Implementing Minimum Error Rate Classifier

Anika Tanzim
Dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204072@aust.edu

***Abstract*—The objective of this experiment is to implement minimum error rate classifier. This classifier is also recognized as the Bayes classifier with minimum error. The purpose of this classifier is to reduce the error rate using the posterior probabilities by Gausian distribution.**

***Index Terms*—Gausian Distribution, Contour graph, Coverience, Mean, Normal Distribution.**

I. Introduction

The minimum error rate classifier asks for a decision rule that reduces the probability of error which is the error rate. For general case with risks we have,

$$g_i(x) = R(\alpha_i|x)$$

For minimum error rate classifier we take,

$$R(\alpha_i|x) = 1 - P(\omega_i|x)$$

It says, the more probability increases, the more risk decreases. So, if we want to decrease the rate, we have to increase the probability. For this, the maximum discriminant function corresponds to maximum posterior probability. But we know that, as, Bayesian classifier works with posterior probabilities the decision rule is as follows:

$If\ P(w1|x) > P(w2|x)\ then\ x \in w1$

$If\ P(w1|x) < P(w2|x)\ then\ x \in w2$

We know, $P(\omega_i|x) = P(x|\omega_i) \times P(\omega_i)$ We can apply natural logrithm and get,

$$\ln P(\omega_i|x) = \ln P(x|\omega_i) \times P(\omega_i)$$

$$\ln P(\omega_i|x) = \ln P(x|\omega_i) + \ln P(\omega_i)$$

This is the discriminant function for minimum error rate classifier. Now the likelihood probability is the Gaussian distribution here and it is a multivariate normal, i.e. $P(x|\omega_i \sim N(\mu_i, \Sigma_i))$ The equation we are using here is,

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

Here, N_k is normal disctribution, Σ is co-variance matrix, μ_k is mean, x_i is test data which is a vector. And the decision boundary would be the solution of

$$g_1(x) = g_2(x)$$
$$=> P(\omega_1|x) = P(\omega_2|x)$$
$$=> P(\omega_1|x)\ \text{-}\ P(\omega_2|x) = 0$$
$$=> \frac{\ln P(x|w1)}{\ln P(x|w2)} - \frac{\ln P(w2)}{\ln P(w1)} = 0$$

II. Experimental Design / Methodology

A. *Classifying the sample points*

A dataset is given. We need to classify the sample points. As we have discussed earlier, We classify the data depending on the posterior probability. For this, Prior probabiliy is given. But Likelyhood probabilty needs to be calculated based on the given normal distribution function.

The mean vectors and the co-varience vectors for each two classes are given as follows:

$\mu 1 = [0\ 0]$ $\mu 2 = [2\ 2]$ $\Sigma 1 = [.25\ .3;\ .3\ 1]$ $\Sigma 2 = [.5\ 0;\ 0\ .5]$

Using this, we get two likelyhood probabilites for each class. Multiplying with prior, we get posterior i.e. $g_1(x)$ and $g_2(x)$. The sample points are classified depending on the larger value among them. The output is shown in figure 1.

B. *Plotting the sample points*

All the points are classified into two classes - Class 1 and Class 2. Using Matplotlib, all the points from Class 1 are plotted using blue dot marker and the points from Class 2 are plotted using green star marker in figure 2.

C. *Plotting the sample points and the corresponding probability distribution function along with its contour*

To make a diagram of the contour plotting, a higher dimension of X-Y is taken. We distribute the X and Y dimension into a 3 dimension data. Then we plot the sample points and the corresponding probability distribution function along with its contour. To get the likelyhood values, we pass the values in multivariate distribution and use it to make the contour graph. The contour graph is shown in figure3.

In figure 3, we can see two shapes like bell curves for each class. The top of the curve shows the mean value of the respected class.

D. *Drawing decision boundary*

To make the decision boundary, we have to calculate $g_1(x) - g_2(x) = 0$ and plot this to contour graph. To contour plotting, the decision boundary is calculated by taking the data in higher dimension where we get the likelyhood values. we calculate the decision boundary as the difference of the values.

The shadow shows the contour graph. As for class1, the co-varience shows the co-relation. So, the shadow is in oval shape. On the other hand, as for class2, the co-varience shows no co-relation. So, the shadow is in circle shape.

III. Result Analysis

From the graph, we see that for different mean and co-varience the graph can be different. So, by changing them and taking other sample points ,we can see the shifting of bell curves or the changing shapes in the shadow. The decision boudary can be changed as well. By that, points can be classified to the other class. This is how this classifier reduces the error rate.

The final output of this program is as follows-

```
Output 1:

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

Fig. 1. Task 1 Output: Sample points Classified

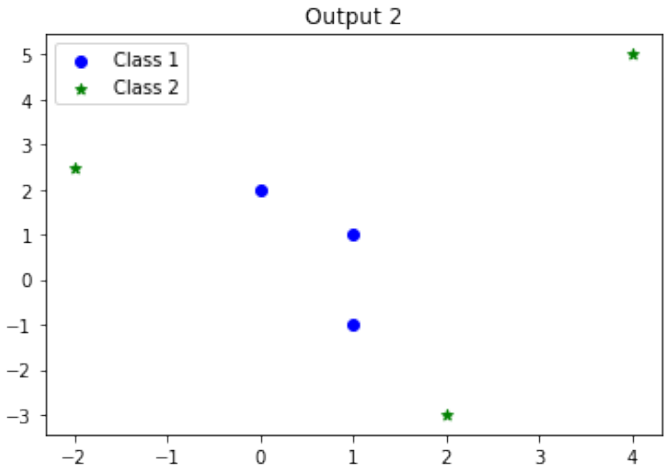

Fig. 2. Task 2 Output: Sample Dataset Points

IV. Conclusion

In this experiment we have implemented the minimum error rate classifier and see the effect of changing mean and sigma values. This classifier is dependent on probability. The more posterior probability increases, the more error rate decreases. So, it has some demerits too. If the Probabilities are not descent, the classifier cannot work well.

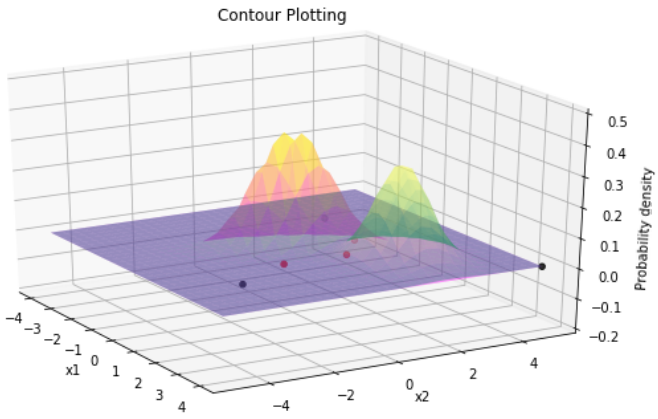

Fig. 3. Task 3 Output :Contour Plotting of Probability Distribution Function including sample points

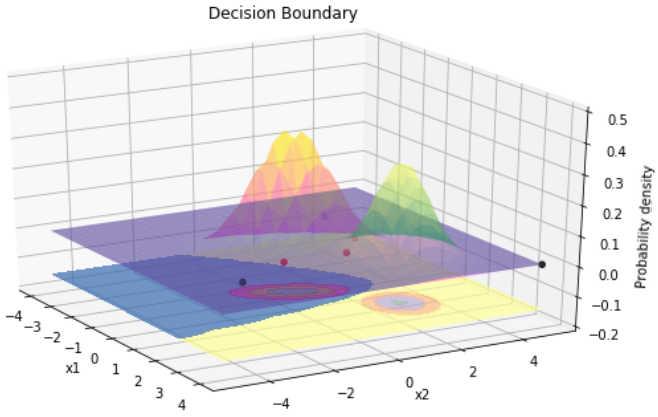

Fig. 4. Task 4 Output :Contour Plotting along with Decision Boundary

V. Algorithm Implementation / Code

```
##task 2##
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

dataset = np.loadtxt('test-Minimum-Error-Rate-
    Classifier.txt',delimiter=",",dtype='float64')

mean_1=np.array([0,0])
mean_2=np.array([2,2])
cov_1=np.array([ [.25,.3], [.3,1] ])
cov_2=np.array([ [.5,0], [0,.5] ])
prior_1=0.5
prior_2=0.5


def normal_distribution(x, mu, sigma):
    D = len(mu)

    a1 = (2 * np.pi)**D          # (2pie)^D
    a2 = np.linalg.det(sigma)    # |cov|
    a3 = 1.0 / np.sqrt(a1 * a2)
```

```
    a4 = np.transpose(x - mu)   #(x-mu)^T
    a5 = np.linalg.inv(sigma)   # covariance^-1
    a6 = (x - mu)               # (x-mu)
    a7=np.dot(a4,a5)
    a8 = -0.5 * (np.dot(a7,a6))

    a = a3 * np.exp(a8)


    return a


X_1=[]
Y_1=[]
X_2=[]
Y_2=[]
print("Output 1:\n")
for i in  dataset:
    x= np.array([i[0],i[1]])
    likelyhood_1 = normal_distribution(x,mean_1 ,
    cov_1)
    likelyhood_2 = normal_distribution(x,mean_2 ,
    cov_2)
    posterior_1= likelyhood_1* prior_1
    posterior_2= likelyhood_2* prior_2
    print(x, end =' ')
    if( posterior_1> posterior_2):
        print("class 1")
        X_1.append(x[0])
        Y_1.append(x[1])
    else:
        print("class 2")
        X_2.append(x[0])
        Y_2.append(x[1])


##task 2##

fig,ax=plt.subplots()
plt.title("Output 2")
ax.scatter(X_1, Y_1, marker='o',color='b',label='
    Class 1')
ax.scatter(X_2, Y_2, marker='*',color='g',label='
    Class 2')
ax.legend()


##task3##

# 2-dimensional distribution will be over variables
    X and Y

X = np.linspace(-4, 4, 30)
Y = np.linspace(-5, 5,30)

X,Y = np.meshgrid(X,Y) # (vector to matrix)

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

def Distribution_3D(data, mu, cov):
    D = mu.shape[0]
    a1 = np.linalg.det(cov)
    a2 = np.linalg.inv(cov)
    a3 = np.sqrt((2*np.pi)**D * a1)
    a4= np.einsum('...a,ab,...b->...', data-mu, a2,
    data-mu)
    a5= -0.5*a4
    a6 = np.exp(a5)
    a7 =a6/ a3
    return a7

Z = Distribution_3D(data, mu1, cov1)
Z1 = Distribution_3D(data, mu2, cov2)

# Create a surface plot and projected filled contour
     plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')

fig.set_figheight(6)
fig.set_figwidth(10)
ax.scatter(X_1, Y_1, 0,color='red')
ax.scatter(X_2, Y_2, 0,color='black')

ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.spring,
    alpha=.4)
ax.plot_surface(X, Y, Z1, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.summer,
    alpha=.4)

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Probability density')
ax.set_title('Contour Plotting')

ax.set_zlim(-0.2,0.3)
ax.set_zticks(np.linspace(-0.2,0.5,8))
ax.view_init(25, -30)
plt.show()


## task - 4
fig = plt.figure()
ax = fig.gca(projection='3d')

fig.set_figheight(6)
fig.set_figwidth(10)
ax.scatter(X_1, Y_1, 0,color='red')
ax.scatter(X_2, Y_2, 0,color='black')

ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.spring,
    alpha=.4)
ax.plot_surface(X, Y, Z1, rstride=1, cstride=1,
    linewidth=1, antialiased=True,cmap=cm.summer,
    alpha=.4)

db=Z-Z1 #decision boundary
ax.contourf(X, Y, db, zdir='z', offset=-0.15, cmap=
    cm.Accent,alpha=0.7)

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Probability density')
ax.set_title('Decision Boundary')
ax.set_zlim(-0.2,0.3)
ax.set_zticks(np.linspace(0.5,-0.2,8))
ax.view_init(25, -30)
plt.show()
```