Designing a Minimum Distance to Class Mean Classifier

Ahmad Subaktagin Jabir
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204061@aust.edu

***Abstract*—The main goal of this experiment is to determine how this model works. This is a simple classifier. Although this classifier doesn't work correctly in all situations the main purpose was to know how minimum distance to class mean classifier works. I have done some tasks to complete this experiment, such as- training the dataset, classifying with the help of linear discriminant function, drawing the decision boundary etc. Also this model provides a good accuracy in classifying the unknown data.**

***Index Terms*—Mean Class, Linear Discriminant Function, Training Data, Testing Data, Decision Boundary, Accuracy.**

I. Introduction

Minimum Distance to Class Mean Classifier means classifying unclassified sample vectors where the vectors are clustered in more than one class. In this classifier, the mean point of sample classes are calculated at first. Then sample points of unknown class are assigned to the class which has minimum distance with the mean class.

We are using Following equation to determine Linear Discriminant Function:

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

Here, X is a feature vector and $\bar{Y}$ denotes mean of an individual class. So according to the given equation, we have to calculate the mean of every class and after that we can find the minimum distance to a class mean by using this rule.

Decision Boundary is a hypersurface that partitions the underlying vector space into two sets, one for each class. It separates one class from another.

II. Experimental Design / Methodology

We have to do our experiment by using two datasets named 'Training Dataset' and 'Testing Dataset'. Following tasks have to be done in order to design a minimum distance to class mean classifier:

1) **Plotting the Training Dataset:** There are two classes given in both datasets. "Training Dataset" is mainly used here. Same color and the same marker is used here to distinguish one class from another. Two Numpy arrays are used here to store the data from the dataset and plot them with different markers.
2) **Classifier testing by using given test points:** The mean points of the two classes are calculated and plotted here. Like task 1, the same color and same marker is used to distinguish one class from another. Here, following equation was used for two classes:

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

So, there were two functions named g_1 and g_2. If g_1 is greater than g_2 then the point is in class 1. Otherwise, the point is in class 2. Numpy arrays are used here to plot all points.
3) **Defining and Drawing Decision Boundary:** If we want to draw a decision boundary, we have to draw a line which has almost equal distance from both the class mean. So,if we consider 2 class i and j, we can say

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

Now putting the values of both function with general Linear Discriminant function, we get

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

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

Here, ω_i and ω_j denotes mean of class i and j. From the obtained equation, we can see that it is in the form of $mx + c = 0$ which is a line equation. By comparing both equation, we can say that $c = -\frac{1}{2}(\omega_i^T\omega_i - \omega_j^T\omega_j)$ and $m = (\omega_i^T - \omega_j^T)$. If we take the lowest and highest value of x and then define a range between these two values then we will get a set of x values. By putting these values we can get set of y values,

$$y = -\frac{(\omega_{ix} - \omega_{jx})x + \frac{1}{2}(\omega_i^T\omega_i - \omega_j^T\omega_j)}{(\omega_{iy} - \omega_{jy})}$$

Then we can plot these x and y values. After plotting them we can get the decision boundary which will divide the feature space with two regions.
4) **Accuracy Calculation:** Calculation of Accuracy is done by using correct classification and total classification.

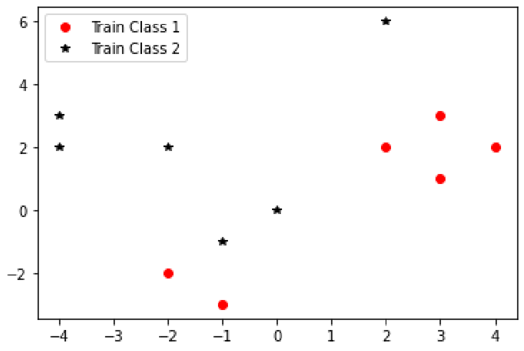
Here correct classification means number of points which are correctly classified and total classification means number of total test classes. Following equation is used here for the calculation:

$$Accuracy = \frac{correct\ classification}{total\ classification} * 100$$

III. Result Analysis

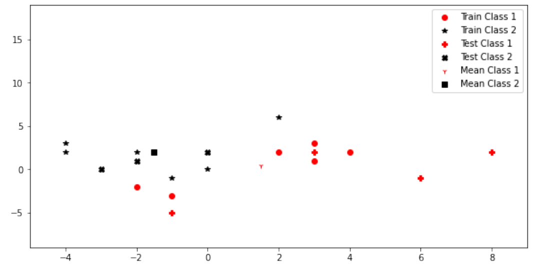
After completing all the tasks, results are given below:

1) Plotting the Training Dataset



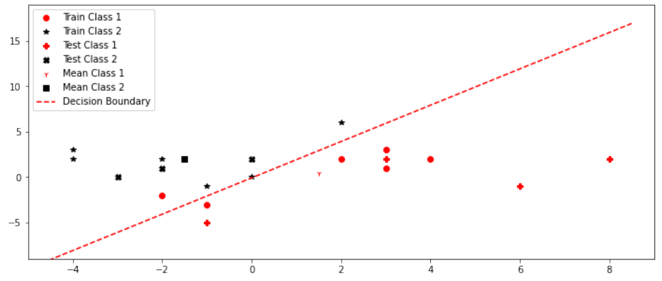
Here, I used Red Circle to denote the Train Class 1 and Black Star to denote the Train Class 2.

2) Plotting Class Means



Here, I used Red circle to denote the Train Class 1, Black Star to denote the Train Class 2, Red Plus to denote the Test Class 1, Black Cross to denote the Test Class 2, Red Tri Down to denote the Mean Class 1 and Black Square to denote the Mean Class 2.

3) Drawing the Decision Boundary



Here, I used the Red '- -' line to denote the Decision Boundary between two classes.

4) Accuracy Calculation

```
Accuracy is: 85.71428571428571 %
```

After Calculation I got 85.71% Accuracy.

IV. Conclusion

From this experiment by using the given dataset there are some classified samples and some misclassified samples. Accuracy is good in the minimum distance to mean classifier. Also there are some demerits of the minimum distance to mean classifier. One of them is misclassifying some samples as the decision boundary between two classes is linear. Also the given dataset is not sufficient here to determine its impact. The model will be stronger if more data is used here.

V. Algorithm Implementation / Code

```
import numpy as np
import matplotlib.pyplot as plt
import csv

## Task 1
train_set = []
with open('train.txt','r') as file:
    new_reader = csv.reader(file,delimiter=' ')
    for row in new_reader:
        train_set.append(row)

for i in range(len(train_set)):
    for j in range(len(train_set[i])):
        train_set[i][j] = int(train_set[i][j])


## Class A and Class B fixing
a = []
b = []
for train in train_set:
    if train[2]==1:
        a.append([train[0], train[1]])
    elif train[2]==2:
        b.append([train[0], train[1]])
class_a = np.array(a)
class_b = np.array(b)

## Displaying the Plot
plt.plot(class_a[:,0:1],class_a[:, 1:], linestyle =
    '', marker='o', color='r', label ="Train Class 1
    ")
plt.plot(class_b[:,0:1],class_b[:, 1:], linestyle =
    '', marker = '*', color = 'k', label ="Train
    Class 2")
plt.legend()
plt.show()


## Task 2

## Importing Test Set File
test_set =[]
with open('test.txt','r') as file:
    new_reader = csv.reader(file,delimiter=' ')
    for row in new_reader:
        test_set.append(row)

for i in range(len(test_set)):
    for j in range(len(test_set[i])):
        test_set[i][j] = int(test_set[i][j])
test_set_new = [[t[0],t[1]] for t in test_set]
test_set_array = np.array(test_set_new)

## Determining Class
classification = []
mean_class_a = np.array([0,0], dtype=np.float64)
mean_class_b = np.array([0,0], dtype=np.float64)

def find_class(weight = np.array([], dtype=np.
    float64)):
    ld_1 = (np.matmul(weight.transpose(),
    mean_class_a))+0.5*(np.matmul(mean_class_a.
    transpose(),mean_class_a))
    ld_2 = (np.matmul(weight.transpose(),
    mean_class_b))+0.5*(np.matmul(mean_class_b.
    transpose(),mean_class_b))

    if(ld_1 >=ld_2):
        classification.extend([1])
        return 1
    else:
        classification.extend([2])
        return 2

mean_class_a[0] = np.mean(class_a[:,0:1], dtype=np.
    float64)
mean_class_a[1] = np.mean(class_a[:,1:], dtype=np.
    float64)
mean_class_b[0] = np.mean(class_b[:,0:1], dtype=np.
    float64)
mean_class_b[1] = np.mean(class_b[:,1:], dtype=np.
    float64)

test_a = []
test_b = []
for i in range(len(test_set_array)):
    if(find_class(test_set_array[i])==1):
        test_a.append([test_set_array[i][0],
    test_set_array[i][1]])
    else:
        test_b.append([test_set_array[i][0],
    test_set_array[i][1]])

final_test_a = np.array(test_a)
final_test_b = np.array(test_b)

## Plotting of the Means of Training Dataset,
    Training Dataset, Test Dataset
plt.xlim(-5,9)
plt.ylim(-9, 19)
plt.rcParams["figure.figsize"] = (10,4)
plt.plot(class_a[:,0:1],class_a[:, 1:], linestyle =
    '', marker='o', color='r', label ="Train Class 1
    ")
plt.plot(class_b[:,0:1],class_b[:, 1:], linestyle =
    '', marker = '*', color = 'k', label ="Train
    Class 2")
plt.plot(final_test_a[:,0:1],final_test_a[:, 1:],
    linestyle = '', marker='P', color='r', label ="
    Test Class 1")
plt.plot(final_test_b[:,0:1],final_test_b[:, 1:],
    linestyle = '', marker = 'X', color = 'k', label
     ="Test Class 2")
plt.plot(mean_class_a[0],mean_class_a[1], linestyle
    = '', marker='1', color='r', label ="Mean Class
    1")
plt.plot(mean_class_b[0],mean_class_b[1], linestyle
    = '', marker = 's', color = 'k', label ="Mean
    Class 2")
plt.legend()
plt.show()

## Task 3

mean_1 = np.array([[mean_class_a[0]], [mean_class_b
    [0]]])
mean_2 = np.array([[mean_class_a[1]], [mean_class_b
    [1]]])

## Determining Line Equation
def line_equation(x):
    y = (mean_class_a[0]-mean_class_b[0])*x-0.5*(np.
    dot(np.transpose(mean_1), mean_1))+0.5*(np.dot(
    np.transpose(mean_2), mean_2))
    z = y/(mean_class_a[1]-mean_class_b[1])
    return -z[0][0]

x_values = []
y_values = []
for i in np.arange(-5, 9, 0.5):
    x_values.append(i)
    y_values.append(line_equation(i))

## Plotting all points and Decision Boundary
plt.xlim(-5,9)
plt.ylim(-9, 19)
plt.rcParams["figure.figsize"] = (20,5)
plt.plot(class_a[:,0:1],class_a[:, 1:], linestyle =
    '', marker='o', color='r', label ="Train Class 1
    ")
plt.plot(class_b[:,0:1],class_b[:, 1:], linestyle =
    '', marker = '*', color = 'k', label ="Train
    Class 2")
plt.plot(final_test_a[:,0:1],final_test_a[:, 1:],
    linestyle = '', marker='P', color='r', label ="
    Test Class 1")
plt.plot(final_test_b[:,0:1],final_test_b[:, 1:],
    linestyle = '', marker = 'X', color = 'k', label
     ="Test Class 2")
plt.plot(mean_class_a[0],mean_class_a[1], linestyle
    = '', marker='1', color='r', label ="Mean Class
    1")
plt.plot(mean_class_b[0],mean_class_b[1], linestyle
    = '', marker = 's', color = 'k', label ="Mean
    Class 2")
plt.plot(x_values, y_values, linestyle='--', color='
    r', label="Decision Boundary")
plt.legend()
plt.show()

## Task 4

undivided_test_set_array = np.array(test_set)
extract_class = []
for t in undivided_test_set_array:
    extract_class.append(t[2])
cnt = 0
for i, j in zip(extract_class, classification):
    if i==j:
        cnt=cnt+1
print("Accuracy is:", cnt/len(extract_class)*100,"%"
    )
```