Designing a Minimum Distance to Class Mean Classifier

Shimul Paul
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204014@aust.edu

***Abstract*—Minimum distance to class mean classifier is a kind of classifier which characterizes an unclassified sample vector as per the base distance between the unclassified one and mean of the classes.In our dataset there are some classified sample vectors.From our test set we can now classify some other sample.This is a very simple classifier technique.**

***Index Terms*—Class mean classifier,Minimum distance**

I. Introduction

Minimum distance to class mean classifier is used for classifying the unknown sample vectors.In this classifier the distance is measured from the mean of the classes.As it is a very simple classifier so it does not need so much calculation.So this classifier takes a very little computation and time for classifying the unknown samples.

We are using the below function for classification,

$$g_i(x) = X\bar{Y}_i^T - \frac{1}{2}\bar{Y}_i^T\bar{Y}_i \tag{1}$$

In function 1, $\bar{Y}$ is the mean of an individual class and X is the feature vector of the class.

II. Experimental Design / Methodology

In the given experiment two dataset were given.One for training and another one for testing.The training dataset consists of 12*2 feature set with labeled class.Also the testing dataset has given with 7*2 feature set with labeling class.Here I give the steps for my experimental design and methodology for classification.

- Firstly, I read all the training data using numpy loadtxt() function and store it in a numpy array.
- Then, I also read all the testing data using numpy loadtxt() function and store it also in an array.
- Then, for the training data I store the values of two feature in two arrays if the class is 1 then classA else in classB.
- Also, for test data I separate the values of two feature in a array and also the class in another array for further testing the accuracy.
- After that, I plot the training classA and training classB values using matplotlib with marker and color.
- Then, I calculate the mean of the training classA and classB using numpy mean function with setting the axis to 0.
- Then, I calculate the the value of testing data using the function 1.
- After that I plot the all training, testing and mean values using different marker and color.
- Then I calculate the decision boundary from the general equation given below:

$$g_1(x) = g_2(x) \tag{2}$$

From here, I calculate the generalize equation which is:

$$Y = -((\bar{Y}_{Ax} - \bar{Y}_{Bx}) * x - \frac{1}{2} * (\bar{Y}_A^T\bar{Y}_A + \bar{Y}_B^T\bar{Y}_B))/(\bar{Y}_{Ay} - \bar{Y}_{By}) \tag{3}$$

Here, $\bar{Y}_{Ax}$ and $\bar{Y}_{Ay}$ are the mean from ClassA and $\bar{Y}_{Bx}$ and $\bar{Y}_{By}$ are the mean from classB. I can get some values of Y for some X_i values.

- Then, I plotted the decision boundary along with training,test and mean class data.
- Finally, I calculate the accuracy using the predicted class and actual test class labeling

III. Algorithm Implementation / Code

Name:Shimul Paul ID: 160204014 Section:A1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as matplot
```

```
#Read the train file
trainData = np.loadtxt('D:/All Videos and Lectures of 4.2/Pattern Lab/Lab 2/train.txt',delimiter = ' ')
#trainData
```

```
#Read the test file
testData = np.loadtxt('D:/All Videos and Lectures of 4.2/Pattern Lab/Lab 2/test.txt',delimiter = ' ')
#testData
```

```
#Let Assume if 1 then its in class A
classA =[]
for a in trainData:
    if a[2] == 1:
        classA.append([a[0],a[1]])

#classA
```

```
#Let Assume if 1 then its in class B
classB =[]
for a in trainData:
    if a[2] == 2:
        classB.append([a[0],a[1]])
#classB
```

```
#let separate test values and their class
testValues =[]
testClass =[]
for a in testData:
    testValues.append([a[0],a[1]])
    testClass.append(a[2])
#print('Values: ',testValues)

#print('Classes: ',testClass)
```

```
#convert the list to np array
classA = np.array(classA)
classB = np.array(classB)
testValues = np.array(testValues)
testClass = np.array(testClass)
#print('testValues class type',type(testClass))
```

```
fig,ax = matplot.subplots()
ax.set_title('Ploting Training Data')
fig.set_figheight(5)
fig.set_figwidth(10)
ax.scatter(classA[:,0],classA[:,1],marker='o',color='r',label='Train Class 1')
ax.scatter(classB[:,0],classB[:,1],marker='*',color='k',label='Train Class 2')
legend = ax.legend(loc='upper left', shadow=False, fontsize='small',labelspacing=1)
legend.get_frame().set_facecolor('None')
matplot.show()
```

```
#mean of class a and class b
meanClassA = np.mean(classA,axis = 0)
meanClassB = np.mean(classB,axis = 0)
```

```
print(meanClassA)
print(meanClassB)
```

```
[1.5 0.5]
[-1.5  2. ]
```

```
predictionClassA = [0]*len(testValues)
predictionClassB = [0]*len(testValues)
predictionClass = [0]*len(testValues)
```

```
#create a temporary numpy array for store
tempValueClass = np.zeros((len(testValues),3))
#print(tempValueClass)
```

```
for i in range(len(testValues)):
    predictionClassA[i] = np.dot(np.transpose(meanClassA),testValues[i]-.5*np.dot(np.transpose(meanClassA),meanClassA))
    predictionClassB[i] = np.dot(np.transpose(meanClassB),testValues[i]-.5*np.dot(np.transpose(meanClassB),meanClassB))
    if(predictionClassA[i]>predictionClassB[i]):
        tempValueClass[i][0] = testValues[i][0]
        tempValueClass[i][1] = testValues[i][1]
        tempValueClass[i][2] = 1
        predictionClass[i] = 1
    else:
        tempValueClass[i][0] = testValues[i][0]
        tempValueClass[i][1] = testValues[i][1]
        tempValueClass[i][2] = 2
        predictionClass[i] = 2
```

```
#Let tempValueClass if 1 then its in class A
classtestA =[]
for a in tempValueClass:
    if a[2] == 1:
        classtestA.append([a[0],a[1]])
#classA
```

```
#Let tempValueClass if 2 then its in class B
classtestB =[]
for a in tempValueClass:
    if a[2] == 2:
        classtestB.append([a[0],a[1]])

#classB
```

```
#convert the list in numpy array
classtestA =np.array(classtestA)
classtestB = np.array(classtestB)
predictionClass = np.array(predictionClass)
```

```
#plotting the meanclass,predicted class and training class
fig, ax = matplot.subplots()
fig.set_figheight(5)
fig.set_figwidth(10)

ax.scatter(classA[:,0],classA[:,1],marker='o',color='r',label='Train Class 1')
ax.scatter(classB[:,0],classB[:,1],marker='*',color='k',label='Train Class 2')
ax.scatter(meanClassA[0],meanClassA[1],marker='v',color='r',label='Mean Class 1')
ax.scatter(meanClassB[0],meanClassB[1],marker='H',color='k',label='Mean Class 2')
ax.scatter(classtestA[:,0],classtestA[:,1],marker='s',color='r',label='Test Class 1')
ax.scatter(classtestB[:,0],classtestB[:,1],marker='s',color='k',label='Test Class 2')
legend = ax.legend(loc='upper left', shadow=False, fontsize='small',labelspacing=0.5)
legend.get_frame().set_facecolor('None')
matplot.show()
```

```
X=[0]*12
Y=[0]*12
c=0
for x in range(-4,8,1):
    j=-((meanClassA[0]-meanClassB[0])*x-.5*np.dot(np.transpose(meanClassA),meanClassA)+
        .5*np.dot(np.transpose(meanClassB),meanClassB))/(meanClassA[1]-meanClassB[1])
    X[c]=x
    Y[c]=j
    c+=1
```

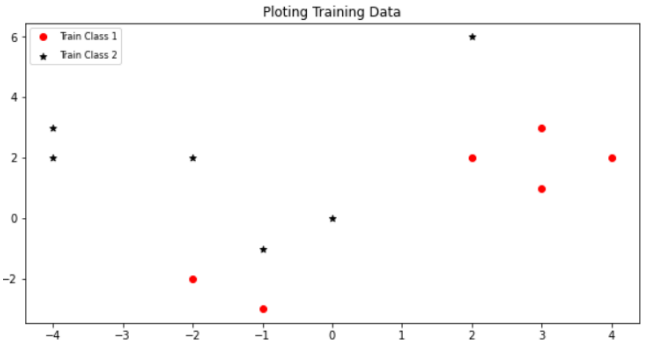
```
fig, ax = matplot.subplots()
fig.set_figheight(5)
fig.set_figwidth(10)
ax.plot(X,Y,"--",label='Decision Boundary',color="g")
ax.scatter(classA[:,0],classA[:,1],marker='o',color='r',label='Train Class 1')
ax.scatter(classB[:,0],classB[:,1],marker='*',color='k',label='Train Class 2')
ax.scatter(meanClassA[0],meanClassA[1],marker='v',color='r',label='Mean Class 1')
ax.scatter(meanClassB[0],meanClassB[1],marker='H',color='k',label='Mean Class 2')
ax.scatter(classtestA[:,0],classtestA[:,1],marker='s',color='r',label='Test Class 1')
ax.scatter(classtestB[:,0],classtestB[:,1],marker='s',color='k',label='Test Class 2')
legend = ax.legend(loc='upper left', shadow=False, fontsize='small',labelspacing=0.5)
legend.get_frame().set_facecolor('None')
matplot.show()
```

```
accuracy = 0
for x,y in zip(testClass,predictionClass):
    if (x == y):
        accuracy +=1
accuracy = accuracy/len(testClass)*100
print('Accuracy = ', accuracy)
```

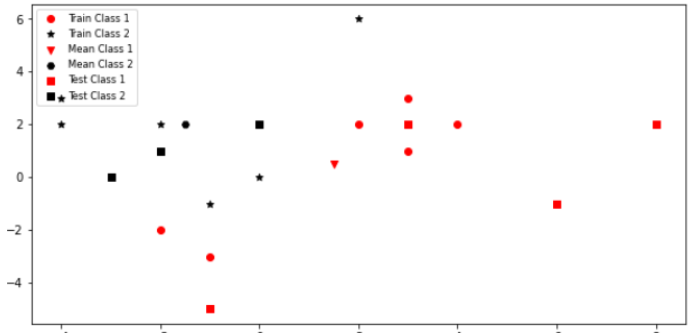
IV. Result Analysis

The results is shown below:

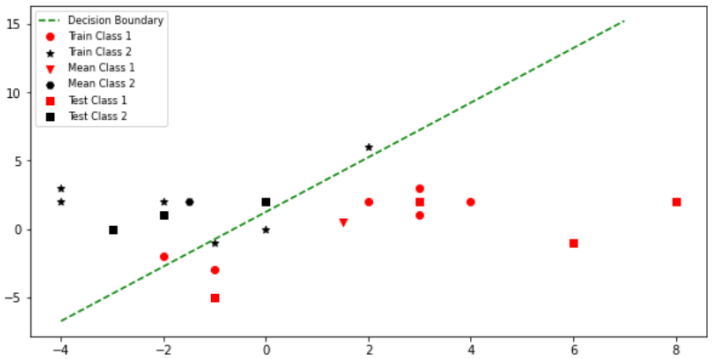
- Firstly, I plot the samples from training dataset with different color and marker with the help of legend.



- Secondly, I calculate the mean and predicted class using given function 1 and plot it with training,predicted test class.



- Thirdly, I calculate the decision boundary and plot it with meanClass,trainClass and testClass Values.



- Finally, I calculate the and get accuracy of 85.71428571428571 % from predicted class and given class.

Conclusion

Here in the given model some data is properly classified and some are not.For this the accuracy is 85.714%.This classifier has a small computation cost but as it has a linear decision boundary, some times it misclassified the given data.