Designing a Minimum Distance to Class Mean Classifier

Tahira Salwa Rabbi Nishat
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh.
160204070@aust.edu

***Abstract*- The minimum distance to class mean classifier works on unclassified data and classifies them according to their distance from each class. If the distance is nearer to class1, then it is classified to class2, if the distance is nearer to class2 then it is classified to class2. A decision boundary line is drawn to separate the classes. From the experiment of the given test data, the accuracy of the found 85.714%.**

***Index Terms*- Classifier, Class Mean Classifier, Mean Distance Classifier, Decision Boundary.**

I. INTRODUCTION

The minimum distance to class mean classifier is one of the basic classifiers. This classifier is trained with a sample data set. Then the mean of each class is derived. From the test data set the distance of each data from those mean points are taken. Test data are classified based on the shortest distance found from any class.

To classify data, the function which is used here,

$$g_i(X) = X^T{}_i\overline{Y} - \tfrac{1}{2}\,{}_i\overline{Y}^T{}_i\overline{Y} \qquad (1)$$

Here, $\overline{Y}$ is the mean value of each class and X is the feature vector.

II. EXPERIMENTAL DESIGN / METHODOLOGY

We were given two data sets. One for training purposes, and the other for testing purposes. The training dataset consists of features and classes. The feature was a 12X2 matrix and their class was given. The steps I have followed for this experience are:

1. I have loaded the train file and test file using loadtxt() method from numpy library.
2. Then, I inserted the train data into two classes, using the given class data.
3. Then, I plotted them using matplotlib.pyplot.

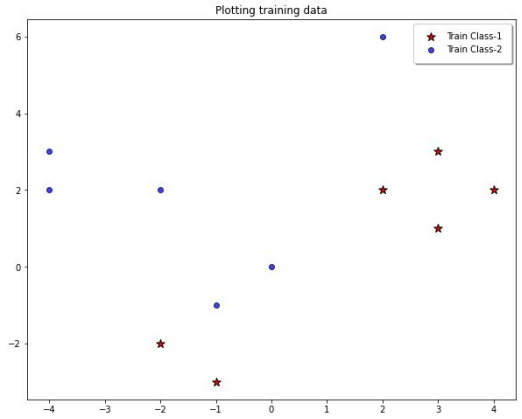

Figure 1. Plotting train data

4. Then I derived the mean value of each class using numpy.mean().
5. I took the test data, converted it into a matrix using numpy.asmatrix(), assigned the first two columns which is the feature vector into a variable, and the last column into another variable that refers to their given classes.
6. Then , I separated the test data into 2 classes, so that it can later be used to check the accuracy.
7. Now, the test data were classified into 2 classes using the function 1.
8. After that, I plotted the train data, test data and the mean data using matplotlib.pyplot.

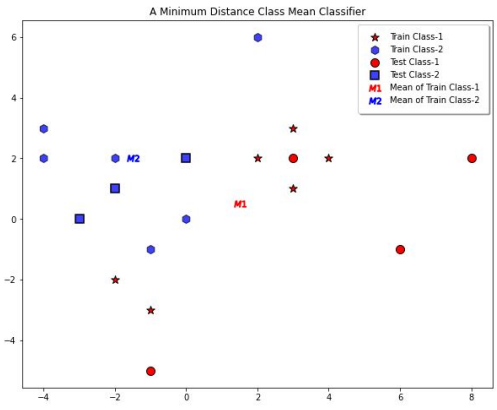

Figure 2: Plotting train data, test data and mean of train data.

9. Then , I calculated the decision boundary from the equation

$$g_i(x) = g_j(x)$$

From this, we can derive that,

$$\omega_i{}^T x - \tfrac{1}{2}\omega_i{}^T\omega_i = \omega_j{}^T x - \tfrac{1}{2}\omega_j{}^T\omega_j$$

$$\Rightarrow x(\omega_i{}^T - \omega_j{}^T) - \tfrac{1}{2}(\omega_i{}^T\omega_i - \omega_j{}^T\omega_j) = 0$$

From this equation. I took the $(\omega_i{}^T - \omega_j{}^T)$ as coefficient and $\frac{1}{2}(\omega_i{}^T\omega_i - \omega_j{}^T\omega_j)$ as constant and calculated y.

$$y = -\frac{m_1 x + c}{m_2}$$

10. Finally I plotted the train data, test data, mean of train data and the decision boundary.
11. At last, I calculated accuracy using the class data I found from step 7 and given class data from the text file.

III. RESULT ANALYSIS

The test data had 7X2 features and using the minimum distance to mean class classifier algorithm, we assigned them into two classes. Three of them were assigned into class 1 and 4 of them were assigned into class 2. Some of the training data were misclassified.

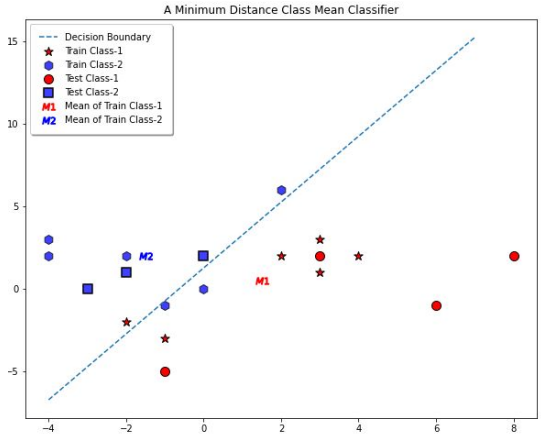

FIgure 3: Decision boundary with train data, test data and mean of train data

Hence, the accuracy was 85.714%.

IV. CONCLUSION

In the experiment, some data got classified correctly, where some were misclassified. It was comparatively easy to implement as there was no complicated calculation to do. It is a pretty basic linear classifier. So, the accuracy rate seems to be a bit lower. If it were a nonlinear classifier, the accuracy rate would be higher.

V. ALGORITHM IMPLEMENTATION / CODE

```
from google.colab import files
uploaded = files.upload()
#Reading Train and Test data files
import numpy as np
trainData = np.loadtxt("train.txt",dtype=int,
delimiter = " ")
testData = np.loadtxt("test.txt", dtype=int,
delimiter = " ")
#Distributing train data into trainClass1 and
trainClass2 based on the data of the last
column
trainClass1                                    =
trainData[np.where(trainData[:,-1]==1)]

trainClass2                                    =
trainData[np.where(trainData[:,-1]==2)]


print('çlass1= ',trainClass1)
print('çlass2= ',trainClass2)
#Plotting train data
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, figsize = (10, 8))
```

```
plt.scatter(trainClass1[:,0], trainClass1[:,1], c = 'red', marker='*', s= 100, label='Train Class-1', edgecolors='black')
plt.scatter(trainClass2[:,0], trainClass2[:,1], c = 'blue', label='Train Class-2', edgecolors='black', alpha=0.75)

plt.title("Plotting training data")
ax.legend(frameon=True, shadow=True, borderpad=1)


plt.show()

#finding the mean of class 1
meanOfClass1 = [np.mean(trainClass1[:,0]),np.mean(trainClass1[:,1])]
meanOfClass1 = np.matrix(meanOfClass1)

#finding the mean of class 2
meanOfClass2 = [np.mean(trainClass2[:,0]),np.mean(trainClass2[:,1])]
meanOfClass2 = np.matrix(meanOfClass2)#converting the test data in a matrix form
testData = np.asmatrix(testData)
#taking the first two column of test data and assigning them in to convertedData
covertedTestData =  testData[:, [0,1]]
#taking the first last column of test data and assigning them in to givenClassOfData
givenClassOfTestData = testData[:,[2]]
#Distributing test data into testClass1 and testClass2 based on the given data of the last column
testClass1 = testData[np.where(testData[:,-1]==1)]

testClass2 = testData[np.where(testData[:,-1]==2)]
#classify test data using the minimum distance to class mean classifier
class1FromTestData = np.empty((0,2))
class2FromTestData = np.zeros((0,2))
foundClassOfTestData = []

for i in range(len(testData)):
            g1 = np.dot(meanOfClass1, covertedTestData[i].transpose()) - (0.5 * np.dot(meanOfClass1, meanOfClass1.transpose()))
            g2 = np.dot(meanOfClass2, covertedTestData[i].transpose()) - (0.5 * np.dot(meanOfClass2, meanOfClass2.transpose()))
  if g1 > g2:
                    class1FromTestData = np.append(class1FromTestData, [[covertedTestData[i].item(0), covertedTestData[i].item(1)]], axis=0)
    foundClassOfTestData.append(1)
  else:
                    class2FromTestData = np.append(class2FromTestData, [[covertedTestData[i].item(0), covertedTestData[i].item(1)]], axis=0)
    foundClassOfTestData.append(2)
#plotting train data, test data, mean of train data and decision boundary
fig, ax = plt.subplots(1, figsize = (10, 8))



plt.title("A Minimum Distance Class Mean Classifier")


#train data
ax.scatter(trainClass1[:,0], trainClass1[:,1], c = 'red', marker='*', s= 100, label='Train Class-1', edgecolors='black', linewidths=1)
ax.scatter(trainClass2[:,0], trainClass2[:,1], c = 'blue', marker='h', s= 100, label='Train Class-2', edgecolors='black', alpha=0.75, linewidths=1)

#test data
ax.scatter([class1FromTestData[:,0]], [class1FromTestData[:,1]], c = 'red', marker='o', s= 100, label='Test Class-1', edgecolors='black', linewidths=1)
ax.scatter([class2FromTestData[:,0]], [class2FromTestData[:,1]], c = 'blue', marker='s',s= 100, label='Test Class-2', edgecolors='black', alpha=0.75, linewidths=2)

#mean of train data
ax.scatter(meanOfClass1[0,0], meanOfClass1[0,1], c = 'red', marker='$M1$', s= 200, label='Mean of Train Class-1', linewidths=1)
ax.scatter(meanOfClass2[0,0], meanOfClass2[0,1], c = 'blue', marker='$M2$',s= 200, label='Mean of Train Class-2', linewidths=1)
```

```
ax.legend(frameon=True,             shadow=True,
borderpad=1)
plt.show()
#Generating the Decision Boundary for a minimum
distance to class mean classifier
X=[]
Y=[]

mc1 = np.asarray(meanOfClass1).flatten()
mc2 = np.asarray(meanOfClass2).flatten()

for i in range(-4,8,1):
                    j=     -((mc1[0]-mc2[0])*i-
0.5*((np.dot(np.transpose(mc1),mc1))           -
(np.dot(np.transpose(mc2),mc2))))/(mc1[1]-mc2[1
])

    X.append(i)
    Y.append(j)
#plotting train data, test data, mean of train
data and decision boundary
fig, ax = plt.subplots(1, figsize = (10, 8))

ax.plot(X,Y,"--", label='Decision Boundary')

plt.title("A   Minimum   Distance   Class   Mean
Classifier")


#train data
ax.scatter(trainClass1[:,0],   trainClass1[:,1],
c  =  'red',  marker='*',  s=  100,  label='Train
Class-1', edgecolors='black', linewidths=1)
ax.scatter(trainClass2[:,0],   trainClass2[:,1],
c  =  'blue',  marker='h',  s=  100,  label='Train
Class-2',    edgecolors='black',    alpha=0.75,
linewidths=1)

#test data
ax.scatter([class1FromTestData[:,0]],
[class1FromTestData[:,1]],    c    =    'red',
marker='o',   s=   100,   label='Test   Class-1',
edgecolors='black', linewidths=1)
ax.scatter([class2FromTestData[:,0]],
[class2FromTestData[:,1]],    c    =    'blue',
marker='s',s=    100,    label='Test    Class-2',
edgecolors='black', alpha=0.75, linewidths=2)
#mean of train data
ax.scatter(meanOfClass1[0,0],
meanOfClass1[0,1], c = 'red', marker='$M1$', s=
200,    label='Mean    of    Train    Class-1',
linewidths=1)
ax.scatter(meanOfClass2[0,0],
meanOfClass2[0,1], c = 'blue', marker='$M2$',s=
200,    label='Mean    of    Train    Class-2',
linewidths=1)
```

```
ax.legend(frameon=True,             shadow=True,
borderpad=1)
plt.show()
#calculating accuracy
accuracy = 0

for    i,    j    in    zip(givenClassOfTestData,
foundClassOfTestData):
  if i == j:
    accuracy = accuracy + 1

accuracy                                        =
(accuracy/len(givenClassOfTestData))*100

print(accuracy)
```