Designing a Minimum Distance to Class Mean Classifier

Md. Yeasir Arafat
Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204093@aust.edu

***Abstract*—We classified a point using Minimum Distance to Class Mean Classifier. And the model is 85.71% accurate.**

***Index Terms*—This classifier uses average value of two classes and measures distance from two average value to the unclassified point. Which distance is smaller it belongs to that class.**

I. Introduction

"Minimum Distance to Class Mean Classifier" is used to classify unclassified sample vectors where the vectors clustered in more than one classes 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 .

II. Experimental Design / Methodology

1. **Plotting all sample points:** Two sample classes are given to 'train.txt' file. We need to plot all the sample points of both classes using different markers. First I read the file from google drive then extract the data for plotting. Finally I plotted the points using matplotlib package.

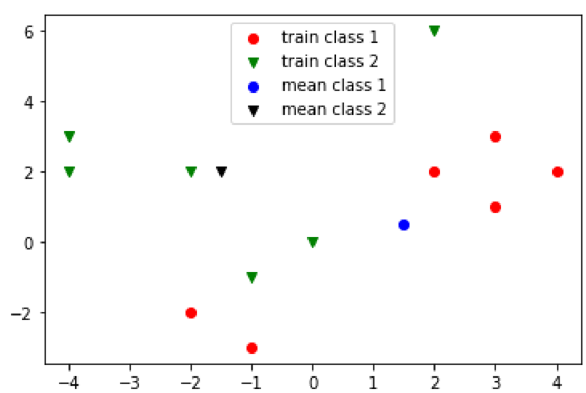

Fig 1: Sample points of Train Data

2. **Classifying Test Data:** Classifying Test Data points and plotting them using same marker for same class and also plot Mean classes with different marker.

We will use Linear Discriminant function to calculate distance from Mean class to each point.

Linear Discriminant Function:

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

if $g_i(X) > g_j(X)$ then X belongs to Class i
else X belongs to Class j

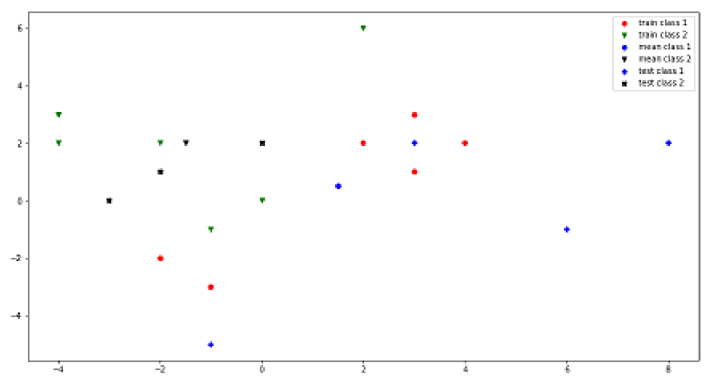

Fig 2: Classified points of Test Dataset and Mean class

3. **Drawing decision boundary:** Decision Boundary between two classes need to be drawn presently. For finding out the decision boundary I considered all possible points whose distance from both the classes is same. That means g1(X) = g2(X) at the decision boundary.

$$\begin{aligned} g(X) &= g_1(X) - g_2(X) \\ &= \bar{W_1}^T X - \frac{1}{2}\bar{W_1}^T\bar{W_1} - \bar{W_2}^T X + \frac{1}{2}\bar{W_2}^T\bar{W_2} \\ &= X(\bar{W_1}^T - \bar{W_2}^T) - \frac{1}{2}(\bar{W_1}^T\bar{W_1} - \bar{W_2}^T\bar{W_2}) \end{aligned} \tag{2}$$

From this formula I derived the linear equation to find the decision boundary. The equation is:

$$
\begin{aligned}
&X(\bar{W_1^T} - \bar{W_2^T}) - \frac{1}{2}(\bar{W_1^T}\bar{W_1} - \bar{W_2^T}\bar{W_2}) = 0\\
&\Rightarrow \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \begin{pmatrix} COEF_1 & COEF_2 \end{pmatrix} + C = 0\\
&\Rightarrow X_1 \times COEF_1 + X_2 \times COEF_2 + C = 0\\
&\Rightarrow X_2 = \frac{X_1 \times COEF_1 + C}{-COEF_1}
\end{aligned} \tag{3}
$$

$$[Here, C = -\frac{1}{2}(\bar{W_1^T}\bar{W_1} - \bar{W_2^T}\bar{W_2})]$$

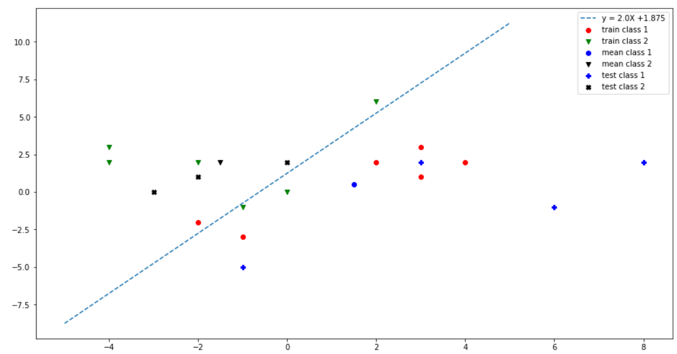

Fig 3: '–' line shows the decision boundary

4. **Accuracy:** we can calculate accuracy using this function:

$$
\begin{aligned}
Accuracy &= \frac{n \times 100}{N}\\
here, n &= number\ of\ accurate\ result,\\
N &= Total\ number\ of\ input
\end{aligned} \tag{4}
$$

III. Result Analysis

In our experiment we use 1 dataset which has 7 different data. In which 6 are successfully classified using this classifier and 1 is not classified correctly. So the accuracy is 85.71%.

IV. Conclusion

In this classifier some sample are classified and some are misclassified. This classifier has simple calculation so that, the speed of the result is very efficient but the main weakness of this algorithm is it's misclassification rate is higher because of the boundary between the two classes is linear.

V. Algorithm Implementation / Code

Read file from gdrive

```
from google.colab import drive
drive.mount('/content/gdrive')
# change working directory on the drive
%cd '/content/gdrive/My Drive/Data/'

# read train.txt file line by line
with open('train.txt', "r") as train:
    FileasList = train.readlines()

# split the string and store it into anothe
train = []
train_x1 = []
train_y1 = []
train_x2 = []
train_y2 = []
for i in range(len(FileasList)):
  train.append(FileasList[i].split())
  if(train[i][2] == '1'):
    train_x1.append(int(train[i][0]))
    train_y1.append(int(train[i][1]))
  else:
    train_x2.append(int(train[i][0]))
    train_y2.append(int(train[i][1]))
```

Fig 4.1: Read train.txt file from gdive

```
# read test.txt file line by line
with open('test.txt', "r") as test:
    FileasList = test.readlines()

# split the string and store it into anothe
test = []
test_x = []
test_y = []
test_class = []
for i in range(len(FileasList)):
  test.append(FileasList[i].split())
  test_x.append(int(test[i][0]))
  test_y.append(int(test[i][1]))
  test_class.append(int(test[i][2]))
```

Fig 4.2: Read test.txt file from gdive

Plotting all sample points from train data and mean class

```
import matplotlib.pyplot as plt

plt.scatter(train_x1, train_y1, c = 'r', marker = 'o', label = 'tra
plt.scatter(train_x2, train_y2, c = 'g', marker = 'v', label = 'tra
plt.legend(loc = 'best')


# Mean of class 1 and class 2
def average(lst):
  return sum(lst) / len(lst)
x1_mean = average(train_x1)
y1_mean = average(train_y1)
x2_mean = average(train_x2)
y2_mean = average(train_y2)

plt.scatter(x1_mean, y1_mean, c = 'b', marker = 'o', label = 'mean 
plt.scatter(x2_mean, y2_mean, c = 'black', marker = 'v', label = 'm
plt.legend(loc = 'upper center')
```

Fig 4.3: Plotting Train data and mean class

Plotting Decision Boundary

```
c = .5 * ( np.dot(np.transpose(m1), m1) - ( np.dot(np.transpose(m2), m2)  )
x = np.transpose(m1) - np.transpose(m2)
z = np.linspace(-5,5,10)
y = (x[0]*z - c)/(x[1] * (-1))

# equation
w0 = ""
if(c<0):
  w0 = "x +"
else:
  w0 = "x -"
eqn = "y = "  + str((x[0]/(-1 * x[1]))) + w0 + str(abs(c))


plt.figure(figsize=(15,8))

# Decision Boundary
plt.plot(z, y, '--', label = eqn)
plt.legend(loc = 'upper right')
```

Fig 4.6: Plotting Decision Boundary

Classifying Test Data Points

```
import numpy as np
m1 = np.array([x1_mean, y1_mean])
m2 = np.array([x2_mean, y2_mean])
test_classify = []

for i in range(len(test_x)):
  x = np.array([test_x[i],test_y[i]])
  g1 =  np.dot(np.transpose(m1), x) - (.5 * ( np.dot(np.transpose(r
  g2 =  np.dot(np.transpose(m2), x) - (.5 * ( np.dot(np.transpose(r

  if(g1 > g2):
    test_classify.append(1)
  else:
    test_classify.append(2)
```

Fig 4.4: Classify Test Dataset

Accuracy of our model

```
n = len(test_class)
rc = 0
for i in range(n):
  if(test_class[i] == test_classify[i]):
    rc += 1
ac = (rc * 100)/n
ac = formatted_float = "{:.2f}".format(ac)
print("Accuracy of our model is: " + str(ac)+"%")
```

Fig 4.7: Accuracy

Plotting Test Data

```
test_x1 = []
test_y1 = []
test_x2 = []
test_y2 = []
for i in range(len(test_classify)):
  if(test_classify[i] == 1):
    test_x1.append(test_x[i])
    test_y1.append(test_y[i])
  else:
    test_x2.append(test_x[i])
    test_y2.append(test_y[i])

plt.figure(figsize=(15,8))
# train class
plt.scatter(train_x1, train_y1, c = 'r', marker = 'o', label = 'train class 1')
plt.scatter(train_x2, train_y2, c = 'g', marker = 'v', label = 'train class 2')

# mean class
plt.scatter(x1_mean, y1_mean, c = 'b', marker = 'o', label = 'mean class 1')
plt.scatter(x2_mean, y2_mean, c = 'black', marker = 'v', label = 'mean class 2')

# test class
plt.scatter(test_x1, test_y1, c = 'b', marker = 'P', label = 'test class 1')
plt.scatter(test_x2, test_y2, c = 'black', marker = 'X', label = 'test class 2')
plt.legend(loc = 'upper right')
```

Fig 4.5: Plotting test data