Implementing Minimum Error Rate Classifier

Devopriya Tirtho
16.02.04.033
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh

***Abstract*—In 'Machine Learning', for predicting the accurate class of an unknown sample we need to visualize our training samples whether they are linearly separable or not. 'Minimum Error Rate' classifier works with the help of bayes theory to generate the distribution of data points and then decide to predict the class of a data point.**

***Index Terms*—Machine Learning,Minimum Error Rate Classifier, Gaussian Normal Distribution, Probability Density Function, Bayesian Decision Theory.**

I. Introduction

Bayesian Decision Theory is a probabilistic approach to find a particular pattern. The theory works with the quantisation of the trade-off between some cassification approaches by the help of probabilistic approach and the cost that accompany those approaches. Based on this theory, a classifier develops named **'Minimum Error Rate Classifier'**. **Minimum Error Rate Classifier** is a classifier which tries to classify something by assuming the distribution of the thing with the help of **posterior probability** which consists of **likelihood probability** and **prior probability**. This type of classification method is called **generative approach**. Other machine learning approaches try to draw a fine line for classifying pattern whereas, this method try to understand the distribution of patterns.

II. Task and Experimental Design

There is a dataset named 'test' where all the datapoints are given. A row-matrix of 2 elements represents the data. The given data works as the input data which has to be classified with the help of given prior probability information. The likelihood probability will be calculated with the help of given **mean** and **variance matrix**.

$$P(x|\omega 1) = N(\mu 1, \Sigma 1);\ \mu 1 = \begin{bmatrix} 0 & 0 \end{bmatrix} \text{ and } \Sigma 1 = \begin{bmatrix} 0.25 & 0.3 \\ 0.3 & 1 \end{bmatrix} \tag{1}$$

$$P(x|\omega 2) = N(\mu 2, \Sigma 2);\ \mu 2 = \begin{bmatrix} 2 & 2 \end{bmatrix} \text{ and } \Sigma 2 = \begin{bmatrix} 0.5 & 0.0 \\ 0.0 & 0.5 \end{bmatrix} \tag{2}$$

$P(\omega 1)$= 0.5
$P(\omega 2)$= 0.5

- Our first task is to classify the sample points. For classification we use the **bayesian decision rule** which is :

$$P(\omega|x) = (P(x|\omega) * P(\omega))/P(x) \tag{3}$$

Here, the value of $likelihood\, probability$ is unknown. The value of $likelihood\, probability$ can be derived from the **Gaussian Normal Distribution's** formula. The formula is:

$$N_k(x_i|\mu_k, \Sigma_k) = \frac{e^{-0.5(x_i-\mu_k)^T \Sigma_k^{-1}(x_i-\mu_k)}}{\sqrt{(2\pi)^D |\Sigma_k|}} \tag{4}$$

- Secondly, we have to multiply the $prior$ value with the derived $likelihood$ value for each μ and Σ. Based on a decision rule, the classification occurs. The rule is:

$$P(x|\omega_1) * P(\omega_1) > P(x|\omega_2) * P(\omega_2) \tag{5}$$

if, $true$, class=ω_1
$else$, class=ω_2

- Thirdly, we have to plot the data in the graph as follows:

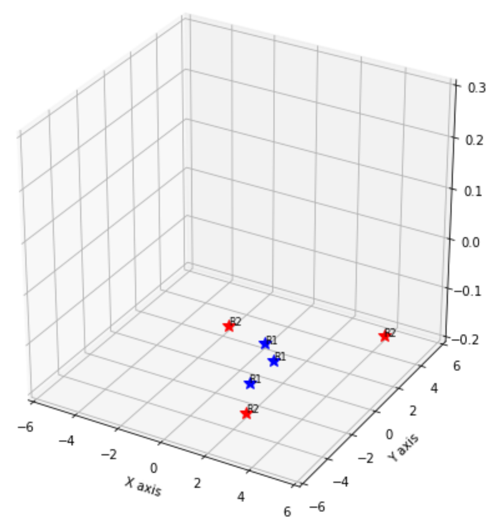

Fig. 1. Datapoints of Class 1 and Class 2

- Then, we have to draw the probability density function and visualize with a contour graph as follows:

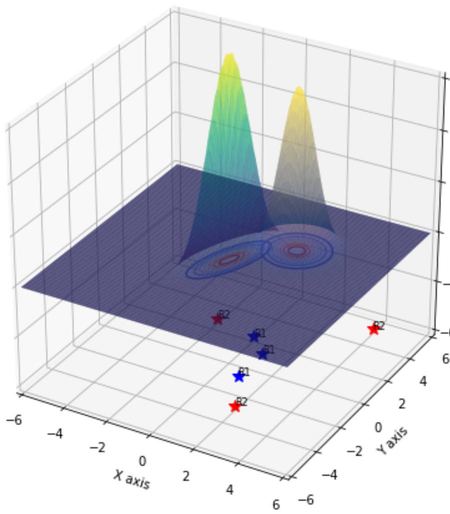

Fig. 2. Visualization of Probability Density Function along with Its Contour

- Finally, we have to draw the **decision boundary** for the model. For drawing the decision boundary we have to find the $y-coordinate$ values for some $x-coordinate's$ values. I have taken a stream of data point from -6 to 6 with equal difference of 0.5 for the $sample\ matrix$ and calculate the $y-coorcinates'$ value with the help of the following equation:

$$Y = \sqrt{\frac{\Sigma_1}{\Sigma_2}} e^{-\frac{(x-\mu_2)^T \Sigma_k^{-1}(x-\mu_2)}{2} + \frac{(x-\mu_1)^T \Sigma_k^{-1}(x-\mu_1)}{2}} \frac{\Theta}{1-\Theta} \tag{6}$$

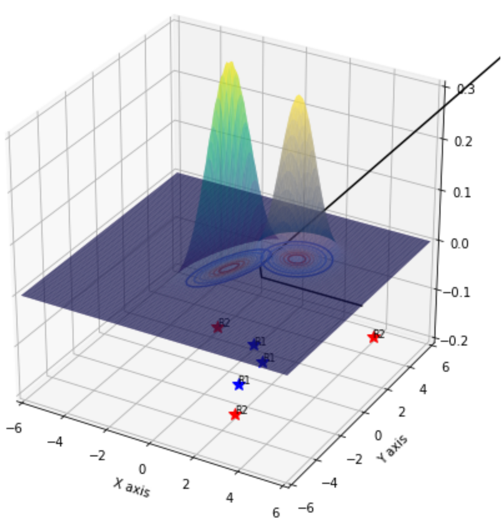

Fig. 3. Drawing of Decision Boundary

III. Result Analysis

TABLE I
TABLE FOR CLASSIFICATION OF SAMPLE DATA

X1	*X2*	*Class*
1.0	**1.0**	**1**
1.0	**-1.0**	**1**
4.0	**5.0**	**2**
-2.0	**2.5**	**2**
0.0	**2.0**	**1**
2.0	**-3.0**	**2**

IV. Python Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

test_data = pd.read_csv('test.txt',
 sep=",", header=None)
test_data.columns = ["x1", "y1"]



#Declaring Parameters
prior1=0.5
prior2=0.5
mu1=np.array([0,0])
mu2=np.array([2,2])
sigma1=np.array([[0.25,0.3],[0.3,1]])
sigma2=np.array([[0.5,0],[0,0.5]])

print(mu1)
print(mu2)
print(sigma1)
print(sigma2)

det_sigma1= np.linalg.det(sigma1)
det_sigma2= np.linalg.det(sigma2)

print(det_sigma1)
print(det_sigma2)

from numpy.linalg import inv
inv_sigma1 = inv(sigma1)
inv_sigma2 = inv(sigma2)

print(inv_sigma1)
print(inv_sigma2)

b=test_data.shape
print(b[0])
RowNumber=b[0]
PI = 3.14159

normal_dist_1=[]
```

```
normal_dist_2 =[]

for i in range (0,RowNumber):
      test_datapoint=np.array
        ([test_data.at[i,'x1'],
      test_data.at[i,'y1']])
      normal_dist_1.append
        ((1/(2*PI*np.sqrt(det_sigma1)))
      *(np.exp(-0.5*(np.dot
        ((test_datapoint - mu1).
transpose()
          , np.dot(inv_sigma1,
        (test_datapoint - mu1)))))))
            normal_dist_2.append
        ((1/(2*PI*np.sqrt(det_sigma2)))
          *(np.exp(-0.5*(np.dot
        ((test_datapoint - mu2).
        transpose(),
           np.dot
        (inv_sigma2,
        (test_datapoint - mu2)))))))

posterior_1 = [i * prior1 for
i in normal_dist_1]
posterior_2 = [i * prior2 for
i in normal_dist_2]

print(posterior_1)
print(posterior_2)

#Testing Class
test_class =[]
for i in range(6):
  if (posterior_1[i]>posterior_2[i]):
    test_class.append(1)
  else:
    test_class.append(2)
test_data['class']=test_class

test_data

plot_data = pd.read_csv('plot.txt',
sep=" ", header=None)
plot_data.columns = ["x1", "y1"]
plot_data

c=plot_data.shape
print(c[0])
RowNumber=c[0]
PI = 3.14159

coordinates =[]


for i in range(0,RowNumber):
    test_datapoint=np.array
      ([plot_data.at
      [i,'x1'],plot_data.at[i,'y1']])
      coordinates.append
      (np.sqrt(det_sigma1/det_sigma2)*
      (np.exp((-
      np.dot((test_datapoint - mu1).
      transpose(),
      np.dot(inv_sigma1,
      (test_datapoint - mu1))))+
      (np.dot((test_datapoint - mu2).
      transpose()
      ,np.dot(inv_sigma2,
      (test_datapoint - mu2)))))))

Yd=np.array(coordinates)
X=plot_data['x1']
Xd=np.array(X)

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D


x1=np.array(test_data['x1'])
y1=np.array(test_data['y1'])

#Parameters to set
mu_x = 0
variance_x = 3

mu_y = 0
variance_y = 15

#Create grid and multivariate normal
x = np.linspace(-6,6,100)
y = np.linspace(-6,6,100)
X, Y = np.meshgrid(x,y)
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X; pos[:, :, 1] = Y
rv = multivariate_norma
      l([0, 0], [[.25, .3], [.3, 1]])
rx = multivariate_norma
      l([2, 2], [[.5, 0], [0, .5]])
#Make a 3D plot

fig = plt.figure(figsize=(8,8))
ax= fig.add_subplot(111, projection="3d")
ax.plot_surface(X, Y, rv.pdf(pos),
cmap="viridis",
 w=0.5, rstride=1, cstride=1, alpha=0.5)

ax.plot_surface(X, Y, rx.pdf(pos),
cmap="cividis",
w=0.5, rstride=1, cstride=1, alpha=0.5)
```

```
cset1 = ax.contour(X, Y, rv.pdf(pos),
zdir='z', offset=0, cmap=cm.coolwarm)

cset2 = ax.contour(X, Y, rx.pdf(pos),
 zdir='z', offset=0, cmap=cm.coolwarm)

b=test_data.shape
print(b[0])
RowNumber=b[0]

class_1_x=[]
class_1_y=[]
class_2_x=[]
class_2_y=[]
for i in range(0,RowNumber):
  if (test_data.at[i,'class']==1):

    class_1_x.append
        (test_data.at[i,'x1'])
    class_1_y.append
        (test_data.at[i,'y1'])

  else:
    class_2_x.append
        (test_data.at[i,'x1'])
    class_2_y.append
        (test_data.at[i,'y1'])


class_1_x=np.array(class_1_x)
class_1_y=np.array(class_1_y)
class_2_x=np.array(class_2_x)
class_2_y=np.array(class_2_y)

ax.plot3D(class_1_x,class_1_y,-0.2,
'b*',
markersize=10,label='Train Class 1')
ax.plot3D(class_2_x,class_2_y,-0.2,
 'r*',
markersize=10,label='Train Class 2')


ax.text(class_1_x[0], class_1_y[0],
-0.2, "R1", size=8,color='k')
ax.text(class_1_x[1], class_1_y[1],
-0.2, "R1", size=8, color='k')
ax.text(class_1_x[2], class_1_y[2],
-0.2, "R1", size=8, color='k')
ax.text(class_2_x[0], class_2_y[0],
-0.2, "R2", size=8, color='k')
ax.text(class_2_x[1], class_2_y[1],
-0.2, "R2", size=8,color='k')
ax.text(class_2_x[2], class_2_y[2],
-0.2, "R2", size=8, color='k')
ax.plot(Xd,Yd,0,color='black')
ax.set_xlim(-6, 6)
ax.set_ylim(-6, 6)
ax.set_zlim(-0.2, 0.3)

ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
#ax.set_zlabel('Probability Density')
plt.show()
```

V. Conclusion

'Minimum Error Rate Classifier' is a simple classification technique which uses probabilistic method to classify sample points. This algorithm works with the help of **Gaussian normal distribution** and assumes the distribution of a pattern. From the distribution it predicts the class of data.