Experimentation with 'Minimum Distance to Class Mean' Classifier

Md.Ashraful Haque
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204101@aust.edu

***Abstract*—Classification algorithms play an important Role in 'Information Science'. Although there are a lot of classifiers available, MDCM is one of the most simplest classifiers out there. It's simplicity and intuition allows student to chose it as the first algorithm in pattern recognition. In this Experiment we got up to 85.71% accuracy.**

***Index Terms*—Mean, Classification, Generalization**

I. Introduction

To classify a sample data between two different classes one of the most intuitive approach is to find out the mean of each class and then calculate the distance from that sample data to each mean. The **shortest distance** from the mean of a certain class will indicate the possibility of that sample data to be from that class.

Let,

d(i) = distance between sample data and the mean of class i and

d(j) = distance between sample data and the mean of class j.

if d(i) $>$ d(j) then x $\in$ j

II. Experimental Design / Methodology

The data data set was split into train and test sections. Where the number of training data was 12 and the number of testing data was 7.

Firstly, we calculated the coordinates of each class mean from the training data by separating the data set according to the the classes.

Secondly, We used the derived Discriminant Function to classify the test data. The Discriminant function is as follows,

$$\mathrm{g}_i\left(x\right) = \mu_i^T x - \frac{1}{2}\mu_i^T\mu_i$$

if , $\mathrm{g}_i\left(x\right) > \mathrm{g}_j\left(x\right)$ then $x \in \mathrm{g}_i\left(x\right)$

here , $\mu_i =$ Mean of class i $x =$ Input vector of coordinates
Thirdly, For calculating the decision boundary we followed the following derivation,

$$\mathrm{g}_1\left(x\right) = \mu_1^T x - \frac{1}{2}\mu_1^T\mu_1$$

$$\mathrm{g}_2\left(x\right) = \mu_2^T x - \frac{1}{2}\mu_2^T\mu_2$$

in Decision boundary, $\mathrm{g}_1\left(x\right) = \mathrm{g}_2\left(x\right)$
So,

$$\mu_1^T x - \frac{1}{2}\mu_1^T\mu_1 = \mu_2^T x - \frac{1}{2}\mu_2^T\mu_2$$

$$\Rightarrow \mu_1^T x - \mu_2^T x = \frac{1}{2}\mu_1^T\mu_1 - \frac{1}{2}\mu_2^T\mu_2$$

$$\Rightarrow \left(\mu_1^T - \mu_2^T\right) x = \frac{1}{2}\left(\mu_1^T\mu_1 - \mu_2^T\mu_2\right)$$

Let, $\left(\mu_1^T - \mu_2^T\right) = \mu^T$
where, μ^T is a vector consisting of two elements. One is for X-axis and one is for Y-axis say, [μ_a , μ_b]
And, x also is a vector where each component is a coordinate consisting of two values say [a , b]
So, we can transform the equation into,

$$\left[\mu_a, \mu_b\right]^T [a, b] = \frac{1}{2}\left(\mu_1^T\mu_1 - \mu_2^T\mu_2\right)$$

Finally, the Equation can be **generalized** into

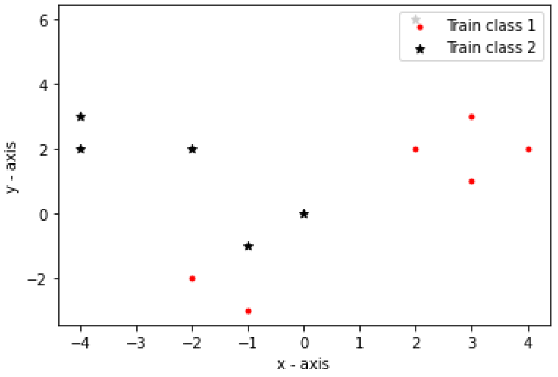
$$b = \frac{\frac{1}{2}\left\{\left(\mu_1^T\mu_1 - \mu_2^T\mu_2\right) - a\mu_a\right\}}{\mu_b}$$

And, Lastly for accuracy calculation,

$$accuracy = \frac{correct}{total}$$

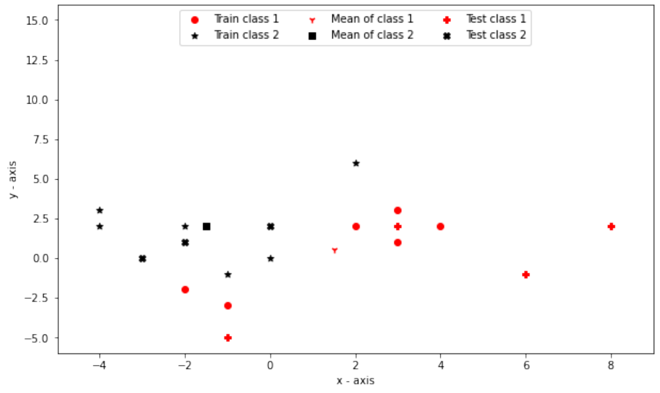
III. Result Analysis

Task 1:
In this task we plotted the Train data set .



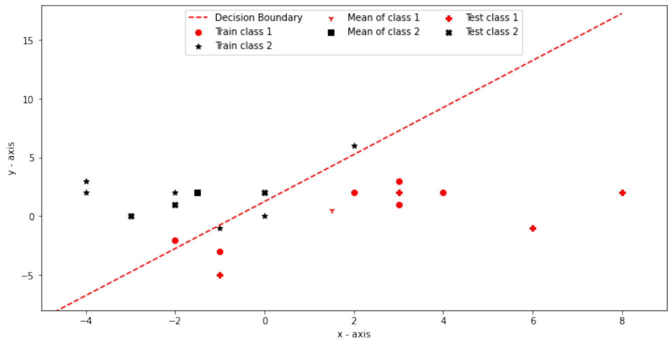
Task 2:

In task 2, we classified the test data and allocated different markers for each class.



Task 3:

In task 3, we drew the decision boundary equation with the help of the derived equation.



Task 4:

In this classification task our model's accuracy was 85.71

IV. Conclusion

This was just an experimentation with a well established legacy algorithm. It helped us to understand the basic ideas behind data, classification and Pattern Recognition.

V. Algorithm Implementation / Code

Preprocessing :

```
trainData = {}
entry = []

with open("train.txt") as f:
    for line in f:
        entry = line.split(" ")
        if (entry[2][-1] == '\n'):
            classInfo =  entry[2][:-1]
        else:
            classInfo =  entry[2]
        trainData[(int(entry[0]),int(entry[1]))] = int(classInfo)
    print(trainData)
```

```
trainData1x = []
trainData1y = []
trainData2x = []
trainData2y = []

for coor , classinfo in trainData.items():
    if(classinfo == 1):
        trainData1x.append(coor[0]);
        trainData1y.append(coor[1]);
    else:
        trainData2x.append(coor[0]);
        trainData2y.append(coor[1]);

print(trainData1x , trainData1y)
print(trainData2x , trainData2y)
```

Training:

```
classMeanX1 = sum(trainData1x) / len(trainData1x)
classMeanY1 = sum(trainData1y) / len(trainData1y)

classMeanX2 = sum(trainData2x) / len(trainData2x)
classMeanY2 = sum(trainData2y) / len(trainData2y)
print(classMeanX1 , classMeanY1 , classMeanX2 , classMeanY2  )
```

Classification:

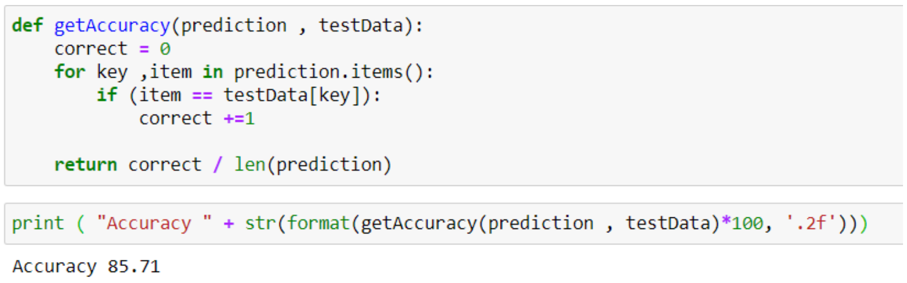
```
def gofX(point,miu):
    value = 0
    value = np.dot( np.transpose( miu), point) - .5*np.dot( np.transpose

    return value;
```

```
def predictClass(point,miu1 , miu2):
    predictedClass = 0

    if(gofX(point,miu1) > gofX(point,miu2)):
        return 1
    else:
        return 2

    return predictedClass
```

Plotting:

```
plt.figure(1,figsize = (12 , 6))

plt.scatter(trainData1x, trainData1y, color = 'r', marker = "o", label = 'Train class 1')

plt.scatter(trainData2x, trainData2y, color = 'k', marker = "*", label = 'Train class 2')

##means

plt.scatter(classMeanX1, classMeanY1, color = 'r', marker = "1", label = 'Mean of class 1')

plt.scatter(classMeanX2, classMeanY2, color = 'k', marker = "s", label = 'Mean of class 2')

##prediction

plt.scatter(predictionData1x , predictionData1y, color = 'r', marker = "P", label = 'Test class 1')

plt.scatter(predictionData2x , predictionData2y, color = 'k', marker = "X", label = 'Test class 2')

#decision boundary
plt.plot(decisionBoundaryX , decisionBoundaryY, "--r", label = 'Decision Boundary')


plt.xlabel('x - axis')
plt.ylabel('y - axis')

plt.axis([-5 ,9 ,-8 , 18])

plt.legend(loc='upper center', ncol=3)

plt.show()
```

Result:



```
def getAccuracy(prediction , testData):
    correct = 0
    for key ,item in prediction.items():
        if (item == testData[key]):
            correct +=1

    return correct / len(prediction)
```

```
print ( "Accuracy " + str(format(getAccuracy(prediction , testData)*100, '.2f')))
```

```
Accuracy 85.71
```

References