Designing a Minimum Distance to Class Mean Classifier

Marufa Kamal
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204073@aust.edu

***Abstract*—Classification algorithms train the data to determine unknown items based on different features to classify the unknown sample to the correct class. In this experiment, we try to analyze one of the supervised classification algorithms, known as the Minimum Distance to Class Mean Classier. We train our model with some labeled training data and determine experimental results from the testing data, observing the accuracy of the model and how well it can classify unknown samples.**

***Index Terms*—Classification, Minimum distance classifier, Mean classifier, Supervised learning**

I. Introduction

The Minimum Distance to Class Mean Classier is a technique that classifies unknown data such as images or sample vectors to their appropriate class based on the distance of the mean vector of the known classes to the unknown data. We at first, calculate the mean vectors $\overline{Y}$ for each individual class where X represents the input feature vectors. Next, using the linear discriminant function

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

we can identify the class of the unclassified sample vectors in the testing data based on mean distance. As Minimum Distance to Class Mean Classier is a linear classifier we can calculate a linear decision boundary separating the classes.

II. Experimental Design / Methodology

Based on the given testing and training data I took on the challenge of completing the four tasks which were assigned. An overview of the process of completion are given below.

A. Plotting Training Sample Points

The training data set consisted of 12 vector samples of which 6 belonged to class 1 and the remaining to class 2. Using the python package '**matplotlib**' I have plotted all sample points from the '*train.txt*' file making sure that the samples from the same class are marked with the same color and marker. Storing the values of the 2 classes in separate **numpy** arrays the points are plotted.

B. Classifying Test data and Determining Mean Vectors

Computing the mean vector results for both the classes using the discriminant function that has been derived from euclidian distance formula

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

I have plotted the 2 mean vector results in the graph. I have also classified the test points depending on the value of $g_1(X)$ and $g_2(X)$. If $g_1(X) > g_2(X)$ the sample point belongs to class 1 and vice-versa.

C. Drawing the Decision Boundary

$$g_1(x) = \mathrm{x}_1^T \bar{Y} - \frac{1}{2}\bar{Y}_1^T \bar{Y} \tag{1}$$

$$g_2(x) = \mathrm{x}_2^T \bar{Y} - \frac{1}{2}\bar{Y}_2^T \bar{Y} \tag{2}$$

From equation (1) and (2), we can write $g_1(x)=g_2(x)$ which further results in

$$x(\omega_1^T - \omega_2^T) - \frac{1}{2}(\omega_1^T \omega_1 - \omega_2^T \omega_2)$$

Here, ω_1 and ω_2 are the mean vectors. Comparing this equation with the line equation Y=mx+c and taking the vector value for x we get the following equation

$$y = -(x(\omega_1 x - \omega_2 x) + \frac{1}{2}(\omega_1^T \omega_1 - \omega_2^T \omega_2))/(\omega_1 y - \omega_2 y)$$

D. Accuracy Calculation

The accuracy is calculated using the following equation

$$\frac{N}{T} * 100$$

Here, N = Number of Correctly classified samples in predicted values and T= Total number of samples in the test data

III. Result Analysis

Results of the Tasks given are described as follows.

a) After plotting the training samples, they remain scattered in the graph respective to their x and y coordinates and marked with different color and marker.

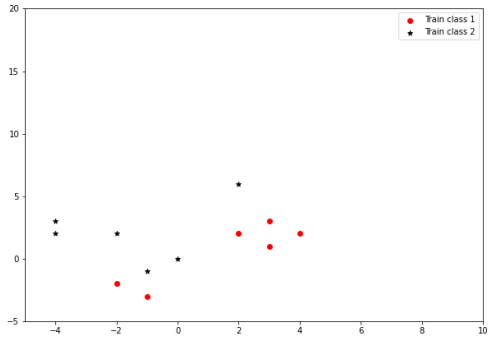

Fig. 1: Plotting Training Sample Points

b) We plot the test data samples after classifying them along with the mean vector of the two classes. All these are marked with separate markers for better understanding.

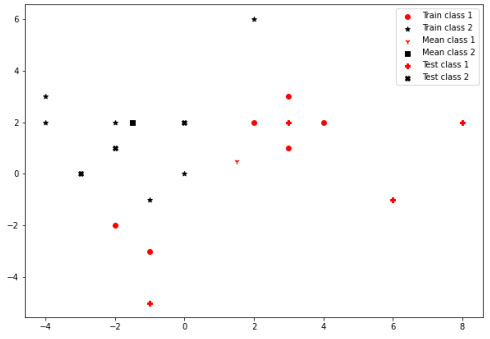

Fig. 2: Classifying Test data and Determining Mean Vectors

c) We draw the linear decision boundary in the graph which separates the 2 class.

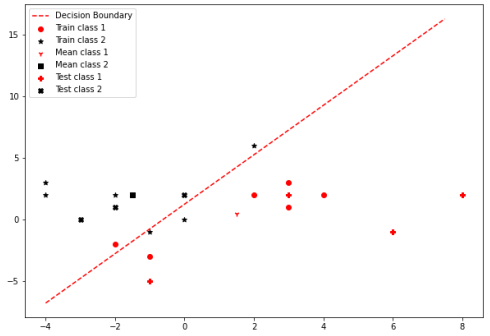

Fig. 3: Drawing the Decision Boundary

d) The model outputs an accuracy of 85.71% after computing it based on the predicted values.

```
Accuracy:  85.71
```

Fig. 4: Accuracy Calculation

IV. Conclusion

Using the model we got an accuracy of 85.71% which is good for this training and testing samples. Only one test sample could not be predicted accurately. Minimum Distance to Class Mean Classifier is a linear classification algorithm where the decision boundary acts in a linear format. For data's which need non-linear decision boundary this method can not provide better results.

V. Algorithm Implementation / Code

The implementation of the tasks completed with python language are given as follows.

```
import matplotlib.pyplot as plt
import numpy as np
train_data =np.int32(np.loadtxt('/content/drive/MyDrive/4.2/train.txt'))
test_data =np.int32(np.loadtxt('/content/drive/MyDrive/4.2/test.txt'));
#print(train_data);
#print(test_data);
print(train_data[1][1]);
train_data_c1=[];
train_data_c2=[];
for item in train_data:
    if item[2]==1:
        #print(item[2]);
        train_data_c1.append(item)
    elif item[2]==2:
        train_data_c2.append(item)
train_data_class1= np.array(train_data_c1);
train_data_class2= np.array(train_data_c2);
#
print(train_data_class1);
print(train_data_class2);
x1,y1= train_data_class1[:,0],train_data_class1[:,1];
x2,y2= train_data_class2[:,0],train_data_class2[:,1];
print("x1",x1);
print("x2",x2);
fig=plt.figure(1, figsize=(10,7))
chart1= fig.add_subplot()
chart1.scatter(x1,y1,marker='o',color='r',label='Train class 1');
chart1.scatter(x2,y2,marker='*',color='k',label='Train class 2');

chart1.axis([-5,10,-5,20]);
chart1.legend()#
```

Fig. 5: Plotting Training Sample Points

```
#finding the mean of both the classes
w1=np.array([[np.mean(x1),np.mean(y1)]])
w2=np.array([[np.mean(x2),np.mean(y2)]])
#print(w1)
p1x,p1y,p2x,p2y,final_p=([] for i in range(5))
acc=0

for a in test_data:
  x=np.array([[a[0]],[a[1]]])
  g1=(np.matmul(w1,x)-0.5*np.matmul(w1,np.transpose(w1)))
  g2=(np.matmul(w2,x)-0.5*np.matmul(w2,np.transpose(w2)))
  print("g1",g1,"g2",g2)
  # check g1 g2
  if (g1[0][0]>g2[0][0]):
    p1x.append(a[0])
    p1y.append(a[1])
    if a[2]==1:
      acc=acc+1
  elif (g1[0][0]<g2[0][0]):
    p2x.append(a[0])
    p2y.append(a[1])
    if a[2]==2:
      acc=acc+1
```

Fig. 6: Classifying Test data and Determining Mean Vectors

```
#min from test and train data
min_test = np.amin(test_data, axis=0)
min_train = np.amin(train_data, axis=0)
xmin=min(min_test[0],min_train[0])
#max from test and train data
max_test = np.amax(test_data, axis=0)
max_train = np.amax(train_data, axis=0)
xmax=max(max_test[0],max_train[0])

db_x,db_y = ([] for i in range(2))

for x in np.arange(xmin,xmax, 0.5):
    db_x.append(x)
    y = ((-(w1[0][0] - w2[0][0]) * x) + 0.5 *
         ((np.dot((w1), np.transpose(w1))) -
         (np.dot(w2,np.transpose(w2)))))/(w1[0][1] - w2[0][1])
    db_y.append(y[0][0])
    #print(y[0][0])


fig=plt.figure(1, figsize=(10,7))
chart1= fig.add_subplot()
chart1.scatter(x1,y1,marker='o',color='r',label='Train class 1');
chart1.scatter(x2,y2,marker='*',color='k',label='Train class 2');
chart1.scatter(w1[0][0],w1[0][1],marker='1',color='r',label='Mean class 1');
chart1.scatter(w2[0][0],w2[0][1],marker='s',color='k',label='Mean class 2');
chart1.scatter(p1x,p1y,marker='P',color='r',label='Test class 1');
chart1.scatter(p2x,p2y,marker='X',color='k',label='Test class 2');
chart1.plot(db_x,db_y, '--', color = 'red', label = 'Decision Boundary')
chart1.legend()
```

Fig. 7: Drawing the Decision Boundary

```
#finding the accuracy
print("Accuracy: ",
      "{:.2f}".format(acc/len(test_data) * 100))
```

Fig. 8: Accuracy Calculation