Designing a Minimum Distance to Class Mean Classifier

Md. Zahid Fesabelilla
Computer Science and Engineering. Ahsanullah University of Science and Technology
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204082@aust.edu

***Abstract*—This document is about Designing a Minimum Distance to Class Mean Classifier. Here, I draw all sample points (train data) from both classes, plot both class means and draw the decision boundary between the two classes and find accuracy using matplotlib,numpy, and some math function.**

***Index Terms*—minimum distance, mean value, classifier**

I. Introduction

The minimum distance classifier is used to classify unknown data to which minimize the distance between the data and the class in multi-feature space [google]. It's like supervised learning because all data are labeled. Here, we have two data set. One is the train data set and the rest is the test data set. We train the machine using train data set where all data are correctly classified and then test the system (classify data, find accuracy) using this test data set to help a class mean classifier.

II. Experimental Design / Methodology

Linear Discriminant Function

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

Firstly, Plot all sample points (train data) from both classes, samples from the same class have the same color and marker. Then, plot all test data and train data using different markers.

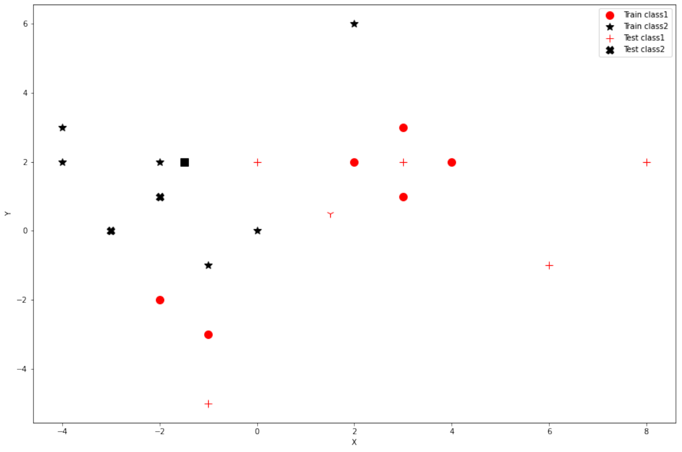


Fig. 1. Identify Train Data and Test Data

Secondly, Calculate two mean points for train class 1 and class 2. Identify these two mean points using two different symbols given below.

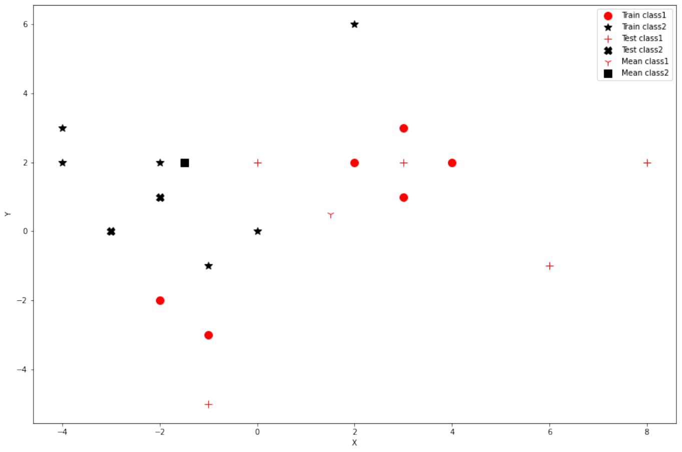


Fig. 2. Mean points for train class 1 and class 2

Thirdly, Drawing the decision boundary between the two classes.

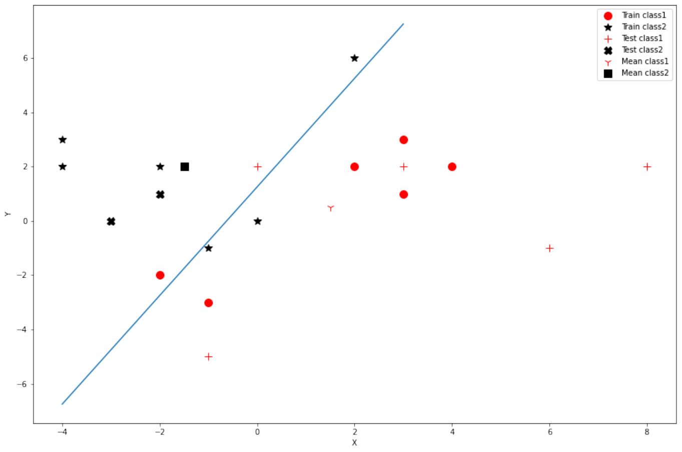


Fig. 3. Decision Boundary

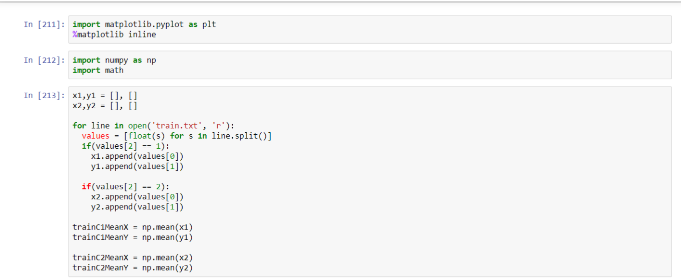
III. Result Analysis

For this classifier I analyze two sets of data , the train data set contains 12 data, and the test data set contains 7 data. Here, only one data is misclassified and this system's accuracy is 85.714%.

IV. Conclusion

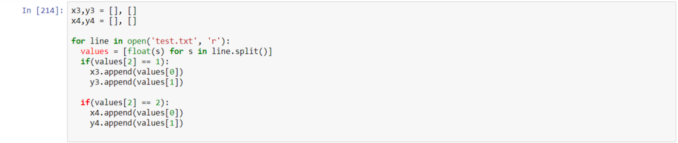
For the small amount of linear data, this classifier performs well, and the accuracy is good enough. But this algorithm has one weakness that is its misclassification rate is higher because the boundary between the two classes is linear.

V. Algorithm Implementation / Code



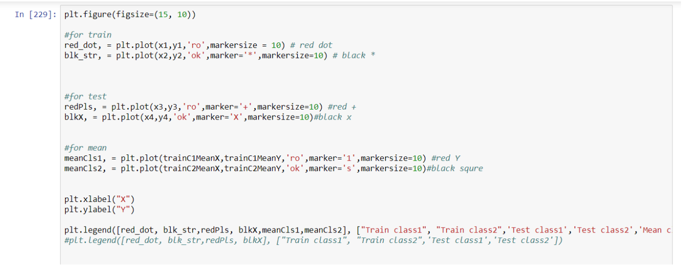
```
In [211]: import matplotlib.pyplot as plt
          %matplotlib inline

In [212]: import numpy as np
          import math

In [213]: x1,y1 = [], []
          x2,y2 = [], []

          for line in open('train.txt', 'r'):
            values = [float(s) for s in line.split()]
            if(values[2] == 1):
              x1.append(values[0])
              y1.append(values[1])

            if(values[2] == 2):
              x2.append(values[0])
              y2.append(values[1])

          trainC1MeanX = np.mean(x1)
          trainC1MeanY = np.mean(y1)

          trainC2MeanX = np.mean(x2)
          trainC2MeanY = np.mean(y2)
```

Fig. 4. Identify Train Data,and Calculate Mean Points



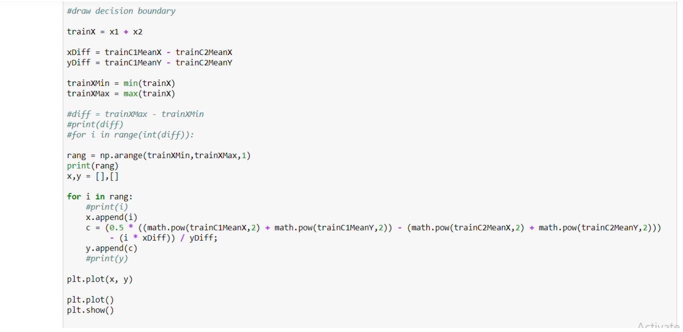
```
In [214]: x3,y3 = [], []
          x4,y4 = [], []

          for line in open('test.txt', 'r'):
            values = [float(s) for s in line.split()]
            if(values[2] == 1):
              x3.append(values[0])
              y3.append(values[1])

            if(values[2] == 2):
              x4.append(values[0])
              y4.append(values[1])
```

Fig. 5. Identify Test Data



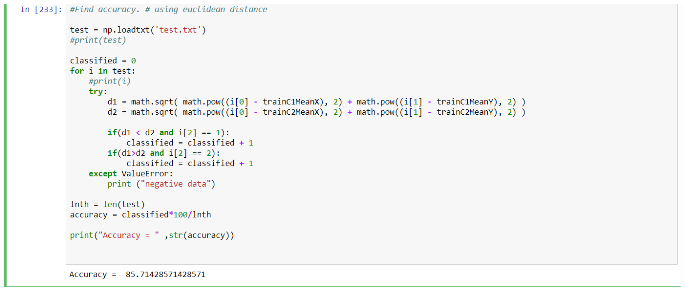
```
In [229]: plt.figure(figsize=(15, 10))

          #for train
          red_dot, = plt.plot(x1,y1,'ro',markersize = 10) # red dot
          blk_str, = plt.plot(x2,y2,'ok',marker='*',markersize=10) # black *



          #for test
          redPls, = plt.plot(x3,y3,'ro',marker='+',markersize=10) #red +
          blkX, = plt.plot(x4,y4,'ok',marker='X',markersize=10)#black x


          #for mean
          meanCls1, = plt.plot(trainC1MeanX,trainC1MeanY,'ro',marker='1',markersize=10) #red Y
          meanCls2, = plt.plot(trainC2MeanX,trainC2MeanY,'ok',marker='s',markersize=10)#black squre


          plt.xlabel("X")
          plt.ylabel("Y")

          plt.legend([red_dot, blk_str,redPls, blkX,meanCls1,meanCls2], ["Train class1", "Train class2",'Test class1','Test class2','Mean c
          #plt.legend([red_dot, blk_str,redPls, blkX], ["Train class1", "Train class2",'Test class1','Test class2'])
```

Fig. 6. Plot all data



```
#draw decision boundary

trainX = x1 + x2

xDiff = trainC1MeanX - trainC2MeanX
yDiff = trainC1MeanY - trainC2MeanY

trainXMin = min(trainX)
trainXMax = max(trainX)

#diff = trainXMax - trainXMin
#print(diff)
#for i in range(int(diff)):

rang = np.arange(trainXMin,trainXMax,1)
print(rang)
x,y = [],[]

for i in rang:
    #print(i)
    x.append(i)
    c = (0.5 * ((math.pow(trainC1MeanX,2) + math.pow(trainC1MeanY,2)) - (math.pow(trainC2MeanX,2) + math.pow(trainC2MeanY,2)))
        - (i * xDiff)) / yDiff;
    y.append(c)
    #print(y)

plt.plot(x, y)

plt.plot()
plt.show()
```

Fig. 7. Decision Boundary



```
In [233]: #Find accuracy. # using euclidean distance

          test = np.loadtxt('test.txt')
          #print(test)

          classified = 0
          for i in test:
              #print(i)
              try:
                  d1 = math.sqrt( math.pow((i[0] - trainC1MeanX), 2) + math.pow((i[1] - trainC1MeanY), 2) )
                  d2 = math.sqrt( math.pow((i[0] - trainC2MeanX), 2) + math.pow((i[1] - trainC2MeanY), 2) )

                  if(d1 < d2 and i[2] == 1):
                      classified = classified + 1
                  if(d1>d2 and i[2] == 2):
                      classified = classified + 1
              except ValueError:
                  print ("negative data")

          lnth = len(test)
          accuracy = classified*100/lnth

          print("Accuracy = " ,str(accuracy))

          Accuracy =  85.71428571428571
```

Fig. 8. Accuracy