Implementing Minimum Error Rate Classifier

Ahmad Subaktagin Jabir
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204061@aust.edu

***Abstract*—The task of this experiment is to classify some sample points using the posterior probabilities. It uses Gaussian Distribution to calculate the likelihood probabilities. The main goal of this classifier is to decrease the rate of error during classification. This classifier is also known as Bayes Classifier with Minimum Error.**

***Index Terms*—Discriminant Function, Likelihood Probabilities Ratio, Posterior Probability, Pattern Recognition, Bayesian Classifier**

I. Introduction

Minimum error rate classifier is a classifier which is used for reducing the error rate. We are given six sample datas to complete this experiment. We have to classify those sample datas. A sample's likelihood probabilities are given by the normal distribution. Normal distribution can be expressed by two parameters - Σ (Sigma) and μ (Mean). Those parameters are given for this experiment.

As Bayesian classifier works with posterior probabilities, the decision rule is given below:

$$\text{If } p(\omega_1|x) > p(\omega_2|x) \text{ then } x \in \omega_1$$
$$\text{If } p(\omega_1|x) < p(\omega_2|x) \text{ then } x \in \omega_2$$

We can calculate posterior probabilities with the help of likelihood. It is given below:

$$\begin{aligned} & P(\omega_i|x) = P(x|\omega_i) * P(\omega_i) \\ \Rightarrow & lnP(\omega_i|x) = lnP(x|\omega_i) * P(\omega_i) \\ \Rightarrow & lnP(\omega_i|x) = lnP(x|\omega_i) + lnP(\omega_i) \end{aligned}$$

Discriminant function for minimum error rate classifier is denoted in this equation. $P(x|w_i)$ means likelihood probabilities and $P(w_i)$ means prior probabilities.

We have to use following equation as our dataset is 2D:

$$N_k(x_i|\mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^D|\Sigma_k|}} \exp\left[-\frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k)\right]$$

Here, N_k is normal distribution, Σ is covariance matrix, μ_k is mean, x_i is test data and it is a vector, $d = 2$ as our all datas are 2D. We will plot all test points in these equations and from this equation we will get likelihood probability values. After that we will multiply likelihood probability values with prior probabilities. By this we will get posterior probabilities.

And for determining decision boundary we will take the following solution:

$$\begin{aligned} & g_1(x) = g_2(x) \\ \Rightarrow & P(\omega_1|x) = P(\omega_2|x) \\ \Rightarrow & P(\omega_1|x) - P(\omega_2|x) = 0 \\ \Rightarrow & P(x|\omega_1).P(\omega_1) - P(x|\omega_2).P(\omega_2) = 0 \end{aligned}$$

Now taking ln

$$\begin{aligned} & lnP(x|\omega_1).P(\omega_1) - lnP(x|\omega_2).P(\omega_2) = 0 \\ \Rightarrow & lnP(x|\omega_1) + lnP(\omega_1) - lnP(x|\omega_2) - lnP(\omega_2) = 0 \\ \Rightarrow & \frac{lnP(x|\omega_1)}{lnP(x|\omega_2)} + \frac{lnP(\omega_1)}{lnP(\omega_2)} = 0 \end{aligned}$$

This is the equation of a decision boundary for minimum error classifier.

II. Experimental Design / Methodology

A. **Classifying the Sample Points:** First of all we will classify all our datas as they are not classified. We will classify them by using the 2 Posterior Probabilities . We are given 2 covariance matrices and 2 mean vectors for classifying them. With help of them at first we will calculate Normal Distribution values for all data points and then we will multiply the values with prior. For 2 Gaussian Distributions we will compare between them. For example, 1st Gaussian Distribution is denoted by $g_1(x)$ and 2nd Gaussian Distribution is denoted by $g_2(x)$. If $g_1(x) > g_2(x)$, then the sample point belongs to the Gaussian Distributions' Corresponding region. This condition denotes that sample point likelihood probabilities are close to the used Gaussian Distribution. So we can assign this sample point to that region.

B. **Plotting the sample points with different markers:** After classifying all sample data points I used the same color and same marker to distinguish one class from another. After plotting all points, I got something like following:

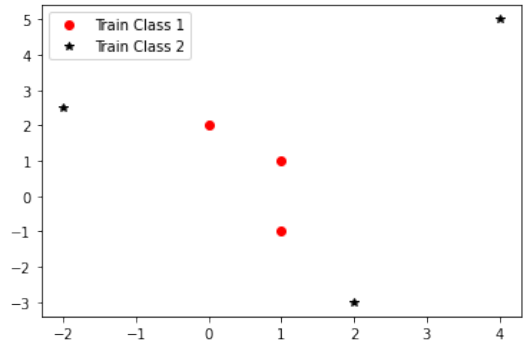

Fig. 1. Plotting The Sample Data Points

After drawing the sample points to their intended class, we need to draw a decision limit to split the whole space into two regions.

C. **Drawing Decision Boundary in Contour Plot:** Firstly, We have to obtain the equation of decision boundary which is $g_1(x) - g_2(x) = 0$. We will draw the decision boundary in Contour plotting. To draw the Contour plotting, we need to take the X and Y dimension to a higher dimension. We need to distribute our 2-dimensional distribution over variables X and Y. We will pack X and Y dimension into 3 dimension data. We can plot the values in Contour Plotting from this data. For Z values, we can pass the values in multivariate distribution. We will calculate the Z values for different classes. From it, we will calculate the decision boundary $db = Z - Z1$. After that we will draw the decision boundary.

III. Result Analysis

1) **Classifying the Sample Points:** After classifying I got following results:

```
Class 1:  [[ 1.  1.]
 [ 1. -1.]
 [ 0.  2.]]
Class 2:  [[ 4.   5. ]
 [-2.   2.5]
 [ 2.  -3. ]]
```

Fig. 2. Classified Sample Data Points

2) **Contour Plotting:** After plotting sample points, the corresponding probability distribution function along with its contour, I got the following figure:

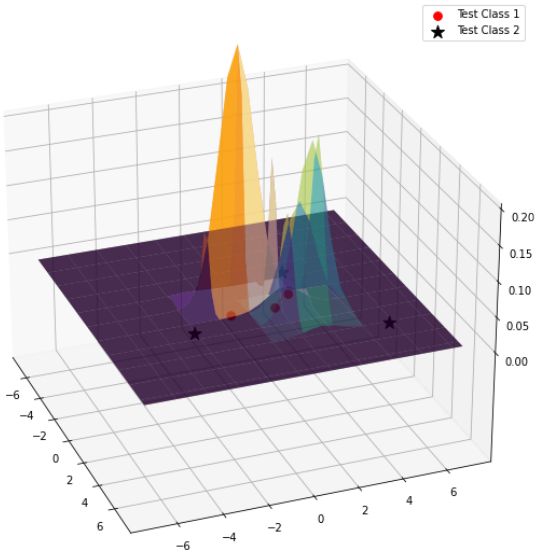

Fig. 3. Contour Plotting with Probability Distribution Function

3) **Drawing Decision Boundary:** After drawing decision boundary in the Contour, I got following figure:

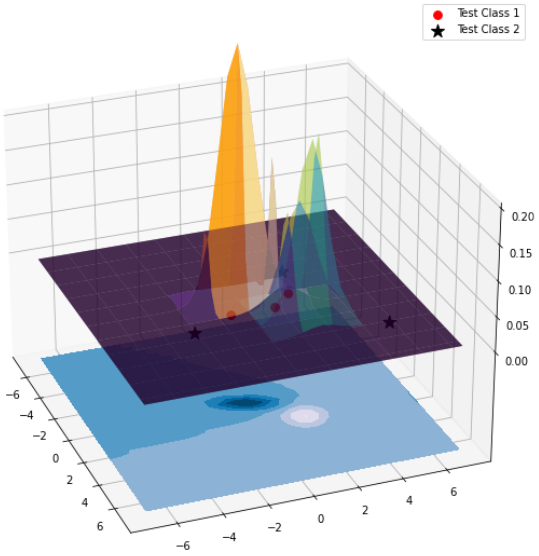

Fig. 4. Decision Boundary plotting in Contour

We know that the Minimum Error Rate Classifier tries to reduce the error. If we change the parameter of Gaussian Distribution, sample values' classes may be shifted to another class as likelihood probabilities can also be shifted towards another class.

IV. Conclusion

We understood the procedure of Minimum Error Rate Classifier in this experiment. Also, we came to know what Sigma and the Mean of Gaussian Distribution are. There are some disadvantages also in this classifier. This classifier fully depends on probability. It should be understood regarding the Gaussian Distribution.

V. Algorithm Implementation / Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv
from sympy import *
import math

train_set = []
with open('test-Minimum-Error-Rate-Classifier.txt','
    r') as file:
    new_reader = csv.reader(file,delimiter=',')
    for row in new_reader:
        train_set.append(row)

for i in range(len(train_set)):
    for j in range(len(train_set[i])):
        train_set[i][j] = float(train_set[i][j])

x = []
for train in train_set:
    x.append([train[0], train[1]])
x=np.array(x)
x

mu1=np.array([0,0])
sigma1 = np.array([[.25,.3],[.3,1]])
p_class1 = 0.5

mu2 = np.array([2, 2])
sigma2 = np.array([[.5,0],[0,.5]])
p_class2 = 0.5


# ### Classifying All Sample Points

class1 = []
class2 = []

for i in range(len(x)):
    d1 = x[i,:]-mu1
    a1 = 1 / (((2 * np.pi) ** (2/2)) * (np.linalg.
    det(sigma1)**(1/2)))
    expon1 = np.exp(-(1/2) * np.matmul(np.transpose(
    d1), np.matmul(np.linalg.inv(sigma1), d1)))
    likelihood1 = a1*expon1
    g1 = likelihood1*p_class1

    d2 = x[i,:]-mu2
    a2 = 1 / (((2 * np.pi) ** (2/2)) * (np.linalg.
    det(sigma2)**(1/2)))
    expon2 = np.exp(-(1/2) * np.matmul(np.transpose(
    d2), np.matmul(np.linalg.inv(sigma2), d2)))
    likelihood2 = a2*expon2
    g2 = likelihood2*p_class2

    if g1>g2:
        class1.append(x[i,:])
    else:
        class2.append(x[i,:])

class1 = np.array(class1)
class2 = np.array(class2)
print("Class 1: ", class1)
print("Class 2: ", class2)


# ### Plotting All Sample Points

plt.plot(class1[:,0:1],class1[:, 1:], linestyle = ''
    , marker='o', color='r', label ="Train Class 1")
plt.plot(class2[:,0:1],class2[:, 1:], linestyle = ''
    , marker = '*', color = 'k', label ="Train Class
     2")
plt.legend()
plt.show()


# ### Contour Plotting

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

# Our 2-dimensional distribution will be over
    variables X and Y
N = 32
X = np.linspace(-7, 7, N)
Y = np.linspace(-7, 7, N)
X, Y = np.meshgrid(X, Y)

# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y
# print(X)

def multivariate_gaussian(pos, mu, Sigma):
    n = mu.shape[0]
    Sigma_det = np.linalg.det(Sigma)
    Sigma_inv = np.linalg.inv(Sigma)
    N = np.sqrt((2*np.pi)**n * Sigma_det)
    fac = np.einsum('...k,kl,...l->...', pos-mu,
    Sigma_inv, pos-mu)
    return np.exp(-fac / 2) / N

Z1 = multivariate_gaussian(pos, mu1, sigma1)
# print(mu.shape[0])
Z2 = multivariate_gaussian(pos, mu2, sigma2)
# Create a surface plot and projected filled contour
     plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')

ax.scatter(class1[:,0], class1[:,1], marker='o', c='
    red',s=60,label='Test Class 1',alpha= 1.0)
ax.scatter(class2[:,0], class2[:,1], marker='*', c='
    black',s=160,label='Test Class 2',alpha= 1.0)


ax.plot_surface(X, Y, Z1, rstride=3, cstride=3,
    linewidth=1, antialiased=True,
                cmap=cm.inferno, alpha= .7)
ax.plot_surface(X, Y, Z2, rstride=3, cstride=3,
    linewidth=1, antialiased=True,
                cmap=cm.viridis, alpha= .4)

ax.legend()

# Adjust the limits, ticks and view angle
ax.set_zlim(-0.15,0.2)
ax.set_zticks(np.linspace(0,0.2,5))
ax.view_init(27, -21)

plt.rcParams["figure.figsize"] = (20,10)
plt.show()
```

```
# ### Drawing Decision Boundary

# Our 2-dimensional distribution will be over
    variables X and Y
N = 32
X = np.linspace(-7, 7, N)
Y = np.linspace(-7, 7, N)
X, Y = np.meshgrid(X, Y)

# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y
# print(X)

def multivariate_gaussian(pos, mu, Sigma):
    n = mu.shape[0]
    Sigma_det = np.linalg.det(Sigma)
    Sigma_inv = np.linalg.inv(Sigma)
    N = np.sqrt((2*np.pi)**n * Sigma_det)
    fac = np.einsum('...k,kl,...l->...', pos-mu,
    Sigma_inv, pos-mu)
    return np.exp(-fac / 2) / N

Z1 = multivariate_gaussian(pos, mu1, sigma1)
# print(mu.shape[0])
Z2 = multivariate_gaussian(pos, mu2, sigma2)
# Create a surface plot and projected filled contour
     plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')
db=Z1-Z2

ax.scatter(class1[:,0], class1[:,1], marker='o', c='
    red',s=60,label='Test Class 1',alpha= 1.0)
ax.scatter(class2[:,0], class2[:,1], marker='*', c='
    black',s=160,label='Test Class 2',alpha= 1.0)


ax.plot_surface(X, Y, Z1, rstride=3, cstride=3,
    linewidth=1, antialiased=True,
                cmap=cm.inferno, alpha= .7)
ax.plot_surface(X, Y, Z2, rstride=3, cstride=3,
    linewidth=1, antialiased=True,
                cmap=cm.viridis, alpha= .4)
ax.contourf(X, Y, db, zdir='z', offset=-0.15, cmap=
    cm.PuBu )

ax.legend()

# Adjust the limits, ticks and view angle
ax.set_zlim(-0.15,0.2)
ax.set_zticks(np.linspace(0,0.2,5))
ax.view_init(27, -21)

plt.rcParams["figure.figsize"] = (20,10)
plt.show()
```