q9spam

February 23, 2024

[ ]:
```
import cv2
import seaborn
import sys
if sys.version_info[0] < 3:
        raise Exception("Python 3 not detected.")
import numpy as np
import matplotlib.pyplot as plt
from scipy import io
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
import scipy.stats as stats
from PIL import Image
from collections import defaultdict

spamData = np.load(f"../data/spam-data-hw3.npz")
print("\nloaded %s data!" % "spam")
fields = "test_data", "training_data", "training_labels"
for field in fields:
    print(field, spamData[field].shape)
spamTest = spamData["test_data"]
spamAllData = spamData["training_data"]
spamAllLabels = spamData["training_labels"]
```

```
loaded spam data!
test_data (1000, 32)
training_data (4171, 32)
training_labels (4171,)
```

[ ]:
```
SEED = 0
```

[ ]:
```
spamTrainX, spamTestX, spamTrainY, spamTestY = train_test_split(spamAllData,␣
↪spamAllLabels, test_size=500, random_state=SEED, shuffle=True)
```

[ ]:
```
print("spam trainData shape: ", spamTrainX.shape)
print("spam testData shape: ", spamTestX.shape)
print("spam trainLabel shape: ", spamTrainY.shape)
print("spam testLabel shape: ", spamTestY.shape)
```

```
spam trainData shape:  (3671, 32)
spam testData shape:  (500, 32)
spam trainLabel shape:  (3671,)
spam testLabel shape:  (500,)
```

[ ]:
```python
def computeMean(inputData, inputLabels):
    #Fit Gaussian to Data
    spamTrainCount = np.zeros(2)
    spamTrainRunningSum = np.zeros((2, 32))
    spamTrainMean = np.zeros((2, 32))

    for (data, label) in zip(inputData, inputLabels):
        spamTrainCount[label] += 1
        spamTrainRunningSum[label] += data

    for i in range(0,2):
        spamTrainMean[i] = spamTrainRunningSum[i] / spamTrainCount[i]

    # print(mnistTrainRunningSum.shape)
    # print(mnistTrainMean.shape)
    # print(mnistTrainCount[0])

    return spamTrainCount, spamTrainMean
```

[ ]:
```python
def computeClassVarience(inputData, inputLabels, mean):
    #QDA Estimation of Varience
    classVar = np.zeros((10, 32, 32))

    for (data, label) in zip(inputData, inputLabels):
        meanDiff = data - mean[label]
        classVar[label] += np.outer(meanDiff, meanDiff)
    return classVar
```

[ ]:
```python
def computeLDAVar(classVar, totalPoints):
    #Find Pooled Varience
    pooledVarience = np.zeros((32, 32))
    for i in range(0, 2):
        pooledVarience += classVar[i]

    spamTrainLDAVar = pooledVarience / totalPoints

    return spamTrainLDAVar
```

[ ]:
```python
def linDiscrim(classIndex, classMean, point, classPreCalc, priorProb):
    result = np.dot(classPreCalc[classIndex], point)
    result = result - (np.dot(classPreCalc[classIndex], classMean[classIndex]))/
 ↪2
```

```
    result = result + np.log(priorProb)
    return result

errorLin = []
for numTestPoints in [100, 200, 500, 1000, 2000, 3671]:
    #Train on i points
    trainSetX = spamTrainX[:numTestPoints]
    trainSetY = spamTrainY[:numTestPoints]

    # print(trainSetX.shape)
    # print(trainSetY.shape)

    classCount, classMean = computeMean(trainSetX, trainSetY)
    classVar = computeClassVarience(trainSetX, trainSetY, classMean)
    LDAVar = computeLDAVar(classVar, numTestPoints)
    LDAVarPseudoInv = np.linalg.pinv(LDAVar)

    # print(classMean.shape)
    # print(LDAVarPseudoInv.shape)

    classPreCalc = {}
    #Precalculate some values
    for i in range(0, 2):
        classPreCalc[i] = np.dot(classMean[i].T, LDAVarPseudoInv)

    #Using linear Disc: Test
    errorCount = 0
    correctCount = 0
    for testPointIndex in range(0, len(spamTestX)):
        linDisc = np.empty((2))
        for i in range(0, 2):
            pi = classCount[i]/numTestPoints
            #print(pi)
            linDisc[i] = linDiscrim(i, classMean, spamTestX[testPointIndex],␣
 ↪classPreCalc, pi)
        pred = np.argmax(linDisc)
        if (pred != spamTestY[testPointIndex]):
            errorCount += 1
        else:
            correctCount += 1
    print(1-(correctCount/(errorCount+correctCount)))
    errorLin.append((numTestPoints, errorCount/(errorCount+correctCount)))
```

```
0.252
0.19799999999999995
0.18400000000000005
0.17800000000000005
0.17400000000000004
```

0.18000000000000005

[ ]:
```
for numTestPoints in [2000]:
    #Train on i points
    trainSetX = spamTrainX[:numTestPoints]
    trainSetY = spamTrainY[:numTestPoints]

    # print(trainSetX.shape)
    # print(trainSetY.shape)

    classCount, classMean = computeMean(trainSetX, trainSetY)
    classVar = computeClassVarience(trainSetX, trainSetY, classMean)
    LDAVar = computeLDAVar(classVar, numTestPoints)
    LDAVarPseudoInv = np.linalg.pinv(LDAVar)

    # print(classMean.shape)
    # print(LDAVarPseudoInv.shape)

    classPreCalc = {}
    #Precalculate some values
    for i in range(0, 2):
        classPreCalc[i] = np.dot(classMean[i].T, LDAVarPseudoInv)

    pred = []
    for testPointIndex in range(0, len(spamTest)):
        linDisc = np.empty((2))
        for i in range(0, 2):
            pi = classCount[i]/numTestPoints
            linDisc[i] = linDiscrim(i, classMean, spamTest[testPointIndex],␣
↪classPreCalc, pi)
        pred.append(np.argmax(linDisc))
```

[ ]:
```
print(pred)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
```

```
0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0]
```

[ ]:
```
import pandas as pd
list = [*range(1, len(pred)+1)]
outputDict = {"Id":list, "Category": pred}
df = pd.DataFrame(outputDict)
df.to_csv('spamResult.csv', index=False)
```

[ ]:
```

```