Implementing Minimum Error Rate Classifier

Tahira Salwa Rabbi Nishat
Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204070@aust.edu

***Abstract*—We were given some data that includes a test data, mean, sigma and prior.We distributed the given data into two classes using the Minimum Error Rate Classifier. We calculated the likelihood probabilities using the Multivariate Normal Density Function.I plotted them along with the decision boundary in a 3D surface.**

***Index Terms*—Multivariate normal density function, Gaussian density function, normal deviate, plotting bell curve, Gaussian bell curve plotting.**

I. Introduction

Bayesian decision theory is important in statistics and are often used to represent real-valued random variables whose distributions are not known.It enables us to predict the error we will get when we generalize to novel patterns.The Bayes Theory helps to calculate the Posterior probability(which is an conditional probability) of an unknown variable with the help of Likelihood probability, Prior probability and Evidence.

Conditional Probability:Conditional probability is a measure of the probability of an event occurring, given that another event (by assumption, presumption, assertion or evidence) has already occurred.
Posterior Probability:Posterior is a probability density function, but it's normalised with respect to all possible parameter values
Likelihood Probability:The likelihood is a probability density function, normalised with respect to all possible data outcomes.
Prior Probability:A prior probability, in Bayesian statistical inference, is the probability of an event based on established knowledge, before empirical data is collected.

The formula to calculate a posterior probability of x belongs to class w :

$$P(w_i/x) = \frac{P(x/w_i) * P(w_i)}{P(x)}$$

Here, $P(w_i/x)$ is the posterior, $P(x/w_i)$ is likelihood, and $P(w_i)$ is prior and $P(x)$ is evidence for $i = 1, 2$ class no. It can be informally expressed as

$$Posterior = \frac{Likelihood \times prior}{Evidence}$$

From there we come to a Decision Rule which is:

if $P(w_1/x) > P(w_2/x), then x \in w_1$
if $P(w_1/x) < P(w_2/x), then x \in w_2$

The p.d.f. of a multivariate normal distribution x in d dimensions is given by:

$$g_i(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} \exp\left[-\frac{(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)}{2}\right] (1)$$

where x is a d-component column vector, μ is the d-component mean vector, Σ is dxd co-variance matrix, $|\Sigma|$ is its determinant and Σ^{-1} is its inverse. Taking $\ln$ on both sides the equation becomes:

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i) - \frac{d}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma_i| + \ln P(w_i)$$

From eq^n (1) formula we calculated $g_1(x)$ and $g_2(x)$.

$$g_1(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_1|}} \exp\left[-\frac{(x-\mu_1)^T \Sigma_1^{-1} (x-\mu_1)}{2}\right]$$

$$g_2(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_2|}} \exp\left[-\frac{(x-\mu_2)^T \Sigma_2^{-1} (x-\mu_2)}{2}\right]$$

The decision boundary is the difference between them.

$$g(x) = g_2(x) - g_1(x)$$

II. Experimental Design / Methodology

We were given a data set,the value of Σ_1, Σ_2, μ_1, μ_2.
The steps I have followed for this experience are:

1) I have loaded the train file and test file using loadtxt() method from numpy library.
2) Then i initialized Σ_1, Σ_2, μ_1, μ_2 with the given values.

$$\Sigma_1 = \begin{bmatrix} 0.25 & 0.3 \\ 0.3 & 1 \end{bmatrix}, \mu_1 = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$\Sigma_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \mu_2 = \begin{bmatrix} 2 & 2 \end{bmatrix}$$

3) From the Normal Distribution Function I distributed the data into two classes: class1 and class2.
4) Then I plotted them in 2D surface using matplotlib.pyplot in python.

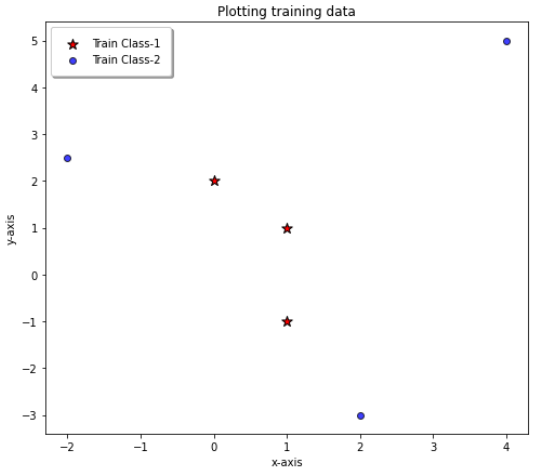

Fig. 1. Plotting Data

5) Then I drew a 3D plane including data points, the corresponding probability distribution function along the help of cm matplotlib specifically using surface and contour plotting.
6) Finally, I drew the Decision boundary. It is essentially a line between the two contour.

III. Result Analysis

The minimum error rate classifier tries to minimize the error. I drew a 3D plane including data points, the corresponding probability distribution function specifically using surface and contour plotting. the result is :

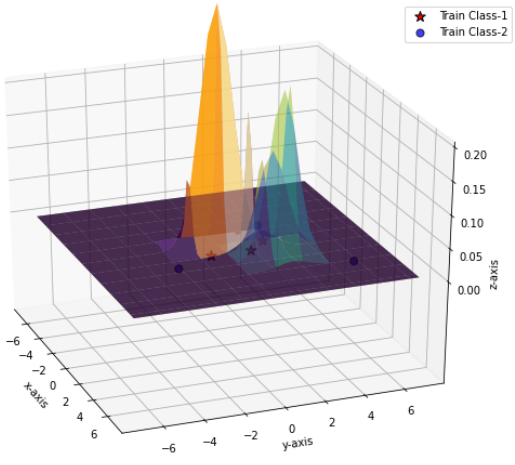

Fig. 2. p.d.f with data points,contour

After the when the decision boundary was added which is the difference between the two contours, the result becomes:

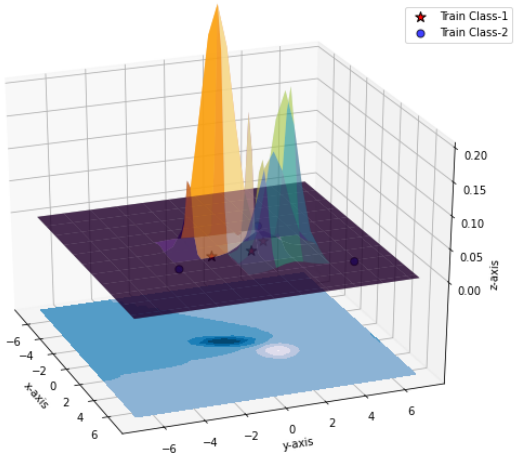

Fig. 3. p.d.f with data points,contour and decision boundary

If we change the parameters of the Gaussian distribution, the sample value can be shifted to another class to minimize the error.

IV. Conclusion

From this experiment we have understood this better specially from seeing how the points got plotted on the 3D surface and the how contour line worked.The decision boundary gave a better experimental idea on how the Bayes Theory helped to classify those points.

V. Algorithm Implementation / Code

```
# -*- coding: utf-8 -*-
"""160204070_Assignment3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
    fWcrvcokmzx2pXmzKIwkTTDjTokxL30k
"""

import pandas
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

from google.colab import files
uploaded = files.upload()

#Reading Train and Test data files
import numpy as np
df_train = np.loadtxt("test-Minimum-Error-Rate-
    Classifier.txt", dtype= float, delimiter = ",")
df=df_train

#initializing sigma and mean
x=np.array([1,1])
mu1=np.array([0,0])
Sigma1=np.matrix([[.25 ,.3],[.3 ,1]])

mu2=np.array([2,2])
Sigma2=np.matrix([[.5 ,0],[0 ,.5]])

w1=np.empty(shape=[0,2])
```

```
w2=np.empty(shape=[0, 2])

#Normal Distribution Formula after taking log on its
     both side
def normalDistribution(x,y,u,sigma,w):
    return  -np.log(2*np.pi)-0.5*np.log(np.linalg.
    det(sigma))-0.5*(np.dot(np.dot(np.transpose([x,y
    ]-u),np.linalg.inv(sigma)),([x,y]-u)))+np.log(w)

#Classify samples into classes
for i in range(0,df.shape[0]):
    print(df[i,:]);
    x1 = normalDistribution(df[i,0],df[i,1], mu1,
    Sigma1,0.5);
    x2 = normalDistribution(df[i,0],df[i,1], mu2,
    Sigma2, 0.5);
    if(x1 > x2):
        w1 = np.append(w1,np.array([df[i, :]]), axis
    =0)
    else:
        w2 = np.append(w2, np.array([df[i, :]]),
    axis=0)

print(w1)
print(w2)
class1 = np.insert(w1, 2, values=1, axis=1)
class2 = np.insert(w2, 2, values=2, axis=1)
print(class1)
print(class2)

#plotting 3D
fig = plt.figure()
ax =fig.add_subplot(projection='3d')

ax.set_xlim3d(7,-7)
ax.set_ylim3d(-7,7)
ax.set_zlim3d(-0.3,0.3)

ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
ax.set_zlabel("z-axis")


ax.scatter3D(w1[:,0],w1[:,1],[-.3,-.3,-.3], c = 'red
    ',marker='*',label='w1', s=100,edgecolors='black
    ')
ax.scatter3D(w2[:,0],w2[:,1],[-.3,-.3,-.3], c = '
    blue',label='w2',edgecolors='black', alpha=0.75,
     s=50)

#plotting 2d
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, figsize = (8, 7))


plt.scatter(class1[:,0], class1[:,1], c = 'red',
    marker='*', s= 100, label='Train Class-1',
    edgecolors='black')
plt.scatter(class2[:,0], class2[:,1], c = 'blue',
    label='Train Class-2', edgecolors='black', alpha
    =0.75)

plt.title("Plotting training data")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
ax.legend(frameon=True, shadow=True, borderpad=1)


plt.show()

#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


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

#a figure which should include these points, the
    corresponding probability distribution function

Z1 = multivariate_gaussian(pos, mu1, Sigma1)
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], c = 'red',
    marker='*', s= 100, label='Train Class-1',
    edgecolors='black')
ax.scatter(class2[:,0], class2[:,1], c = 'blue',
    label='Train Class-2', edgecolors='black', alpha
    =0.75, s=50)

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()

ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
ax.set_zlabel("z-axis")

# 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"] = (10,8)
plt.show()

#Along with decesion boundary
fig = plt.figure()
ax = fig.gca(projection='3d')


ax.scatter(class1[:,0], class1[:,1], c = 'red',
    marker='*', s= 100, label='Train Class-1',
    edgecolors='black')
ax.scatter(class2[:,0], class2[:,1], c = 'blue',
    label='Train Class-2', edgecolors='black', alpha
    =0.75, s=50)

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)
```

```
#Decesion Boundary
decesionBoundary = Z1-Z2
ax.contourf(X, Y, decesionBoundary, zdir='z', offset
    =-0.15, cmap=cm.PuBu )

ax.legend()

ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
ax.set_zlabel("z-axis")

# 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"] = (10,8)
plt.show()
```