pa3

February 25, 2021

[1]:

```
import math
import statistics
import numpy as np
import pandas as pd
import random
import copy
from scipy import stats
```

[2]:

```
testDataFile = open("pa3test.txt")
testlist = list(testDataFile)
testMatrix = [e.split(" ") for e in testlist]
testMatrix = [[int(x) for x in i] for i in testMatrix]

trainDataFile = open("pa3train.txt")
trainlist = list(trainDataFile)
trainMatrix = [e.split(" ") for e in trainlist]
trainMatrix = [[float(x) for x in i] for i in trainMatrix]

dictDataFile = open("pa3dictionary.txt")
dictionary = list(dictDataFile)
dictionary = [x[:-2] for x in dictionary]
```

[3]:

```
testMatrixLabel12 = []
trainMatrixLabel12 = []

for dataPoint in testMatrix:
    if dataPoint[819] == 1 or dataPoint[819] == 2:
        testMatrixLabel12.append(dataPoint)
for dataPoint in trainMatrix:
    if dataPoint[819] == 1 or dataPoint[819] == 2:
        trainMatrixLabel12.append(dataPoint)
```

[4]:

```
## label 1 is -1 and label 2 is 1
def buildPerceptron(w):
    for i in range(0, len(trainMatrixLabel12) - 1):
        dotproduct = sum(a[0] * a[1] for a in zip(w[i], trainMatrixLabel12[i][:
 ↪819]))
        y = 1 if (trainMatrixLabel12[i][819] == 2) else -1
        if (dotproduct * y) <= 0:
            yx = [a * y for a in trainMatrixLabel12[i][:819]]
            w[i+1] = [sum(x) for x in zip(w[i], yx)]
        else:
            w[i+1] = w[i]
```

[5]:

```
def findPerceptronError(dataMatrix, resultW):
    numWrong = 0;
    numTotal = 0;
    for i in dataMatrix:
        dotproduct = sum(a[0] * a[1] for a in zip(resultW, i[:819]))
        y = 1 if (i[819] == 2) else -1
        if (dotproduct * y) <= 0:
            numWrong += 1
        numTotal += 1
    return (numWrong/numTotal)
```

[6]:

```
zeroVec = [ [0] * 819 for _ in range(len(trainMatrixLabel12))]
w = zeroVec
```

[7]:

```
##find error
buildPerceptron(w)
resultW = w[1089]
print("1-pass trainning error is: " +␣
 ↪str(findPerceptronError(trainMatrixLabel12, resultW)))
print("1-pass testing   error is: " +␣
 ↪str(findPerceptronError(testMatrixLabel12, resultW)))
w = zeroVec
w[0] = resultW
buildPerceptron(w)
resultW = w[1089]
print("2-pass trainning error is: " +␣
 ↪str(findPerceptronError(trainMatrixLabel12, resultW)))
print("2-pass testing   error is: " +␣
 ↪str(findPerceptronError(testMatrixLabel12, resultW)))
w = zeroVec
w[0] = resultW
buildPerceptron(w)
resultW = w[1089]
print("3-pass trainning error is: " +␣
 ↪str(findPerceptronError(trainMatrixLabel12, resultW)))
print("3-pass testing   error is: " +␣
 ↪str(findPerceptronError(testMatrixLabel12, resultW)))
w = zeroVec
w[0] = resultW
buildPerceptron(w)
resultW = w[1089]
```

```
print("4-pass trainning error is: " +␣
 ↪str(findPerceptronError(trainMatrixLabel12, resultW)))
print("4-pass testing   error is: " +␣
 ↪str(findPerceptronError(testMatrixLabel12, resultW)))
```

```
1-pass trainning error is: 0.04128440366972477
1-pass testing   error is: 0.05305039787798409
2-pass trainning error is: 0.04036697247706422
2-pass testing   error is: 0.0610079575596817
3-pass trainning error is: 0.021100917431192660
3-pass testing   error is: 0.04509283819628647
4-pass trainning error is: 0.01926605504587156
4-pass testing   error is: 0.04774535809018567
```

[8]:

```
def buildLogisticRegression(w, iteration):
    for i in range(0, iteration-1):
        summation = [0] * len(trainMatrixLabel12);
        for n in range(0, len(trainMatrixLabel12)):
            y = 1 if (trainMatrixLabel12[n][819] == 2) else -1
            yx = [a * y for a in trainMatrixLabel12[n][:819]]
            wx = sum(a[0] * a[1] for a in zip(w[i], trainMatrixLabel12[n][:
 ↪819]))
            ywx = y * wx
            ##if overflown set element to 0
            try:
                element = [a / (1 + math.exp(ywx)) for a in yx]
            except OverflowError:
                element = [0 for a in yx]
            summation = [sum(a) for a in zip(summation, element)]
        stepped = [a * 0.001 for a in summation]
        w[i+1] = [sum(a) for a in zip(w[i], stepped)]
```

[9]:

```
iterations = 100
wLog = [ [0] * 819 for _ in range(iterations)]
buildLogisticRegression(wLog, iterations)
print("10-pass log regression trainning error: " +␣
 ↪str(findPerceptronError(trainMatrixLabel12, wLog[9])))
print("10-pass log regression testing   error: " +␣
 ↪str(findPerceptronError(testMatrixLabel12, wLog[9])))
print("50-pass log regression trainning error: " +␣
 ↪str(findPerceptronError(trainMatrixLabel12, wLog[49])))
print("50-pass log regression testing   error: " +␣
 ↪str(findPerceptronError(testMatrixLabel12, wLog[49])))
print("100-pas log regression trainning error: " +␣
 ↪str(findPerceptronError(trainMatrixLabel12, wLog[99])))
print("100-pas log regression testing   error: " +␣
 ↪str(findPerceptronError(testMatrixLabel12, wLog[99])))
```

```
10-pass log regression trainning error: 0.44036697247706424
10-pass log regression testing   error: 0.4562334217506631
50-pass log regression trainning error: 0.04128440366972477
50-pass log regression testing   error: 0.0610079575596817
100-pas log regression trainning error: 0.02018348623853211
100-pas log regression testing   error: 0.04509283819628647
```

[10]:

```
#find 3 pass w
w = zeroVec
buildPerceptron(w)
resultW = w[1089]
w = zeroVec
w[0] = resultW
buildPerceptron(w)
resultW = w[1089]
w = zeroVec
w[0] = resultW
buildPerceptron(w)
threePassW = w[1089]

big1 = threePassW.index(max(threePassW))
threePassW[big1] = 0
big2 = threePassW.index(max(threePassW))
threePassW[big2] = 0
big3 = threePassW.index(max(threePassW))

print("top three highest coordinates in perceptron are: " + dictionary[big1] +␣
 ↪", " + dictionary[big2] + ", " + dictionary[big3])

smol1 = threePassW.index(min(threePassW))
threePassW[smol1] = 0
smol2 = threePassW.index(min(threePassW))
threePassW[smol2] = 0
smol3 = threePassW.index(min(threePassW))

print("top three lowest coordinates in perceptron are: " + dictionary[smol1] +␣
 ↪", " + dictionary[smol2] + ", " + dictionary[smol3])

big1 = wLog[49].index(max(wLog[49]))
wLog[49][big1] = 0
big2 = wLog[49].index(max(wLog[49]))
wLog[49][big2] = 0
big3 = wLog[49].index(max(wLog[49]))

print("top three highest coordinates in log reg are: " + dictionary[big1] + ",␣
 ↪" + dictionary[big2] + ", " + dictionary[big3])
```

```
smol1 = wLog[49].index(min(wLog[49]))
wLog[49][smol1] = 0
smol2 = wLog[49].index(min(wLog[49]))
wLog[49][smol2] = 0
smol3 = wLog[49].index(min(wLog[49]))

print("top three lowest coordinates in log reg are: " + dictionary[smol1] + ",␣
 ↪" + dictionary[smol2] + ", " + dictionary[smol3])
```

```
top three highest coordinates in perceptron are: he, team, game
top three lowest coordinates in perceptron are: file, program, line
top three highest coordinates in log reg are: he, game, they
top three lowest coordinates in log reg are: window, file, use
```

[11]:

```
## label else is -1 and label label is 1
def buildOneVSAll(label):
    w = [ [0] * 819 for _ in range(len(trainMatrix))]
    for i in range(0, len(trainMatrix) - 1):
        dotproduct = sum(a[0] * a[1] for a in zip(w[i], trainMatrix[i][:819]))
        y = 1 if (trainMatrix[i][819] == label) else -1
        if (dotproduct * y) <= 0:
            yx = [a * y for a in trainMatrix[i][:819]]
            w[i+1] = [sum(x) for x in zip(w[i], yx)]
        else:
            w[i+1] = w[i]
    return w[len(trainMatrix) - 1]
```

[12]:

```
Classifier = [ [0] * 819 for _ in range(6)]
for i in range(1, 7):
    Classifier[i - 1] =  buildOneVSAll(i)
```

[13]:

```
#return label for the label predicted, -1 for dont know
def predict(dataPoint):
    # 1 for this label, -1 for other label
    predictions = []
    unique = 0
    otherNum = 0
    for i in range(0, len(Classifier)):
        dotproduct = sum(a[0] * a[1] for a in zip(Classifier[i], dataPoint))
        if dotproduct >= 0:
            predictions.append(1)
        else:
            predictions.append(-1)
    for j in range(0, len(predictions)):
        if predictions[j] == 1:
            if unique == 1:
                return -1
            else:
                uniqueInd = j
                unique = 1
        else:
            otherNum += 1
            if otherNum == 6:
                return -1
    return uniqueInd + 1
```

[14]:
```
def findNumLabel(labelNumList):
    for i in range(1, 7):
        for dp in testMatrix:
            if dp[819] == i:
                labelNumList[i-1] += 1

labelNumList = [0]*6
findNumLabel(labelNumList)
```

[15]:
```
def buildConfusion(confusionMatrix):
    predictionList = []
    for dataPoint in testMatrix:
        predictionList.append(predict(dataPoint))

    for i in range(0,6):
        for j in range(0,6):
            count = 0
            for k in range(0, len(predictionList)):
                if (predictionList[k] == (i+1) and testMatrix[k][819] == (j+1)):
                    count += 1
            confusionMatrix[i][j] = count / labelNumList[j]
    for j in range(0,6):
        count = 0
        for k in range(0, len(predictionList)):
            if (predictionList[k] == -1 and testMatrix[k][819] == (j+1)):
                count += 1
        confusionMatrix[6][j] = count / labelNumList[j]
```

[16]:
```
confusionMatrix = [ [0] * 6 for _ in range(7)]
buildConfusion(confusionMatrix)
```

[17]:
```
print(pd.DataFrame(confusionMatrix))
```

```
          0         1         2         3         4         5
0  0.702703  0.010417  0.028571  0.021739  0.000000  0.000000
1  0.037838  0.677083  0.045714  0.032609  0.025641  0.037037
2  0.000000  0.015625  0.337143  0.000000  0.000000  0.000000
3  0.010811  0.005208  0.000000  0.646739  0.000000  0.000000
4  0.016216  0.015625  0.040000  0.005435  0.724359  0.111111
5  0.010811  0.015625  0.034286  0.000000  0.076923  0.574074
```

```
6  0.221622  0.260417  0.514286  0.293478  0.173077  0.277778
```

0.1 The perceptron classifier has the highest accuracy for class 5, since the matrix[4][4] entry is highest.

0.2 The perceptron classifier has the least accuracy for class 3, since the matrix[2][2] entry is lowest.

0.3 The perceptron classifier most often mistakenly classifies an example in class 6 as belonging to class 5, since the highest non-diagonal entry is [4][5].