.sum((X - mean)**2))
    return Y
```

[12]:
```
target = pd.DataFrame(y_train.to_numpy())
target.columns = ["label"]
X_recomplete = pd.concat([pd.DataFrame(X_train.to_numpy()),target],axis = 1)
X_recomplete
```

[12]:
```
             0          1          2          3          4          5  \
0   -12.470149 -22.764214  25.854409  -7.872594 -29.778225  15.559930
1   -13.452591 -38.498148   6.261032   2.939254   4.481049   4.556061
2   -35.465647  -4.926576  -7.895512  13.953842  -5.543100  -6.879875
3   -25.776232 -24.063425  10.668826   1.931122  -3.718425  -2.880087
4     0.238321   8.657132  18.254694  13.704750  -2.343360 -21.417495
..         …          …          …          …          …          …
275 -17.880372   5.887619   4.140011  21.170258  -5.287500 -10.634399
276 -13.796903  24.401468   7.964249   1.970064   5.356964   3.561706
277  48.136781  30.840925  -1.333773   9.199799  20.946557 -14.491309
278  -1.565518  26.154681 -28.312928  -2.682017  14.476955   0.294051
279  18.489601 -48.362660   5.913987 -21.389318  -8.503165  -8.524571

             6          7          8          9  …        23        24  \
0     4.050746   8.460022  -0.199687 -11.354322  … -1.081742 -3.656365
1     5.891295  -0.457788  -5.253466  -6.895147  … -3.228498 -3.974198
2     3.305456  -4.281920  -3.479988 -10.934135  …  0.999866 -2.149057
3    21.570968  10.836741   1.605833   4.938735  … -0.767468 -0.786237
4    -0.625946 -12.444465   5.960021  -0.003394  … -0.777802  0.930801
..         …          …          …          …  …  …        …
```

```
275   3.866715   0.174368  -3.719179   2.081378  …  1.677615  0.409395
276   1.629407 -20.914326  -5.768807  -5.083873  … -2.919046  1.519434
277 -24.954809   3.914137   0.620176  -2.174642  … -0.562042  7.657260
278  18.801110  11.354332  -5.577051   5.767145  … -1.274335 -7.426593
279  -5.861535   1.067542  10.100787  11.317871  … -2.861158 -4.906727

           25        26         27        28        29        30        31  \
0    0.817381 -4.961834   6.313481 -5.174302 -1.528832  0.941685  0.669370
1    0.279343 -2.051225  -4.910065 -0.264720 -0.686249  1.245296  4.396015
2    1.761309  1.730883  -2.350602 -0.423686 -7.856503 -1.560900  0.591192
3    6.978522  7.386602  -5.555768 -2.648823 -0.337393 -3.605931 -6.937106
4    0.050183  1.712293   9.910267 -5.100609  1.578396 -3.644937  1.209926
..        …         …          …         …         …         …         …
275 -4.457368  2.471959   5.820890  1.041556 -0.472682 -0.116670  0.312237
276 -0.648941 -1.746017  -3.288061 -3.427770  2.763512  1.830751 -5.572531
277  6.196506  1.104143  11.503930  1.715735 -2.183316 -1.218641  1.807812
278  3.318042  1.828388   0.512862  1.052292 -9.496571 -4.267750 -4.922249
279  7.471773 -1.812927  -9.581378  3.887263 -6.112964  6.345869  1.775716

     label
0       11
1       20
2       29
3       30
4       34
..      …
275     29
276      2
277      7
278      1
279     16

[280 rows x 33 columns]
```

[13]:
```
meanData = X_recomplete.groupby("label").mean()
```

[14]:
```
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)
```

[15]:

```
y_test = pd.DataFrame(y_test)
```

[16]:

```
# 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 : 85.83333333333333
```

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]:

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

[35]:
```
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]:
```
y_test = y_test.to_frame()
```

[37]:
```
# 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
```