Designing a Minimum Distance to Class Mean Classifier

Mayisha Farzana
dept.Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204028@aust.edu

***Abstract*—The objectives of this experiment is to know how a simple classifier works. The classifier implemented in this experiment may not work correctly in all situation but the purpose to know how a classifier works can be accomplished. Minimum Distance to Class Mean Classifier is used for classifying sample data with respect to the class mean value. There are data points which are labeled with two classes in the train data. The linear discriminant function linearly separates the two classes. Using the train data the model is first trained and using the test data the model is tested, that is which data are correctly classified and which are not. This method provides a good accuracy in classifying the unknown data.**

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

I. Introduction

The minimum distance to class mean classifier defines classes in terms of the distance from the mean vector for the class. In this type of classifier the mean point for the sample points of known classes are at first calculated, and then the sample points of unknown class are assigned to the class for which the distance to class mean is the minimum.

We are using the function :

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

Here $\bar{Y}$ is the mean of an individual class, X is the feature vector. So we have to calculate the mean of individual classes and then using this rule we can find the minimum distance to a class mean.

The decision boundary separates each class from another. Minimum distance to class mean classifier is a linear classifier. So this algorithm fails to classify all samples correctly because of its linear property. When many samples is taken and they are scattered in feature space then linear decision boundary is unable to classify all of them.

II. Experimental Design / Methodology

Two data set is given which have labeling of classes with it. One is for training and another one is for testing the model.

A. Plotting the training dataset : Two classes in each dataset were given. Firstly, we have to plot all the sample points. Samples from the same class, were plotted using the same color and marker so that different classes can be distinguished easily. For this I have taken two numpy arrays to store the classified data according to their class level and plotted them with different marker.

B. Test the classifier by the given test points : Task 02 is to classify the data points and plot the mean values. The mean values of x and y are calculated. Then g1 and g2 are computed using the test datas and mean values. If g1 is greater than g2 the point is in class 1 else in class 2. These predicted class values with the data points are saved in the array.

C. Defining and drawing the decision boundary :

To draw the decision boundary we have to draw a line in which every point is equidistant from both the class mean. Thus the discriminant function value on the line for each of the classes should be same. If gi(x) is the discriminant function value of class 'i' then we can write

$$g_i(x) = g_j(x) ------(i)$$

By substituting the value with the discriminant function value we obtain

$$w_i^T x – 0.5 w_i^T w_i = w_j^T x - 0.5 w_j^T w_j$$

$$x(w_i^T - w_j^T) - 0.5(w_i^T w_i - w_j^T w_j) = 0$$

Where w_i and w_j are the mean of class i and j resepectively. Now this equation is a form of mx+c=0 ——(ii) which is an equation of line. By comparing this two it can be written that $c = -0.5(w_i^T w_i - w_j^T w_j)$ and $m = (w_i^T - w_j^T)$. If we take the lowest and highest value of x and then define a range between this two values then we will obtain a set of x values. By putting these values of c in the above equation we get a set of y values.

$$y = -((w_i x - w_j x)x + 0.5(w_i^T w_i - w_j^T w_j))/(w_i y - w_j y)$$

Then after plotting these values of x and y we will obtain a decision boundary which will divide the whole feature space with two region. Using the equation y= mx + c, the x and y values are plotted which is a line or decision boundary between two classes.

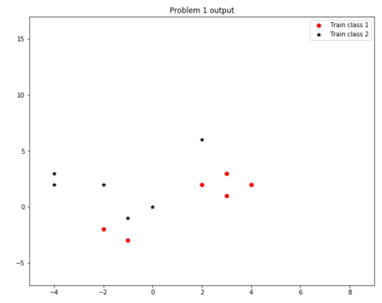
D. Calculating the accuracy: The accuracy is calculated using –

$$(correctclassification/totalclassification) * 100$$

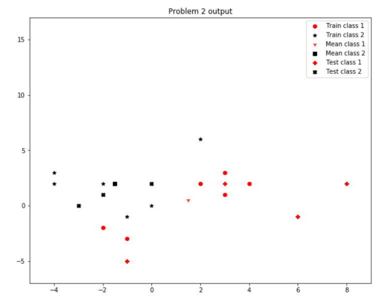
Where correct classification is the number of data points which are correctly classified, that is the predicted class value matched the test class value. Total classification is the number of total test classes.

III. Result Analysis

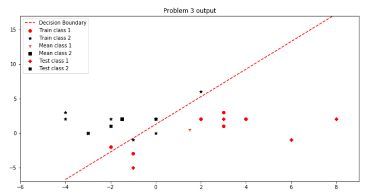
The results of four tasks are given one by one.
1. Plotting of sample points of training data set.



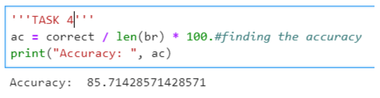
Train class 1 and train class 2 are plotted using different markers and different colors.
2. Plotting of the means of the training data set and the test data set according to their predicted classes.



Means of the classes and test class 1 and test class 2 are plotted with different markers and colors.
3. Plotting of the decision boundary.



The decision boundary using the above mention equation is plotted.
4. Accuracy of the model using given class and predicted class.



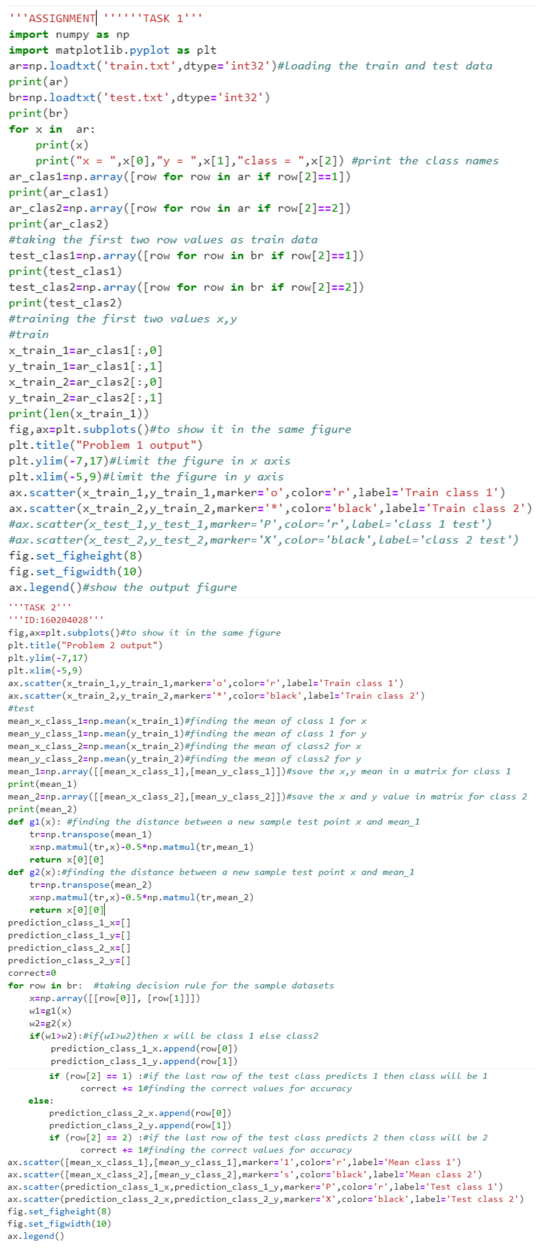
```
'''TASK 4'''
ac = correct / len(br) * 100.#finding the accuracy
print("Accuracy: ", ac)

Accuracy:  85.71428571428571
```

Accuracy is measured using the above mentioned equation.The accuracy for this dataset is 85.71%

IV. Conclusion

In this model using the given dataset, some samples are classified and some are misclassified. The minimum distance to class mean classifier model is easy for calculations and it provides a good accuracy. But the demerits of this algorithm is that, it misclassifies some samples as the boundary between the two classes is linear.The number of data points in the dataset is low.Using more data will make the model more robust with increased accuracy.

V. Algorithm Implementation / Code



```
'''ASSIGNMENT '''''TASK 1'''
import numpy as np
import matplotlib.pyplot as plt
ar=np.loadtxt('train.txt',dtype='int32')#Loading the train and test data
print(ar)
br=np.loadtxt('test.txt',dtype='int32')
print(br)
for x in  ar:
    print(x)
    print("x = ",x[0],"y = ",x[1],"class = ",x[2]) #print the class names
ar_clas1=np.array([row for row in ar if row[2]==1])
print(ar_clas1)
ar_clas2=np.array([row for row in ar if row[2]==2])
print(ar_clas2)
#taking the first two row values as train data
test_clas1=np.array([row for row in br if row[2]==1])
print(test_clas1)
test_clas2=np.array([row for row in br if row[2]==2])
print(test_clas2)
#training the first two values x,y
#train
x_train_1=ar_clas1[:,0]
y_train_1=ar_clas1[:,1]
x_train_2=ar_clas2[:,0]
y_train_2=ar_clas2[:,1]
print(len(x_train_1))
fig,ax=plt.subplots()#to show it in the same figure
plt.title("Problem 1 output")
plt.ylim(-7,17)#Limit the figure in x axis
plt.xlim(-5,9)#Limit the figure in y axis
ax.scatter(x_train_1,y_train_1,marker='o',color='r',label='Train class 1')
ax.scatter(x_train_2,y_train_2,marker='*',color='black',label='Train class 2')
#ax.scatter(x_test_1,y_test_1,marker='P',color='r',label='class 1 test')
#ax.scatter(x_test_2,y_test_2,marker='X',color='black',label='class 2 test')
fig.set_figheight(8)
fig.set_figwidth(10)
ax.legend()#show the output figure
'''TASK 2'''
'''ID:160204028'''
fig,ax=plt.subplots()#to show it in the same figure
plt.title("Problem 2 output")
plt.ylim(-7,17)
plt.xlim(-5,9)
ax.scatter(x_train_1,y_train_1,marker='o',color='r',label='Train class 1')
ax.scatter(x_train_2,y_train_2,marker='*',color='black',label='Train class 2')
#test
mean_x_class_1=np.mean(x_train_1)#finding the mean of class 1 for x
mean_y_class_1=np.mean(y_train_1)#finding the mean of class 1 for y
mean_x_class_2=np.mean(x_train_2)#finding the mean of class2 for x
mean_y_class_2=np.mean(y_train_2)#finding the mean of class2 for y
mean_1=np.array([[mean_x_class_1],[mean_y_class_1]])#save the x,y mean in a matrix for class 1
print(mean_1)
mean_2=np.array([[mean_x_class_2],[mean_y_class_2]])#save the x and y value in matrix for class 2
print(mean_2)
def g1(x): #finding the distance between a new sample test point x and mean_1
    tr=np.transpose(mean_1)
    x=np.matmul(tr,x)-0.5*np.matmul(tr,mean_1)
    return x[0][0]
def g2(x):#finding the distance between a new sample test point x and mean_1
    tr=np.transpose(mean_2)
    x=np.matmul(tr,x)-0.5*np.matmul(tr,mean_2)
    return x[0][0]
prediction_class_1_x=[]
prediction_class_1_y=[]
prediction_class_2_x=[]
prediction_class_2_y=[]
correct=0
for row in br:  #taking decision rule for the sample datasets
    x=np.array([[row[0]], [row[1]]])
    w1=g1(x)
    w2=g2(x)
    if(w1>w2):#if(w1>w2)then x will be class 1 else class2
        prediction_class_1_x.append(row[0])
        prediction_class_1_y.append(row[1])
        if (row[2] == 1) :#if the last row of the test class predicts 1 then class will be 1
            correct += 1#finding the correct values for accuracy
    else:
        prediction_class_2_x.append(row[0])
        prediction_class_2_y.append(row[1])
        if (row[2] == 2) :#if the last row of the test class predicts 2 then class will be 2
            correct += 1#finding the correct values for accuracy
ax.scatter([mean_x_class_1],[mean_y_class_1],marker='1',color='r',label='Mean class 1')
ax.scatter([mean_x_class_2],[mean_y_class_2],marker='s',color='black',label='Mean class 2')
ax.scatter(prediction_class_1_x,prediction_class_1_y,marker='P',color='r',label='Test class 1')
ax.scatter(prediction_class_2_x,prediction_class_2_y,marker='X',color='black',label='Test class 2')
fig.set_figheight(8)
fig.set_figwidth(10)
ax.legend()
```

```python
'''TASK 3'''
fig,ax = plt.subplots()
fig.set_figheight(6)
fig.set_figwidth(12)
plt.ylim(-7, 17)
plt.xlim(-6, 9)
plt.title("Problem 3 output")
#plt.grid(color = 'red', linestyle = '-')
ax.scatter(x_train_1, y_train_1, marker = 'o', color = 'red', label = 'class 1 train')
ax.scatter(x_train_2, y_train_2, marker = '*', color = 'black', label = 'class 2 train')
ax.scatter(prediction_class_1_x, prediction_class_1_y, marker = 'P', color = 'red', label = 'class 1 train')
ax.scatter(prediction_class_2_x, prediction_class_2_y, marker = 'X', color = 'black', label = 'class 2 train')
#ploting mean
ax.scatter([mean_x_class_1], [mean_y_class_1], marker = '1', color = 'red', label = 'class 1 train')
ax.scatter([mean_x_class_2], [mean_y_class_2], marker = 's', color = 'black', label = 'class 2 train')
#Decision boundary equation y=mx+c
def eq(x) :
    u = ((mean_x_class_1 - mean_x_class_2) * x) - 0.5 * (np.dot(np.transpose(mean_1), mean_1)) + 0.5 * (np.dot(np.transpose(mean_2), mean_2))
    v = u / (mean_y_class_1 - mean_y_class_2)
    return -v[0][0]
#y=-m1*x+c/m2
#c=-0.5*(w1^t*w1-w2^tw2)
xs = []
ys = []
xx = []
yy = []
for x in np.arange(-4, 10, 0.5):#limiting a range for drawing the decision boundary increasing it by 9.5
    xx.append(x)
    yy.append(eq(x))
ax.plot(xx, yy, '--', color = 'red', label = 'Decision Boundary')
ax.legend()
plt.show()
'''TASK 4'''
ac = correct / len(br) * 100.#finding the accuracy
print("Accuracy: ", ac)
```