Ahsanullah University of Science & Technology

Department of Computer Science & Engineering

Designing a Minimum Distance to Class Mean Classifier

Course Title: Pattern Recognition Lab

Course No: CSE 4214

Assignment No: 01

Date of Submission: 4.3.2020

Submitted By:

Name: Md. Reasad Zaman Chowdhury

Section: A1

Student ID: 160104004

Designing a Minimum Distance to Class Mean Classifier

Md. Reasad Zaman Chowdhury

16.01.04.004

Abstract:

'Minimum Distance to Class Mean Classifier' is a classification technique used to classify unclassified samples where vectors clustered in more than one classes are given. For example, our dataset contains training and testing data. Training data are already classified into different classes. Our objective will be to classify testing data by using Minimum Distance to Class Mean Classifier.

Keyword:

Classifier, Discriminant Function, Decision Rule, Accuracy.

Introduction:

'Minimum Distance to Class Mean Classifier' is one of the simplest classification techniques used to classify unknown samples. Because of its simpler calculations it is faster to train the model. But it generally used in the case where different classes are easier to separate.

The main theory behind this technique is very simple. Suppose that we have a n set of training samples X = {x1, x2, x3, ……. , x_n} each having dimension d. Also assume that our training data is labeled with m different classes. From training set it calculates mean value for the each of the m classes. Assume that i'th class has the a mean vector Y_i = {Y_{i1} , Y_{i2} , Y_{i3} ,…., Y_{id} } where Y_{ij} is the mean value of the j'th dimension of i'th class. From this processes we get our mean vector Y = {Y_1, Y_2, Y_3,……, Y_m}.

For classifying unknown samples, it uses the following discriminant function:

$$g_i(X) = X^T \overline{Y}_i - \frac{1}{2} \overline{Y}_1^T \overline{Y}_i$$

Where $g_i(X)$ is the discriminant function for class i over unknown sample, X is the unknown sample and Y_i is the mean vector for class i. We need to find the value of the discriminant function for all the m classes we have. We have the following decision rule for classifying the unknown sample:

$$\text{if } g_i(X) > g_j(X) \text{ then } X \in \text{Class if } i \neq j$$

That means that unknown samples will go to the class having largest value of discriminant function.

Experimental Design:

a. **Plot All Sample Points:**

At first, we visualized our training data by using scatter plot. Since our training data has 2 points, we plotted the data of different classes with different colors. Here blue points are class 1 samples and red points are class 2 samples.

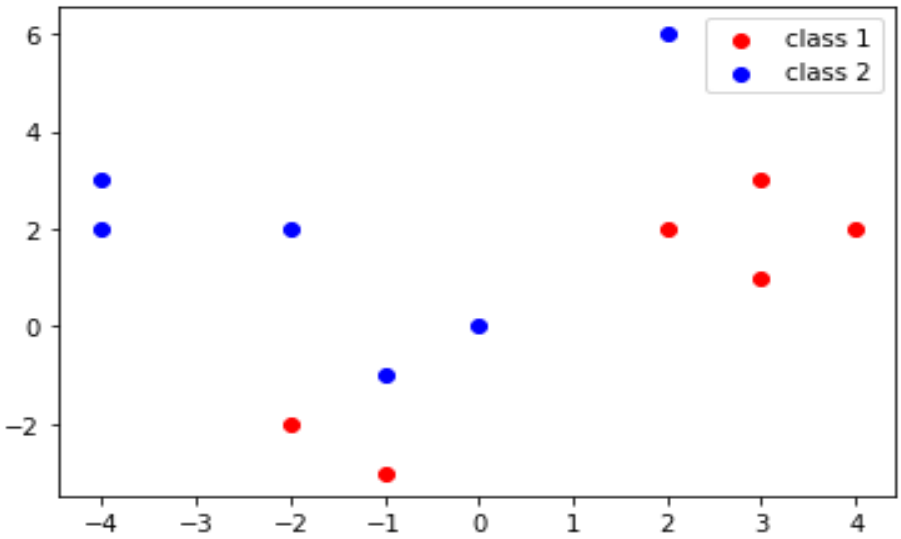

Fig 1: Scatter plot of training data points

b. Classify the test data:

Mean points of 2 classes are calculated and plotted with the same class color but with different markers. Then for each data points of our testing dataset we calculated the discriminant function value $g_i(X)$ for each class. After that our data point will be assigned to the class based on the decision rule mentioned earlier.

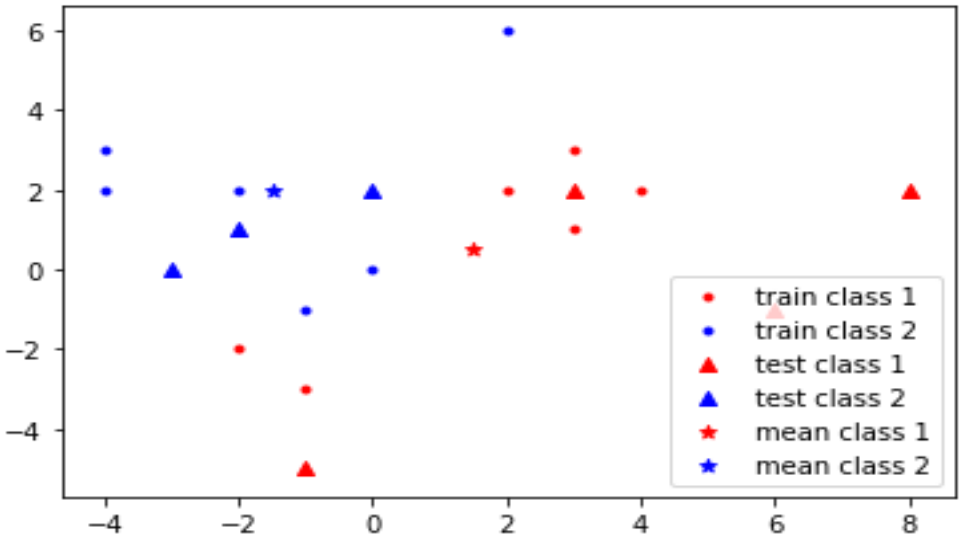

Fig 2: Scatter plot of training, testing and mean data points

In the above scatter plot red points indicates class 1 data and blue points indicates class 2 data. For training, testing and mean data points different markers have been used. Circle

indicates training data, triangle indicates testing data points and star indicates mean data points of our training samples.

c. Drawing Decision Boundary:

Decision boundary implies all the points whose distance from all the classes are same. However minimum and maximum values of x1 is used to calculate x2 values for our decision boundary which can be expressed in the following form:

$$(\overline{w_1}^T - \overline{w_2}^T)x - \tfrac{1}{2}(\overline{w_1}^T \overline{w_1} - \overline{w_2}^T \overline{w_2}) = 0$$

Where W_i is the mean vector of ith class.

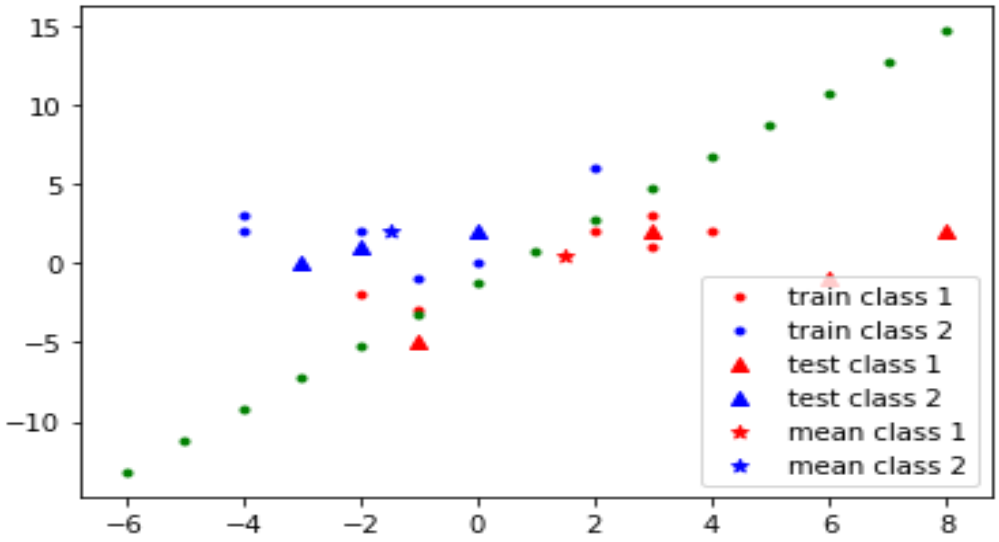

Fig 3: Drawing decision boundary

In Fig 3 green dots indicate the linear decision boundary which separates 2 class in our dataset.

d. Finding Accuracy:

Accuracy is a performance metric used in classification problem. This can tell how good a classifier performs for unknown cases. Accuracy has been calculated by the following formula:

Accuracy = (number data truly classified / number of total data points) * 100%

From the accuracy score we found out that our classifier has a accuracy of 85.714%.

Algorithm Implementation:

```
# In[1]:


get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd



# In[2]:



#Reading training and testing dataset
df_train = pd.read_csv('train.txt', sep=" " ,  header = None, dtype = 'Int64')
df_test = pd.read_csv('test.txt', sep=" " ,  header = None, dtype = 'Int64')



# In[3]:



#slicing the features and labels from training samples
x = np.array(df_train[0])
y = np.array(df_train[1])
label = np.array(df_train[2])

cdict = {1: 'red', 2: 'blue'}
```

```
#scatter plot according to the class label
fig, ax = plt.subplots()
for g in np.unique(label):
    ix = np.where(label == g)
    ax.scatter(x[ix], y[ix], c = cdict[g], label = 'class '+str(g))
ax.legend()
plt.show()


# In[4]:


#splitting 2 classes from dataframe
w1 = np.array(df_train[df_train[2] == 1])
w2 = np.array(df_train[df_train[2] == 2])

#calculating mean of 2 classes
u1 = np.array( [np.mean(w1[:,0]) , np.mean(w1[:,1])] )
u2 = np.array( [np.mean(w2[:,0]) , np.mean(w2[:,1])] )

#slicing unknown data from trainning samples
x_test = np.array(df_test.iloc[:,0:2])
true_label = df_test.iloc[:,2]
predicted = []

#calculate linear Discriminant Function for test samples
for xi in x_test:
    g1 = np.dot( np.transpose(u1) , xi) - 0.5 * np.dot(np.transpose(u1) , u1)
```

```
    g2 = np.dot( np.transpose(u2) , xi) - 0.5 * np.dot(np.transpose(u2) , u2)

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

#marging the predicted class labels with the testing samples
df_test[3] = predicted

print(df_test)



# In[5]:



#ploting the traing and testing data points
#scatter plot according to the class label

fig, ax = plt.subplots()

for g in np.unique(label):
    ix = np.where(label == g)
    ax.scatter(x[ix], y[ix], marker='.', c = cdict[g], label = 'train class '+str(g))

for g in np.unique(predicted):
    ix = np.where(predicted == g)
    ax.scatter(x_test[ix,0], x_test[ix,1], marker='^', c = cdict[g], label = 'test class '+str(g))
```

```
ax.scatter(u1[0], u1[1], marker='*', c = 'r', label = 'mean class 1')
ax.scatter(u2[0], u2[1], marker='*', c = 'b', label = 'mean class 2')

ax.legend(loc='lower right')
plt.show()


# In[6]:


#Decision Boundary

m = np.transpose(u1) - np.transpose(u2)
m1 = m[0]
m2 = m[1]

c = 0.5 * ( np.dot(np.transpose(u1), u1) - np.dot(np.transpose(u2) , u2) )

x_db = []
y_db = []

for xi in range(-6, 9):
    yi = - (m1 * xi + c) / m2
    y_db.append(yi)
    x_db.append(xi)
```

In[7]:

#ploting the traing and testing data points along with the decision boundary

```
fig, ax = plt.subplots()

for g in np.unique(label):
    ix = np.where(label == g)
    ax.scatter(x[ix], y[ix], marker='.', c = cdict[g], label = 'train class '+str(g))

for g in np.unique(predicted):
    ix = np.where(predicted == g)
    ax.scatter(x_test[ix,0], x_test[ix,1], marker='^', c = cdict[g], label = 'test class '+str(g))

ax.scatter(u1[0], u1[1], marker='*', c = 'r', label = 'mean class 1')
ax.scatter(u2[0], u2[1], marker='*', c = 'b', label = 'mean class 2')

ax.scatter(x_db, y_db, marker='.', c = 'g')

ax.legend(loc='lower right')
plt.show()
```

In[9]:

```
#Printing the accuracy
accuracy = ( len(true_label[true_label == predicted]) / len(true_label) ) * 100
print("Accuracy:",accuracy)
```

Result Analysis:

We calculated that our Minimum Distance to Class Mean Classifier has an accuracy of 85.714%. Although it is quite a good score for a classification problem, we cannot say that our classifier performs better. For training our classifier we used only 12 data and for testing we used 7 data which is not sufficient to generalize a decision about how good our classifier is.

Conclusion:

"Minimum Distance to Class Mean Classifier" is easier to implement for is simpler calculations. Because of its simple calculation it has a faster to train. This algorithm performs better for cases where data of different classes can be easily separable. But for more complex data, this performs very poorly which is the main weakness of this classifier.