Designing a Minimum Distance to Class Mean Classifier

Mayeesha Humaira
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204008@aust.edu
section - A1

***Abstract*—The objective of this assignment is to design a minimum distance to class mean classifier.**

***Index Terms*—decision boundary, class mean, linear discriminant function**

I. Introduction

Minimum distance to the class mean classifier is used to classify an unknown point using the linear discriminant function. When there is more than one class of data clusters minimum mean distance of each class is used to classify unknown points.

II. Experimental Design / Methodology

Task:
Two-class set of prototypes have to be taken from "train.txt" and "test.txt" files.

1) Plot all sample points (train data) from both classes, but samples from the same class should have the same color and marker.
2) Using a minimum distance classifier with respect to 'class mean', classify the test data points by plotting them with the designated class-color but a different marker. Use the Linear Discriminant Function given below. Also, plot the class means.

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

3) Draw the decision boundary between the two classes.
4) Find accuracy.

Solution:

1) task-1 Plotting Training Data:
Given two-class set of prototypes in a file train.txt as Shown below:

2 2 1
3 1 1
-4 3 2
3 3 1
-1 -3 1
2 6 2
4 2 1
-2 -2 1
0 0 2
-2 2 2
-1 -1 2
-4 2 2

The above data are given in the train.txt file where the first column represents the x-coordinates, the second column represents the y-coordinates and the third column represents the class label.
Firstly, I read the data points from the train.txt file. Then using class labels data points were plotted using matplotlib. I employed different markers and colors for different classes. As in the given data there were two classes I used red dot markers to plot all data of class 1 and blue star marker to plot all data of class 2. The image of the plot is shown in Fig. 1.

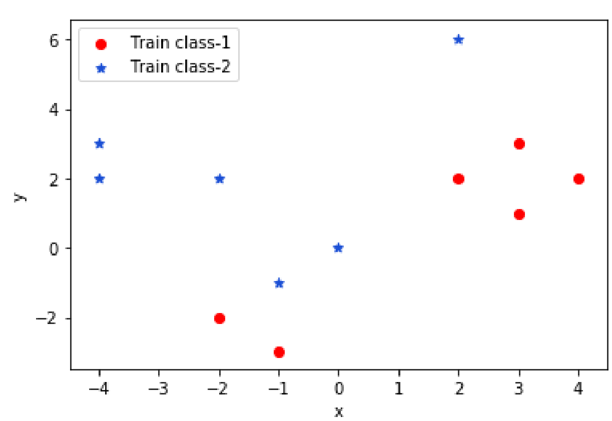

Fig. 1. Graphical representation of training data using matplotlib.

2) task-2 Classifying testing with respect to class mean and plotting testing data and mean:
To classify test data I first computed mean of each classes in training data. The mean of class 1 was [1.5, 0.5] and the mean of class 2 was [-1.5, 2.0]. The plot of mean of two classes along with training data using matplotlib is shown in Fig. 2 where red

triangle represents mean of class 1 and blue square represents mean of class 2. Then using the formula $g_i(X) = X^T\overline{Y_i} - \frac{1}{2}\overline{Y_i}^T\overline{Y_i}$ where $\overline{Y_i}$ is the mean of class i testing samples were classified. The testing samples were given in a test.txt file. The samples of test.txt file were:

-1 -5 1
3 2 1
-2 1 2
8 2 1
6 -1 1
0 2 1
-3 0 2

Only the first and the second column of this file were used as x-coordinate and y-coordinate respectively. If for a testing sample $g_i > g_j$ were i and j represents different classes then that sample belongs to class i else if the testing sample has $g_i < g_j$ were i and j represents different classes then it belongs to class j. After determining

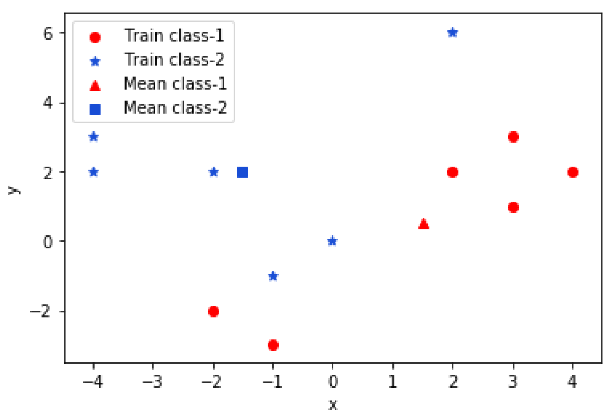

Fig. 2. Graphical representation of training data and mean of each class using matplotlib.

which class each test sample belongs to they were also plotted using matplotlib. Testing sample belonging to class 1 were plotted using red plus marker and samples of class 2 were plotted using blue cross marker. This plot is shown in Fig. 3.

3) task-3 Drawing Decision boundary between two classes: To draw the decision boundary I derived an equation using Linear discriminant function of two classes g_1 and g_2. A point in class-1 is in decision boundary when:

$$g_1 = 0 \tag{2}$$

A point in class-2 is in decision boundary when:

$$g_2 = 0 \tag{3}$$

From Eq. 9 and Eq. 3 it can be said that:

$$g_1 = g_2 \tag{4}$$

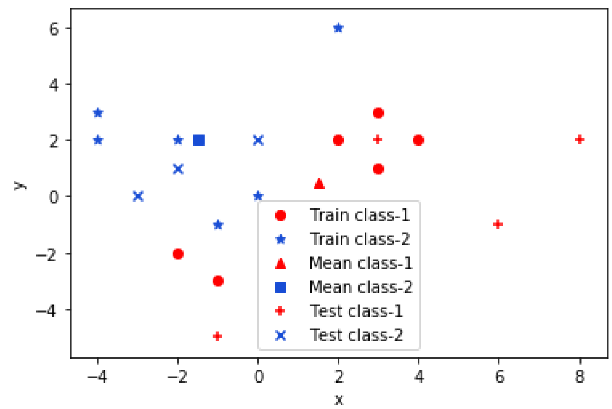

Fig. 3. Graphical representation of training data, mean of each class and classified testing data using matplotlib.

Substituting Eq. 1 in Eq. 4 we get:

$$\overline{\omega}_1^T X - \frac{1}{2}\overline{\omega}_1^T\overline{\omega}_1 = \overline{\omega}_2^T X - \frac{1}{2}\overline{\omega}_2^T\overline{\omega}_2 \tag{5}$$

$$(\overline{\omega}_1^T - \overline{\omega}_2^T)X - \frac{1}{2}\overline{\omega}_1^T\overline{\omega}_1 + \frac{1}{2}\overline{\omega}_2^T\overline{\omega}_2 \tag{6}$$

$$(\overline{\omega}_1^T - \overline{\omega}_2^T)X - \frac{1}{2}(\overline{\omega}_1^T\overline{\omega}_1 - \frac{1}{2}\overline{\omega}_2^T\overline{\omega}_2) = 0 \tag{7}$$

where X in Eq. 7 is:

$$X = \begin{bmatrix} x \\ y \end{bmatrix} \tag{8}$$

Replacing Eq. 8 in Eq. 7 we get:

$$y = \frac{0.5(\overline{\omega}_1^T\overline{\omega}_1 - \overline{\omega}_2^T\overline{\omega}_2) + (\overline{\omega}_{1x}^T - \overline{\omega}_{2x}^T)x}{\overline{\omega}_{1y}^T - \overline{\omega}_{2y}^T} \tag{9}$$

where $\overline{\omega}_i$ is the mean of class i, $\overline{\omega}_{ix}$ is the x component of mean of class i and $\overline{\omega}_{iy}$ is the y component of mean of class i.

Range of x-coordinates in training set was -4 to 4. x-coordinate can be iterated within this range to get the y-coordinates to draw the decision boundary using Eq. 9. I iterated x by 0.3 from range -4 to 4 to get corresponding y values. Training data, classified testing data, mean of each class and the decision boundary in a single plot is shown in Fig. 4.

III. Result Analysis

Wit respect to the decision boundary two training point were not correctly classified and one test sample was not correctly classified. Hence, I obtained training accuracy of 83.33% and test accuracy of 85.71%.

IV. Conclusion

This algorithm is easy to implement. As the procedures and procedures are simple hence computations are faster. However, this algorithm has higher chance of misclassification as the decision boundary is linear.

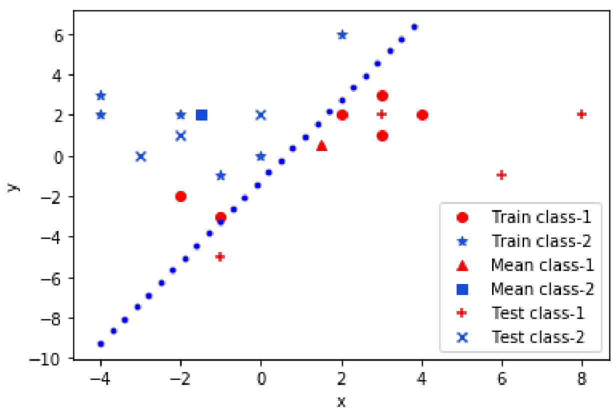

Fig. 4. Graphical representation of decision boundary along with training data, mean of each class and classified testing data using matplotlib.

V. Algorithm Implementation / Code

Code:

```python
#Plotting training data
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
#loading training data
#Plotting training data
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

#Plotting training data
x,y,z = np.loadtxt('train.txt',unpack=True, delimiter=' ')
plt.xlabel('x')
plt.ylabel('y')
for m in range(len(z)):
    #print(m + 1, days[m])
    if z[m]==1:
        xc1=plt.scatter(x[m], y[m], color='r')
    elif z[m]==2:
        xc2=plt.scatter(x[m], y[m], marker='*', color='#184DD5')

plt.legend([xc1, xc2], ["Train class-1", "Train class-2"])
plt.show()

#x component of mean1
m1x=[]
for m in range(len(z)):
    if z[m]==1:
        m1x.extend([x[m]])

print(m1x)
ux1=np.mean(m1x)
print(ux1)
#y component of mean1
m1y=[]
for m in range(len(z)):
    if z[m]==1:
        m1y.extend([y[m]])

print(m1y)
uy1=np.mean(m1y)
print(uy1)
#x component of mean2
m2x=[]
for m in range(len(z)):
    if z[m]==2:
        m2x.extend([x[m]])

print(m2x)
ux2=np.mean(m2x)
print(ux2)
#y component of mean2
m2y=[]
for m in range(len(z)):
    if z[m]==2:
        m2y.extend([y[m]])

print(m2y)
uy2=np.mean(m2y)
print(uy2)

#making list of x y components of mean of two classes
mean1=[ux1,uy1]
print(mean1)
mean2=[ux2,uy2]
print(mean2)

#plotting mean of two classes along with training data
plt.xlabel('x')
plt.ylabel('y')
for m in range(len(z)):
    if z[m]==1:
        xc1=plt.scatter(x[m], y[m], color='r')
    elif z[m]==2:
        xc2=plt.scatter(x[m], y[m], marker='*', color='#184DD5')

xm1=plt.scatter(ux1, uy1, marker='^', color='r')
xm2=plt.scatter(ux2,uy2, marker='s', color='#184DD5')
plt.legend([xc1, xc2, xm1, xm2], ["Train class-1", "Train class-2", "Mean class-1", "Mean class-2"])
plt.show()

#classifying testing data using minimum distance to class mean classifier
w = np.loadtxt('test.txt',unpack=False, delimiter=' ')
plt.xlabel('x')
plt.ylabel('y')
for m in range(len(z)):
    if z[m]==1:
        xc1=plt.scatter(x[m], y[m], color='r')
    elif z[m]==2:
        xc2=plt.scatter(x[m], y[m], marker='*', color='#184DD5')

xm1=plt.scatter(ux1, uy1, marker='^', color='r')
xm2=plt.scatter(ux2,uy2, marker='s', color='#184DD5')
for i in range(len(w)):
    g1=pd.Series(w[i, 0:2]).dot(pd.Series(mean1))-0.5*pd.Series(mean1).dot(pd.Series(mean1))
    g2=pd.Series(w[i, 0:2]).dot(pd.Series(mean2))-0.5*pd.Series(mean2).dot(pd.Series(mean2))
    if g1>g2:
        xt1=plt.scatter(w[i,0], w[i,1], marker='+', color='r')
    elif g1<g2:
        xt2=plt.scatter(w[i,0],w[i,1], marker='x', color='#184DD5')
plt.legend([xc1, xc2, xm1, xm2,xt1,xt2], ["Train class-1", "Train class-2", "Mean class-1", "Mean class-2",
```

```
                                         "Test class-1", "Test class-2"])
plt.show()

#y intercept
minx=min(x)
print(minx)
maxx=max(x)
print(maxx)

for j in range(int(minx),int(maxx),1):
    print(j)
s1=pd.Series(mean1).dot(pd.Series(mean1))
s2=pd.Series(mean2).dot(pd.Series(mean2))

intercept=0.5*(s1-s2)
print(intercept)
#x-coefficient and y coefficient
coeff=np.array(mean1)-np.array(mean2)
print(coeff)
print(coeff[1])
#list of range of x from -4 to 4
import decimal
def float_range(start, stop, step):
  while start < stop:
    yield float(start)
    start += decimal.Decimal(step)

rangex=list(float_range(int(minx), int(maxx), '0.3'))
print(rangex)

#classifying testing data using minimum distance to class mean classifier
w = np.loadtxt('test.txt',unpack=False, delimiter=' ')
plt.xlabel('x')
plt.ylabel('y')
pred=[]
for m in range(len(z)):
    #print(m + 1, days[m])
    if z[m]==1:
        plt.scatter(x[m], y[m], color='r')
    elif z[m]==2:
        plt.scatter(x[m], y[m], marker='*', color='#081F59')

plt.scatter(ux1, uy1, marker='^', color='r')
plt.scatter(ux2,uy2, marker='s', color='#081F59')
for i in range(len(w)):
    g1=pd.Series(w[i, 0:2]).dot(pd.Series(mean1))-0.5*pd.Series(mean1).dot(pd.Series(mean1))
    g2=pd.Series(w[i, 0:2]).dot(pd.Series(mean2))-0.5*pd.Series(mean2).dot(pd.Series(mean2))
    if g1>g2:
        plt.scatter(w[i,0], w[i,1], marker='+', color='r')
        pred.extend([1])
    elif g1<g2:
        plt.scatter(w[i,0],w[i,1], marker='x', color='#081F59')
        pred.extend([2])

for j in range(len(rangex)):
    yval=-(coeff[0]*(rangex[j])+intercept)/coeff[1]
    plt.plot(rangex[j],yval, '.b')


print(pred)

#Testing Accuracy
cnt=0
for k in range(len(w)):
    if pred[k]==w[k][2]:
        cnt=cnt+1

print(cnt)
print(len(w))

test_acc=cnt/len(w)*100
print("Test accuracy: ", test_acc, "%")

#Training Accuracy
train_acc=10/12*100
print("Train accuracy: ", train_acc, "%")
```