Designing a Minimum Distance to Class Mean Classifier

Anas Sikder
Department of CSE
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204021@aust.edu

***Abstract*—"Minimum Distance to Class Mean Classifier" is used to classify unclassified sample vectors where the vectors clustered in more than one class are given. For example, in a dataset containing n sample vectors of dimension d, some given sample vectors are already clustered into classes and some are not. We can classify the unclassified sample vectors with Class Mean Classifier. It is a simple classifier but works fine. In the experiment I did I found 85% accuracy and among seven samples six are classified and one misclassified.**

***Index Terms*—Linear Discriminant Function, Euclidian distance,class mean,train class ,test class,classfier,classified ,unclassified.**

I. Introduction

A. Motivation

The motivation behind this experiment is to classify the test dataset using the Minimum Distance to Class Mean Classifier algorithm. Also to make an analysis of how accurate the algorithm performs.

B. Theory

For designing the classifier firstly find the mean of ω_1 class and ω_2 class and calling it as μ_1 and μ_2 respectively.Let assumme a new data point x comes and have to assign it one of ω_1 and ω_2 class.So need to calculate the Euclidian distance between μ_1 and x as $D_1(x)$ and μ_2 and x as $D_2(x)$.

If $D_1(x) < D_2(x)$,x $\in \omega_1$ class and if not x $\in \omega_2$ class.Now,we can write is as:

$$-D_1(x) > -D_2(x), x \in \omega_1$$

$$-D_1^2(x) > -D_2^2(x), x \in \omega_1$$

which is our decision boundary.After applying some mathematical transformation the distance from the mean of each class using Linear Discriminant Function:

$$g(x) = \mu^T x - \frac{1}{2}\mu^T \mu$$

where

$$g(x) = -\frac{1}{2}D_i^2(x)$$

II. Experimental Design / Methodology

- *T*ask 01: Firstly I plotted all sample points (train data) from both classes, also make sure that samples from the same class have the same color and marker. For plotting the train data if weight is 1 then I move them into a sub train class 1 and if weight is 2 then I move them into sub train class 2.
- *T*ask 02: Then using a minimum distance classifier with respect to 'class mean', classify the test data points by plotting them with the different class-color with different markers using the Linear Discriminant Function and plotted the class means.
- *T*ask 03:After that, the decision boundary between the two classes has been drawn.Getting the decision boundary I used minimum and maximum values of x to calculate y using the following function for the approximate boundary between the two classes:

$$y = g_2(x) - g_1(x)$$

$$y = -((\mu_1^T - \mu_2^T) - \frac{1}{2}(\mu_1^T \mu_1 - \mu_2^T \mu_2))$$

- *T*ask 04:Finally I found the accuracy.Equation for finding the accuracy is:

$$accuracy = \frac{number\ of\ correct\ class\ count}{length\ of\ test\ dataset} * 100$$

III. Result Analysis

The results of the four tasks are given in sequence of the question.

A. Plotting of sample points of training data set

B. Plotting of the test data set and means of the training data set and the test data set according to their predicted classes.

C. Plotting of the decision boundary

D. Accuracy

Accuracy of the model using given class and predicted class. Accuracy is 85.714 %.

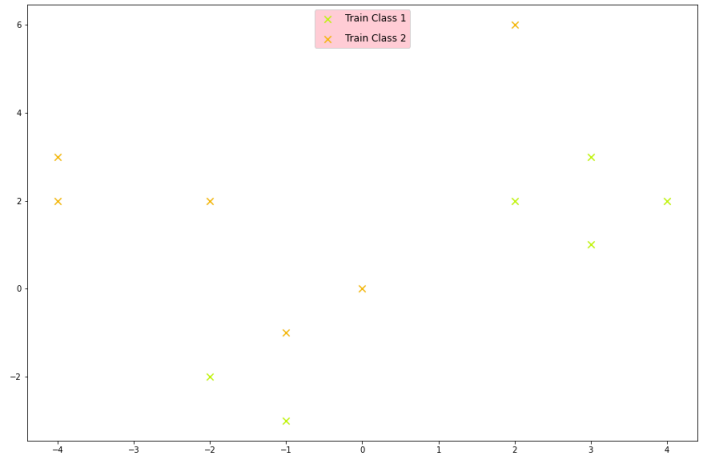

Fig. 1. Train Set

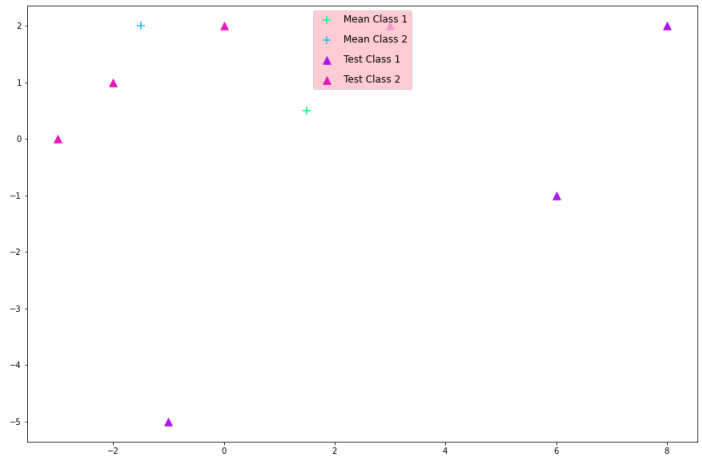

Fig. 2. Test Set And Mean Set

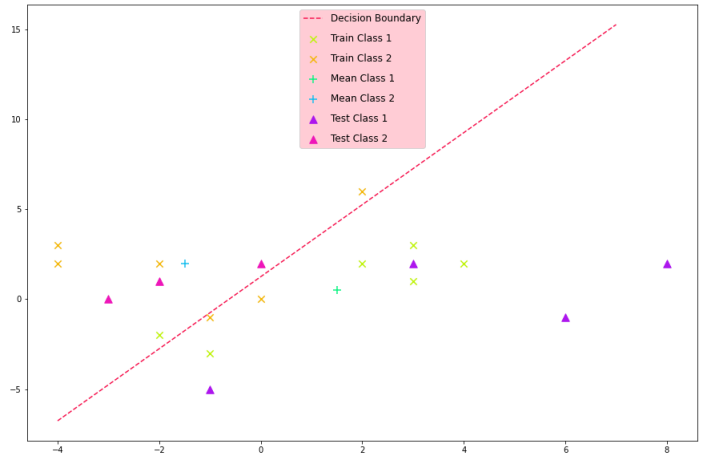

Fig. 3. Decision Boundary

IV. Conclusion

In our experiment, the train set was classified, and by using a Minimum Distance to Class Mean Classifier test dataset had classified with an accuracy of 85.71%. But I did the task on only seven tests set with a misclassification rate of 14.29%. So when the dataset will increases the rate of misclassification will be higher.

V. Algorithm Implementation / Code

In this section snapshot of the codes are given:

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

```
traindataset=pd.read_csv('train.txt',sep=" ",header=None)
train=traindataset.to_numpy();
#print(train)
testdataset=pd.read_csv('test.txt',sep=" ",header=None)
#print(testdataset)
test=testdataset.to_numpy();
```

```
Trainclass1 =[([i[0],i[1]]) for i in train if i[2] == 1]
Trainclass2 =[([i[0],i[1]]) for i in train if i[2] == 2]
Trainclass1 = np.array(Trainclass1)
Trainclass2 = np.array(Trainclass2)
#print('TrainClass 1: ',Trainclass1)
#print('TrainClass 2: ',Trainclass2)
```

```
testVal =[(i[0],i[1]) for i in test]
testClass = [i[2] for i in test]

testVal=np.array(testVal)
testClass=np.array(testClass)

#print('Test Values: ',testVal)
#print('Test class',testClass)
```

Fig. 4. Code Snap 1

```
sum_X_Class1 = 0
sum_Y_Class1 = 0

for i in Trainclass1:
    sum_X_Class1,sum_Y_Class1 = (sum_X_Class1 + i[0]),(sum_Y_Class1 + i[1])

#print(sum_X_Class1)
#print(sum_Y_Class1)
meanclass1 = [sum_X_Class1/len(Trainclass1), sum_Y_Class1/len(Trainclass1)]
meanclass1=np.array(meanclass1)
sum_X_Class2 = 0
sum_Y_Class2 = 0
for i in Trainclass2:
    sum_X_Class2,sum_Y_Class2 = (sum_X_Class2 + i[0]),(sum_Y_Class2 + i[1])

meanclass2 = [sum_X_Class2/len(Trainclass2), sum_Y_Class2/len(Trainclass2)]
meanclass2=np.array(meanclass2)
#print(meanclass2)
```

```
predtn1 = np.zeros(7)
predtn2 = np.zeros(7)
predclass = np.zeros(7)
temp = np.zeros((7,3))
#print(predtn1,predtn2,predclass,temp)

for i in range(len(testVal)):
    predtn1[i]=np.dot(meanclass1.T, testVal[i]) - 0.5*np.dot(meanclass1.T,meanclass1)
    #print(predtn1)
    predtn2[i]=np.dot(meanclass2.T,testVal[i]) - 0.5*np.dot(meanclass2.T,meanclass2)
    #print(predtn2)
    if(predtn1[i]>predtn2[i]):
        temp[i][0],temp[i][1],temp[i][2],predclass[i] = testVal[i][0],testVal[i][1],1,1
        #print(temp[i])

    else:
        temp[i][0],temp[i][1],temp[i][2],predclass[i] = testVal[i][0],testVal[i][1],2,2
```

Fig. 5. Code Snap 2

```
class1=[[i[0],i[1]] for i in temp if i[2]==1]
class2=[[i[0],i[1]] for i in temp if i[2]==2]

class1=np.array(class1)
class2=np.array(class2)
#print(class1,class2)
```

```
X = np.zeros(12)
Y = np.zeros(12)
point = 0
for x in range(-4,8,1):
    y = -((meanclass1[0]-meanclass2[0])*x - 0.5 *
          np.dot(meanclass1.T,meanclass1)+0.5*np.dot(meanclass2.T,meanclass2))/(meanclass1[1]-meanclass2[1])
    #print (y)
    X[point] = x
    Y[point] = y
    point= point + 1
#print(X,Y)
fig, axis = plt.subplots()
fig.set_figheight(10)
fig.set_figwidth(15)

#decision boundary ploting
axis.plot(X,Y,"--",label='Decision Boundary',color='#F81852')

#scatting train values
axis.scatter(Trainclass1[:,0],Trainclass1[:,1],marker='x',color='#C0F40D',s=70,label='Train Class 1')
axis.scatter(Trainclass2[:,0],Trainclass2[:,1],marker='x',color='#F4B80D',s=70,label='Train Class 2')
```

Fig. 6. Code Snap 3

```
#scatting mean values
axis.scatter(meanclass1[0],meanclass1[1],marker='+',color='#0DF483', s=90, label='Mean Class 1')
axis.scatter(meanclass2[0],meanclass2[1],marker='+',color='#16BFEE',s=90,label='Mean Class 2')

#scatting test values
axis.scatter(class1[:,0],class1[:,1],marker='^',color='#AD16EE',s=90,label='Test Class 1')
axis.scatter(class2[:,0],class2[:,1],marker='^',color='#EE16B9',s=90,label='Test Class 2')

#labeling decoration
legend = axis.legend(loc='upper center',fontsize='large',labelspacing=1.0)
legend.get_frame().set_facecolor('pink')
```

```
#Accuracy
acc = 0
acc = [acc+1 for m,n in zip(testClass,predclass) if m == n]
accuracy = sum(acc)/len(testClass) *100
print(format(accuracy,'.2f'),'%')
```

Fig. 7. Code Snap 4