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
```

```python
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]:
```python
# 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]:
```python
# 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]:
```python
# 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
```