Implementing Minimum Error Rate Classifier

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

***Abstract*—The main objective of this experiment is to determine the classes of the given sample points by designing a minimum error rate classifier.**

***Index Terms*—Bayesian classifier, Minimum error classifier, Normal distribution, Gaussian distribution, Multi variant.**

I. Introduction

In case of classification problems, minimum error rate classifier is a technique which minimizes the error rate while predicting sample points/data. It reduces the error rate but at the same time increases the posterior probability with respect to the class it belongs to. In case of general loss function the risk factor we have is:

$$g_i(x) = R(\alpha_i|X) \tag{1}$$

$$R(\alpha_i|X) = 1 - P(\omega_i|X) \tag{2}$$

The minimum error classifier uses a decision rule which is based on Bayesian classifier and posterior probability as follows:

$$\begin{aligned} if, P(\omega_1|x) > P(\omega_2|x), x \in \omega_1 \\ else, P(\omega_1|x) < P(\omega_2|x), x \in \omega_2 \end{aligned} \tag{3}$$

The posterior probabilities can be written as likelihood probability*prior probability based on Bayesian classifier as follows:

$$P(\omega_i|x) = p(x|\omega_i) * P(\omega_i) \tag{4}$$

The maximum discriminant function corresponds to the posterior probability i.e

$$g_i(x) = P(\omega_i|x) \tag{5}$$

So, from equation (4) and (5) we can write the discriminant function,

$$g_i(x) = p(x|\omega_i) * P(\omega_i) \tag{6}$$

In order to calculate the likelihood probability of the 2D data points given in the sample text, we use normal distribution equation as follows:

$$N_k(x_i|\mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^D|\Sigma_k|}} e^{(-\frac{1}{2}(x_i-\mu_k)^T\Sigma^{-1}(x_i-\mu_k))} \tag{7}$$

In the equation N_k is the normal distribution, μ_k is the mean, Σ_k is the co-variance matrix and x_i represents the 2 dimensional sample points. After calculating the likelihood probability we multiply it with the prior probability in order to get the posterior probability.

II. Experimental Design / Methodology

Let us now look at the different steps taken to solve conduct this experiment.

A. Classification of the Sample Points

The test dataset contains 6 sample points each being a 2 dimensional vector. At first we read the text file *test-Minimum-Error-Rate-Classifier.txt* which contains the x and y points without specifying the class it belongs to. We store these values and proceed forward.

B. Plotting the Given Sample Points

After loading the text file I now have to classify the sample points. This is done using the decision rule given in equation(3). But in order to do so, I have to at first calculate the likelihood probability using normal/Gaussian's distribution. For that I use equation(7) and determine the result. Next, as the prior probability values are given, we multiply both the likelihood and prior probability and calculate the discriminant function for both the classes for each sample point. If $P(\omega_1|x) > P(\omega_2|x)$ then the sample point belongs to class 1 and if $P(\omega_1|x) < P(\omega_2|x)$ the sample point belongs to class 2. We then use the python package '**matplotlib**' to plot the predicted sample points using red and black markers to differentiate them.

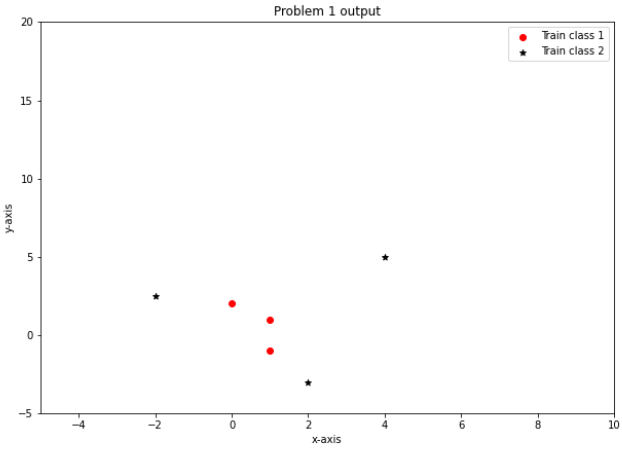

Fig. 1. Plotting Training Sample Points

C. Drawing the Decision Boundary in Contour Plot

To draw the decision boundary the following calculations has to be done:

$$g_1(x) = g_2(x)$$
$$\Rightarrow P(\omega_1|x) = P(\omega_2|x)$$
$$\Rightarrow p(x|\omega_1) * P(\omega_1) = p(x|\omega_2) * P(\omega_2)$$
$$\Rightarrow p(x|\omega_1) * P(\omega_1) - p(x|\omega_2) * P(\omega_2) = 0$$

Taking ln,

$$\Rightarrow lnp(x|\omega_1) * P(\omega_1) - lnp(x|\omega_2) * P(\omega_2) = 0$$
$$\Rightarrow lnp(x|\omega_1) + lnP(\omega_1) - lnp(x|\omega_2) - lnP(\omega_2) = 0$$

$$\Rightarrow ln(\frac{p(x|\omega_1)}{p(x|\omega_2)}) - ln(\frac{P(\omega_2)}{P(\omega_1)}) = 0 \quad (8)$$

Since all the points are given in 2 dimension if we want to plot them in a contour plot we have to take the points to a higher dimension as contour plot has to be in 3D. Equation (8) is used to determine the decision boundary between the classes. The x and y values are inserted in a 3D distribution. For Z values, the values are passed through multivariate normal distribution for 2 different class. The decision boundary is calculated subtracting the z points for class 1 and 2. Finally 2 bell shaped curves are produced with contour and the decision boundary for each class is drawn.

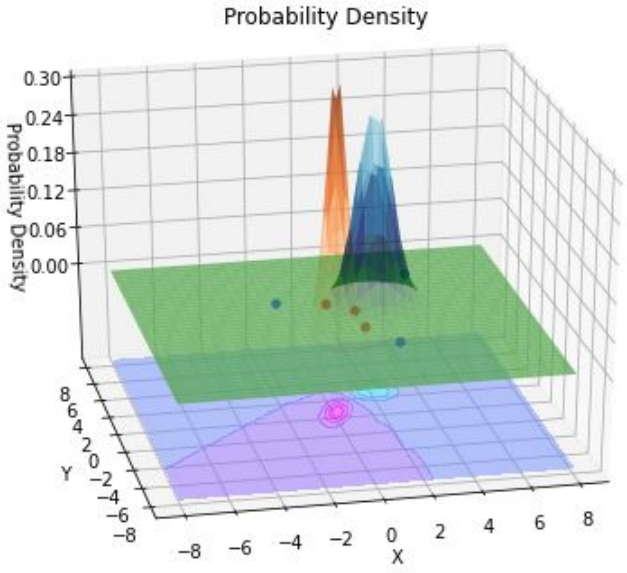

Fig. 2. Contour plot with probability distribution function

III. Result Analysis

Minimum error rate classifier tries to minimize the the maximum possible overall risk or error but at the same time tries to increase the posterior probability for a specific class. If the mean and variance is changed then the decision boundary can be shifted and the accuracy to predict the correct class for a given point may be reduced or increased. Taking all the parameters in concern the minimum error rate classifier tries to draw a decision boundary along with the contour which is the region where the function has constant density.

IV. Conclusion

In conclusion we have learned how to implement the minimum error rate classifier for different points with respect to the given mean and variance values using Gaussian multivariate distribution. This classifier depends more on the probability which can be a drawback.

V. Algorithm Implementation / Code

```
# -*- coding: utf-8 -*-
"""PR_Assignment3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
    iJ9YhP5QA_WXnAPwviXFNC2ilXROVnnX
"""

'''Problem 1 '''
import numpy as np
train_dataset = np.loadtxt('/content/drive/MyDrive
    /4.2/pattern /test-Minimum-Error-Rate-Classifier
    .txt',delimiter=",",dtype='float64')
print(train_dataset)
#Given,
w1=w2=0.5
cov1=np.array([[.25,.3],[.3,1]])
cov2=np.array([[.5,.0],[0,.5]])
mean1=np.array([0,0])
mean2=np.array([2,2])
print(mean1,w1,w2)
print( cov1, cov2)

D = mean1.shape[0]
def normal_distribution(mean, cov, x):
    cov_det = np.linalg.det(cov) #finding the
    determinant of covarience
    cov_inv = np.linalg.inv(cov) #finding the
    inverse of covarience
    N = np.sqrt((2*np.pi)**D*cov_det)
    xu_T=  np.transpose(x - mean) #(x-mu)T
    xu = (x- mean) #(x-mu)
    mul  = 0.5 * (np.dot(xu_T,cov_inv).dot(xu))
    expo= np.exp(-mul / 2)
    result= expo/ N
    return result

x1=[]
y1=[]
x2=[]
y2=[]
for train in  train_dataset:
    post1=w1*normal_distribution(mean1,cov1,np.array
    ([train[0],train[1]]))
    post2=w2*normal_distribution(mean2,cov2,np.array
    ([train[0],train[1]]))
    print( post1, post2)
    if( post1< post2):
        x2.append(train[0])
        y2.append(train[1])
    else:
        x1.append(train[0])
        y1.append(train[1])
print("Class1",x1,y1)
```

```
print("Class2",x2,y2)
import pandas as pd
import matplotlib.pyplot as plt
fig=plt.figure(1, figsize=(10,7))
plt.title("Problem 1 output")
plt.xlabel('x-axis', color='black')
plt.ylabel('y-axis', color='black')
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()#
#plt.savefig('TrainClass.png')

D = mean1.shape[0]
def gaussian_distribution1(mean, cov, x):
  cov_det = np.linalg.det(cov) #finding the
    determinant of covarience
  cov_inv = np.linalg.inv(cov) #finding the inverse
    of covarience
  N = np.sqrt((2*np.pi)**D*cov_det)  # 2*pi^D*determ
  mul = np.einsum('...i,ij,...j->...', x-mean,
    cov_inv, x-mean) #explicit mode summation:
    desired summation
  expo= np.exp(-mul / 2)
  result= expo/ N
  return result

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
X = np.linspace(-8, 8, 40)
Y = np.linspace(-8,8 , 40)
X, Y = np.meshgrid(X, Y)
x= np.empty(X.shape + (2,))
x[:, :, 0] = X
x[:, :, 1] = Y
Z = gaussian_distribution1( mean1, cov1,x)
Z1 = gaussian_distribution1( mean2, cov2,x)
fig = plt.figure()
ax = fig.gca(projection='3d')
fig.set_figheight(6)
fig.set_figwidth(8)
z=0
ax.scatter(x1,y1,color='red',alpha=1)
ax.scatter(x2,y2,color='blue',alpha=1)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.Oranges,
    alpha=.5)
ax.plot_surface(X, Y, Z1, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.ocean,
    alpha=.5)
ax.set_title('Probability Density')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Probability Density')
ax.set_zlim(-0.15,0.2)
ax.set_zticks(np.linspace(.30,0.0,6))
ax.view_init(27, -102)
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
X = np.linspace(-8, 8, 40)
Y = np.linspace(-8,8 , 40)
X, Y = np.meshgrid(X, Y)
x= np.empty(X.shape + (2,))
x[:, :, 0] = X
x[:, :, 1] = Y
Z = gaussian_distribution1( mean1, cov1,x)
Z1 = gaussian_distribution1( mean2, cov2,x)
db=(Z-Z1)
fig = plt.figure()
ax = fig.gca(projection='3d')
fig.set_figheight(6)
fig.set_figwidth(8)
z=0
ax.scatter(x1,y1,color='red',alpha=1)
ax.scatter(x2,y2,color='blue',alpha=1)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.Oranges,
    alpha=.5)
ax.plot_surface(X, Y, Z1, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.ocean,
    alpha=.5)
ax.contourf(X, Y, db, zdir='z', offset=-.15,cmap=cm.
    cool,alpha=0.5)
ax.set_title('Probability Density')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Probability Density')
ax.set_zlim(-0.15,0.2)
ax.set_zticks(np.linspace(.30,0.0,6))
ax.view_init(27, -102)
plt.show()
```