Implementing Minimum Error Rate Classifier

Mayisha Farzana
dept.Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204028@aust.edu

***Abstract*—This experiment aims to identify specific sample points using the posterior probabilities used to measure the probability probabilities by Gaussian distribution. The goal of this classifier form is to decrease the rate of error during classification. So this classifier takes decisions based on the most posterior probabilities. This classifier is also known as the Bayes classifier with minimum error.**

***Index Terms*—Discriminant functions, pattern recognition,likelihood probabilities ratio, Bayesian classifier, posterior probability**

I. Introduction

A classifier is a minimal error rate classifier, and its function is to decrease the error rate. We are given six sample details in this experiment; we have to identify those. The normal distribution gives a sample's probability probabilities. With two parameters, sigma and mean, any normal distribution can be expressed. Those parameters are given in this experiment. As Bayesian classifier works with posterior probabilities the decision rule is as follows :

$$If\ p(w_1|x) > p(w_2)|x)\ then\ x\epsilon w_1$$

$$If\ p(w_1|x) < p(w_2)|x)\ then\ x\epsilon w_2$$

The posterior probabilities can be calculated with the help of likelihood probabilities. This can be written as:

$$P(w_i|x) = P(x|w_i).P(w_i)$$

$$=> lnP(w_i|x) = lnP(x|w_i).P(w_i)$$

$$=> lnP(w_i|x) = lnP(x|w_i) + P(w_i)$$

This equation gives the discriminant function for minimum error rate classifier.In here $P(x|w_i)$ is likelihood probabilities and $P(w_i)$ is prior probabilities.

As our dataset is in 2D, so we have to use the following equations:

$$N_k(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}_k|}} e^{\left(-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_k)\right)}$$

Here, N_k is normal disctribution, Σ is co-variance matrix, μ_k is mean, x_i is test data which is a vector, d=2 for our experiment because all the data are 2D. We will plot all the test points in these equations, and from this equation we will get the likelihood probability values. After getting the values, if we multiple the values with prior probabilities, we will get our required value of posterior probability. Posterior probability= likelihood probability* prior probability.

And the decision boundary would be the solution of

$$g_1(x) = g_2(x)$$

$$=> p(w_1|x) = p(w_2|x)$$

$$=> p(w_1|x) - p(w_2|x) = 0$$

$$=> P(x|w_1).P(w_1) - P(x|w_2).P(w_2) = 0$$

Taking ln,

$$=> lnP(x|w_1).P(w_1) - lnP(x|w_2).P(w_2) = 0$$

$$=> lnP(x|w_1) + lnP(w_1) - lnP(x|w_2) - lnP(w_2) = 0$$

$$=> lnP(x|w_1)/lnP(x|w_2) - lnP(w_2)/lnP(w_1) = 0$$

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

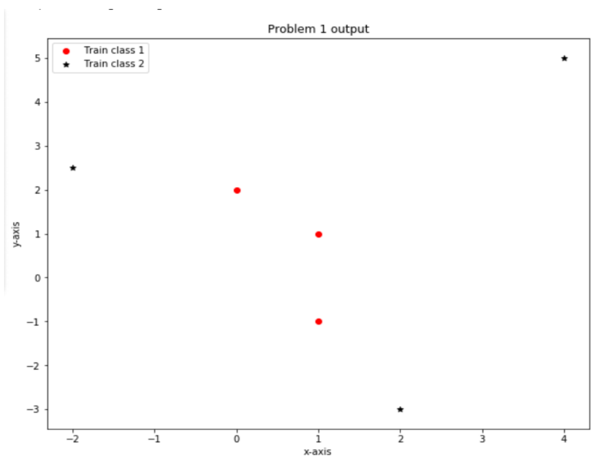
II. Experimental Design / Methodology

A.Classifying the sample points: At first, we will read the test.txt file. From this file, we will get the x vector values. As we can see that, the classes are not classified in the text dataset. So we need to classify it.

B. Plotting the sample points with different markers: In this part, for classifying the dataset class, we need to calculate the posterior probability values. We know that,

$$P(w_i|x) = P(x|w_i).P(w_i)$$

posterior probability=likelihood probability* prior probability As we know that, we can get the likelihood values from normal distribution. We will calculate the normal distribution values. Using this values and prior values, we can easily get the posterior probability values. From this values, we need to apply decision rule. We calculate the value of g(x) for each sample point with the two given Gaussian distributions and check for the following conditions $g_1(x) > g_2(x)$. If the above condition is true, then the sample point, x, belongs to the Gaussian distributions' corresponding regions. The value of g1(x) greater than g2(x) means the sample point likelihood probabilities close to the used Gaussian distribution to assign this sample point to that region. After plotting all the sample values the output is as follows-



Now that we are plotting all the samples 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 To draw the decision boundary we have to obtain the equation of the decision boundary $g_1(x) - g_2(x) = 0$ We will draw the decision boundary in contour plotting. To draw the contour plotting, at first we need to take the X,Y dimension to a higher dimension. We need to distribute our 2-dimensional distribution over variables X and Y. We will pack the X and Y dimension into a 3 dimension data. From this data, we can plot the values in contour plotting. For Z values, we will pass the values in multivariate distribution. We will calculate the Z values for different class. From it, we will measure the decision boundary db=Z-Z1. After that, we will plot the decision boundary line in the contour plotting. This is how we will draw the decision boundary.

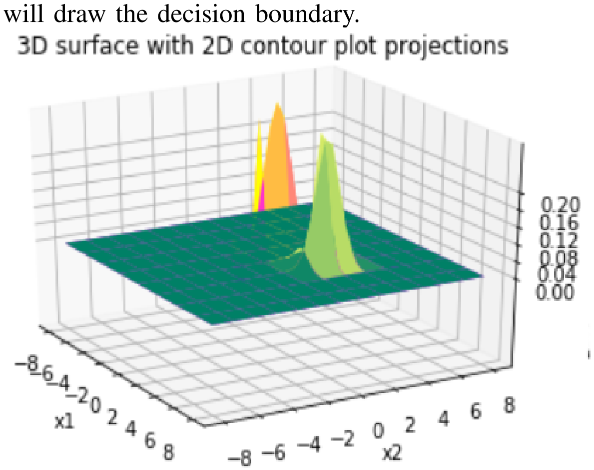


Fig: Contour plotting with probability probability distribution function

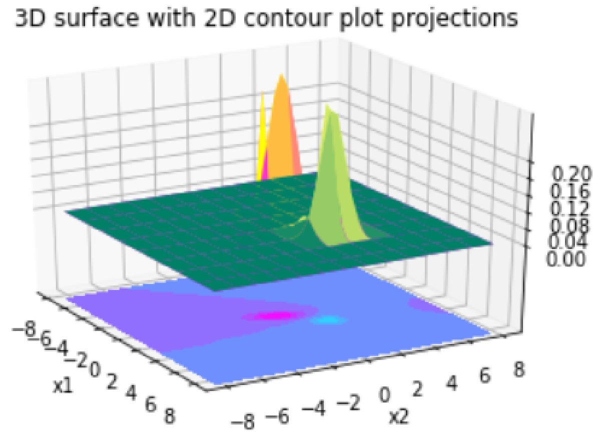


Fig: Contour plotting with decision boundary

III. Result Analysis

As we know that minimum error rate classifier tries to minimize the error. So if we change the parameter of Gaussian distribution the sample values also can be shifted to another class because the likelihood probabilities can also be shifted towards another class.

IV. Conclusion

In this experiment, we came to understand how a minimum error rate classifier functions and what it means by sigma and mean of Gaussian distribution. There are, however, some drawbacks of this classifier. This classifier relies solely on probability and should be understood regarding the Gaussian distribution.

V. Algorithm Implementation / Code

```
'''Problem 1 '''
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train_dataset = np.loadtxt('test-Minimum-Error-Rate-
    Classifier.txt',delimiter=",",dtype='float64')
print(train_dataset)
length=len(train_dataset)
print(length)
import numpy as np

def normal_distribution(x, u, cov):
    k = len(u)                    # scalar dimension
    uu = u.copy() #mu
    xx = x.copy() #x
    t1 = (2 * np.pi)**k            # scalar (2pie)^k
    t2 = np.linalg.det(cov)        # scalar   det(
    covariance)
    t3 = 1.0 / np.sqrt(t1 * t2)    # scalar

    t4 = np.transpose(xx - uu)            # (x-mu)T
    t5 = np.linalg.inv(cov)     # inverse(covariance)
    t6 = (xx - uu)              # (x-mu)
    t7 = -0.5 * (np.dot(t4,t5).dot(t6))
    result = t3 * np.exp(t7)    # 1x1
    return result
#Initialize
mu1=np.array([0,0]).reshape(2,)
mu2=np.array([2,2]).reshape(2,)
cov1=np.array([.25,.3,.3,1]).reshape(2,2)
cov2=np.array([.5,0,0,.5]).reshape(2,2)
```

```
p_w1=0.5
p_w2=0.5

x_class1=[]
y_class1=[]
x_class2=[]
y_class2=[]
for x in  train_dataset:
    posterior1=p_w1*normal_distribution(np.array([x[0],x[1]]),mu1 , cov1)
    posterior2=p_w2*normal_distribution(np.array([x[0],x[1]]),mu2 , cov2)
    print( posterior1, posterior2)
    if( posterior1> posterior2):
        print("class 1")
        x_class1.append(x[0])
        y_class1.append(x[1])
    else:
        print("class 2")
        x_class2.append(x[0])
        y_class2.append(x[1])
fig,ax=plt.subplots()#to show it in the same figure
plt.title("Problem 1 output")
plt.xlabel('x-axis', color='black')
plt.ylabel('y-axis', color='black')
ax.scatter(x_class1,y_class1,marker='o',color='r', label='Train class 1')
ax.scatter(x_class2,y_class2,marker='*',color='black',label='Train class 2')
fig.set_figheight(8)
fig.set_figwidth(10)
ax.legend()#show the output figure
'''Contour Plotting'''

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Our 2-dimensional distribution will be over variables X and Y
N = 40
X = np.linspace(-8, 8, N)#it divides the -8 to 8 range in to 40 piece
Y = np.linspace(-8, 8, N)
X, Y = np.meshgrid(X, Y)#Make N-D coordinate arrays for vectorized evaluations of N-D scalar
#print('X',X)
# Mean vector and covariance matrix
mu1 = np.array([0., 0.])
cov1 = np.array([[ .25 , 0.3], [.3,  1.]])
mu2 = np.array([2.,2.])
cov2 = np.array([[ .5 , 0.], [0.,  .5]])
# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))#taking X shape(40,40) and one additional dimension 2D points
print(pos.shape)
pos[:, :, 0] = X
pos[:, :, 1] = Y
def multivariate_gaussian(pos, mu, cov):
    """Return the multivariate Gaussian distribution on array pos."""
    dimen = mu.shape[0]
    cov_det = np.linalg.det(cov)
    cov_inv = np.linalg.inv(cov)
    N = np.sqrt((2*np.pi)**dimen * cov_det)
    # This einsum call calculates (x-mu)T.Sigma-1.(x-mu) in a vectorized
    # way across all the input variables.
    ans= np.einsum('...a,ab,...b->...', pos-mu, cov_inv, pos-mu)#EINSTEIN SUMMATION

    return np.exp(-ans / 2) / N
Z = multivariate_gaussian(pos, mu1, cov1)
Z1 = multivariate_gaussian(pos, mu2, cov2)
# Create a surface plot and projected filled contour plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')
#db=Z-Z1#decision boundary
z=0
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Probability density')
ax.scatter(x_class1,y_class1, z,color='red')
ax.scatter(x_class2,y_class2, z,color='blue')
#print(len(pos),pos.shape)
ax.plot_surface(X, Y, Z, rstride=3, cstride=5, linewidth=2, antialiased=True,
                cmap=cm.spring)
ax.plot_surface(X, Y, Z1, rstride=3, cstride=5, linewidth=2, antialiased=True,
                cmap=cm.summer)
#ax.contourf(X, Y, db, zdir='z', offset=-0.22, cmap=cm.cool)
ax.set_title('3D surface with 2D contour plot projections')
# Adjust the limits, ticks and view angle
ax.set_zlim(-0.2,0.3)
ax.set_zticks(np.linspace(0,0.2,6))
ax.view_init(25, -30)
plt.show()
'''Drawing Decision Boundary'''
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Our 2-dimensional distribution will be over variables X and Y
N = 40
X = np.linspace(-8, 8, N)#it divides the -8 to 8 range in to 40 piece
Y = np.linspace(-8, 8, N)
X, Y = np.meshgrid(X, Y)#Make N-D coordinate arrays for vectorized evaluations of N-D scalar
#print('X',X)
# Mean vector and covariance matrix
mu1 = np.array([0., 0.])
cov1 = np.array([[ .25 , 0.3], [.3,  1.]])
mu2 = np.array([2.,2.])
cov2 = np.array([[ .5 , 0.], [0.,  .5]])
# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))#taking X shape(40,40) and one additional dimension 2D points
print(pos.shape)
pos[:, :, 0] = X
pos[:, :, 1] = Y
def multivariate_gaussian(pos, mu, cov):
    """Return the multivariate Gaussian distribution on array pos."""
    dimen = mu.shape[0]
    cov_det = np.linalg.det(cov)
    cov_inv = np.linalg.inv(cov)
    N = np.sqrt((2*np.pi)**dimen * cov_det)
    # This einsum call calculates (x-mu)T.Sigma-1.(x-mu) in a vectorized
    # way across all the input variables.
    ans= np.einsum('...a,ab,...b->...', pos-mu, cov_inv, pos-mu)#EINSTEIN SUMMATION

    return np.exp(-ans / 2) / N
Z = multivariate_gaussian(pos, mu1, cov1)
Z1 = multivariate_gaussian(pos, mu2, cov2)
# Create a surface plot and projected filled contour plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')
db=Z-Z1#decision boundary
z=0
```

```
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Probability density')
ax.scatter(x_class1,y_class1, z,color='red')
ax.scatter(x_class2,y_class2, z,color='blue')
#print(len(pos),pos.shape)
ax.plot_surface(X, Y, Z, rstride=3, cstride=5, linewidth=2, antialiased=True,
                cmap=cm.spring)
ax.plot_surface(X, Y, Z1, rstride=3, cstride=5, linewidth=2, antialiased=True,
                cmap=cm.summer)
ax.contourf(X, Y, db, zdir='z', offset=-0.22, cmap=cm.cool)
ax.set_title('3D surface with 2D contour plot projections')
# Adjust the limits, ticks and view angle
ax.set_zlim(-0.2,0.3)
ax.set_zticks(np.linspace(0,0.2,6))
ax.view_init(25, -30)
plt.show()
```