Implementing Minimum Error Rate Classifier

Mayeesha Humaira
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204008@aust.edu
section - A1

***Abstract*—The main aim of this experiment is to design a minimum error rate classifier for a two class problem with given sample data.**

***Index Terms*—Multivariate normal distribution; contour; probability distribution function;**

I. INTRODUCTION

The main objective of Minimum Error Rate Classifier is to decrease the the risk function and increase Posterior Probability. Maximum Discriminant function corresponds to maximum posterior probability. Posterior probability can be broken down to Likelihood and prior as shown in Eq. 1.

$$P(\omega_i|x) = P(x|\omega_i) * P(\omega_i) \tag{1}$$

where $P(\omega_i|x)$ is the posterior probability, $P(x|\omega_i)$ is the likelihood and $P(\omega_i)$ is the prior. Eq. 1 can be also written like this:

$$lnP(\omega_i|x) = lnP(x|\omega_i) + lnP(\omega_i) \tag{2}$$

$$g_i(X) = P(x|\omega_i) * P(\omega_i) \tag{3}$$

Posterior probability equals to discriminate function hence discriminant is equal to Likelihood * Prior.

The decision Rule of this classifier is:
If $g_1(X) > g_2(X)$ then $\in\omega_1$
IF $g_2(X) > g_1(X)$ then $\in\omega_2$

Multivariate Gaussian distribution is needed since dimension of data sample is more than one. The equation of Gaussian multivariate normal distribution is given in Eq. 4.

$$N(\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{\frac{d}{2}}|\sigma_i|^{\frac{1}{2}}} \exp^{-0.5(x-\mu_i)^t\Sigma_i^{-1}(x-\mu_i)} \tag{4}$$

Where Σ is the co-variance matrix and μ is the mean and d is 2 since our data sample is 2 dimentional and i is the class.

II. EXPERIMENTAL DESIGN / METHODOLOGY

A dataset named test.txt was provided which contain data points of two classes. Data in the dataset are shown in Table I. The following task were performed on this dataset.

TABLE I
DATA POINTS IN THE TRAIN-PERCEPTRON.TXT DATASET.

x1	x2
1	1
1	-1
4	5
-2	2.5
0	2
2	-3

1) **Task-1:**
Design a minimum error rate classifier for a two class problem with given data (assuming they follow normal distribution):
$P(x|\omega_1) = N(\mu_1, \Sigma_1)$
where, $\mu_1 = [0, 0]$
and $\Sigma_1 = [0.25, 0.3; 0.3, 1]$;
$P(\omega_1 = 0.5)$

$P(x|\omega_2) = N(\mu_2, \Sigma_2)$
where, $\mu_2 = [2, 2]$
and $\Sigma_2 = [0.5, 0.0; 0.0, 0.5]$;
$P(\omega_2 = 0.5)$
Classify the sample points from "test.txt".
Solution:-

Putting Eq. 4 in Eq. 3 and taking ln we get:

$$g_i(x) = \frac{1}{2}(x-\mu_i)^t\Sigma_i^{-1}(x-\mu_i) - \frac{d}{2}ln(2\pi) - \frac{1}{2}ln|\Sigma_i| + lnP(\omega_i) \tag{5}$$

I calculated the value of g_x for every data points then using decision rule mentioned in introduction classified the data points into respective class.

2) **Task-2:**
Classified samples should have different colored markers according to the assigned class label.
Solution:-
Data points were plotted using matplotlib as shown in Fig 1. Here if $g_1(X) > g_2(X)$ then the red circles represent data points of class-1 and if $g_2(X) > g_1(X)$ the the blue star marks represent data points of class-2.

3) **Task-3:**
Draw a figure which should include these points, the corresponding probability distribution function along

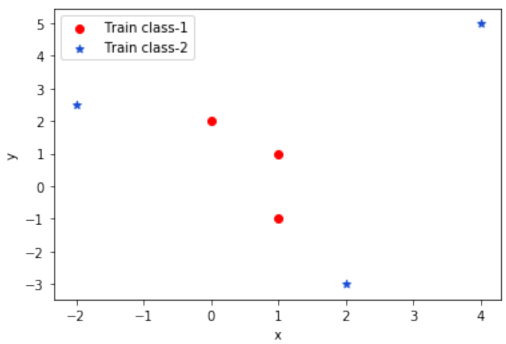

Fig. 1. Graphical representation of training data using matplotlib.

with its contour.

Solution:-

The contour plot with the classified data points are shown in 2 and the probability distribution function along with its contour and classified data points are illustrated in Fig 3

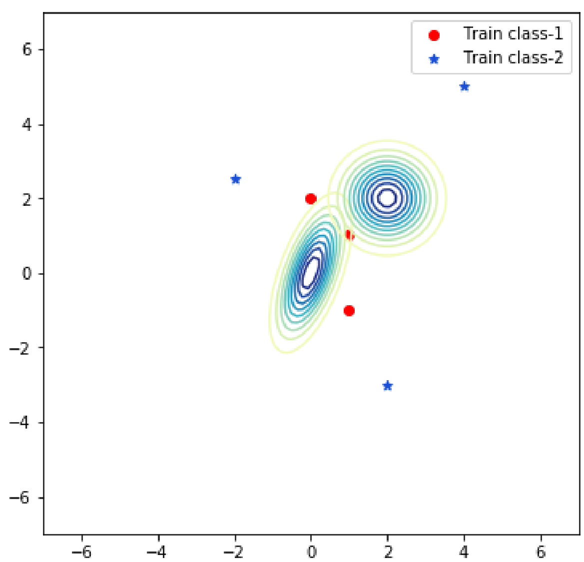

Fig. 2. The contour plot with the classified data points

4) **Task-4:**

Draw decision boundary.

Solution:-

The equation of the decision boundary was generated by subtracting the discriminant of two classes $g_1 - g_2 = 0$ where g_i is given in Eq. 4. The 3D plot of classified data points, probability distribution along with decision boundary is shown in Fig 4.

III. Conclusion

Minimum error rate classifier tries to minimize error. As a result, if parameters of multivariate Gaussian distribution mean and variance are changed the sample values will be shifted to another class.

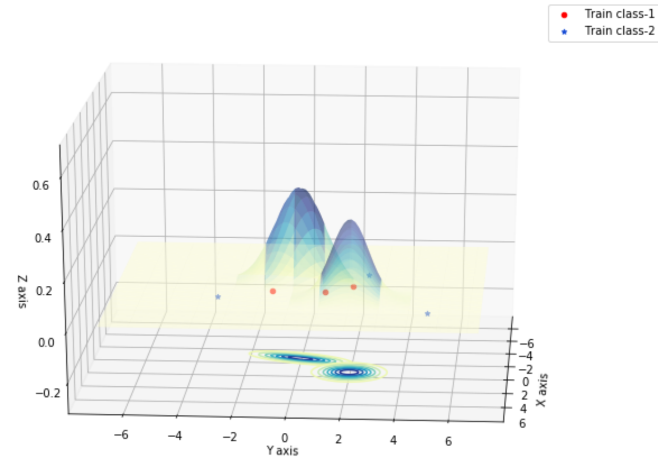

Fig. 3. 3D representation of the probability distribution function along with its contour and classified data points

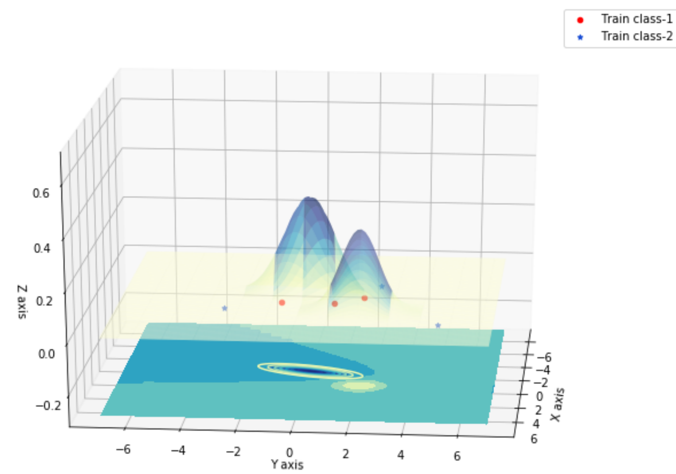

Fig. 4. 3D representation of the probability distribution function along with its contour, classified data points and decision boundary.

IV. Algorithm Implementation / Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm


#Plotting training data
xd,yd = np.loadtxt('test.txt',unpack=True, delimiter
    =' ')
print(type(xd),xd)
print(yd)

#Sigma1
sigma1 = np.array([[.25,.3], [.3,1]])
detsigma1=np.linalg.det(sigma1)
print(sigma1)
print(detsigma1)
```

```python

#Sigma=2
sigma2 = np.array([[.5,0], [0,.5]])
detsigma2=np.linalg.det(sigma2)
print(sigma2)
print(detsigma2)

#mean
mu1=np.array([0, 0])
mu2=np.array([2, 2])

print(type(mu1), mu1)
print(mu2)

#Prior
prior1=0.5
prior2=0.5

#Data before classification
data = np.zeros((len(xd),2))
print(data,  data.shape, data.ndim)

for h in range(len(xd)):
    data[h][0]=xd[h]
    data[h][1]=yd[h]
print(data, type(data))

#classifying data
cl=[]
plt.xlabel(’x1’)
plt.ylabel(’x2’)
for i in range(len(xd)):
    for j in range(1,3):
        if j==1:
            g1 = -0.5*np.dot(np.dot((data[i,:]-mu1).T,np.linalg.inv(sigma1)),(data[i,:]-mu1))-np.log(2*np.pi)-0.5*np.log(np.linalg.det(sigma1))+np.log(prior1)
        elif j==2:
            g2 = -0.5*np.dot(np.dot((data[i,:,]-mu2).T,np.linalg.inv(sigma2)),(data[i,:]-mu2))-np.log(2*np.pi)-0.5*np.log(np.linalg.det(sigma2))+np.log(prior2)

    if g1>g2:
        xc1=plt.scatter(data[i,0], data[i,1], color=’r’)
        cl.append(1)
    else:
        xc2=plt.scatter(data[i,0], data[i,1], marker=’*’, color=’#184DD5’)
        cl.append(2)
plt.legend([xc1, xc2], ["Train class-1", "Train class-2"])
plt.show()

#data with class
datacl = np.zeros((len(xd),3))
print(datacl,  datacl.shape, datacl.ndim)

for h in range(len(xd)):
    datacl[h][0]=xd[h]
    datacl[h][1]=yd[h]
    datacl[h][2]=cl[h]
print(datacl, type(datacl))

#seperating two class data
cl1 =[([ar[0],ar[1],ar[2]]) for ar in datacl if ar[2] == 1]
cl1 = np.array(cl1)
cl2 =[([ar[0],ar[1],ar[2]]) for ar in datacl if ar[2] == 2]
cl2 = np.array(cl2)
print(cl1)
print(cl2)

#contour plot with data
d=2
def multivariate_normal(x, d, mean, covariance):
    """pdf of the multivariate normal distribution."""
    x_m = x - mean
    return (1. / (np.sqrt((2 * np.pi)**d * np.linalg.det(covariance))) *
            np.exp(-(np.linalg.solve(covariance, x_m).T.dot(x_m)) / 2))

# Plot bivariate distribution
def generate_surface(mean, covariance, d):
    """Helper function to generate density surface."""
    nb_of_x = 100 # grid size
    x1s = np.linspace(-7.0, 7, num=nb_of_x)
    x2s = np.linspace(-7.0, 7, num=nb_of_x)
    x1, x2 = np.meshgrid(x1s, x2s) # Generate grid
    pdf = np.zeros((nb_of_x, nb_of_x))
    # Fill the cost matrix for each combination of weights
    for i in range(nb_of_x):
        for j in range(nb_of_x):
            pdf[i,j] = multivariate_normal(
                np.matrix([[x1[i,j]], [x2[i,j]]]),
                d, mean, covariance)
    return x1, x2, pdf  # x1, x2, pdf(x1,x2)

bivariate_mean = np.matrix([[0.], [0.]])  # Mean
bivariate_covariance = np.matrix([
    [0.25, 0.3],
    [0.3, 1.]])  # Covariance
x1, x2, p = generate_surface(
    bivariate_mean, bivariate_covariance, d)
# Plot bivariate distribution
fig, ax = plt.subplots(figsize=(6,6))
d = 2
ax.contour(x1, x2, p, 10, cmap=cm.YlGnBu)
bivariate_mean = np.matrix([[2.], [2.]])  # Mean
bivariate_covariance = np.matrix([
    [0.5, 0.],
    [0., 0.5]])  # Covariance
x1, x2, p = generate_surface(
    bivariate_mean, bivariate_covariance, d)

ax.contour(x1, x2, p, 10, cmap=cm.YlGnBu)

for i in range(len(xd)):
    if datacl[i][2]==1:
        xc1=plt.scatter(data[i,0], data[i,1], color=’r’)
    else:
        xc2=plt.scatter(data[i,0], data[i,1], marker=’*’, color=’#184DD5’)
plt.legend([xc1, xc2], ["Train class-1", "Train class-2"])
plt.show()


#3d plot
# 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)

# Mean vector and covariance matrix
mu = np.array([0., 0.])
Sigma = np.array([[ .25 , 0.3], [.3,  1.]])
mu1 = np.array([2.,2.])
```

```python
Sigma1 = np.array([[ .5 , 0.], [0.,  .5]])

# 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_gaussiandb(pos, mu, Sigma):
    n = mu.shape[0]
    print(n)
    Sigma_det = np.linalg.det(Sigma)
    Sigma_inv = np.linalg.inv(Sigma)
    N = np.sqrt((2*np.pi)**n * Sigma_det)
     # This einsum call calculates (x-mu)T.Sigma-1.(x-mu) in a vectorized
    # way across all the input variables.
    fac = np.einsum('...k,kl,...l->...', pos-mu, Sigma_inv, pos-mu)
    print(fac)
    return np.exp(-fac / 2) / N

Z = multivariate_gaussiandb(pos, mu, Sigma)
print(mu.shape[0])
Z1 = multivariate_gaussiandb(pos, mu1, Sigma1)
# Create a surface plot and projected filled contour plot under it.
db=Z-Z1

# Create grid and multivariate normal
def multivariate_normal(x, d, mean, covariance):
    """pdf of the multivariate normal distribution."""
    x_m = x - mean
    return (1. / (np.sqrt((2 * np.pi)**d * np.linalg.det(covariance))) *
            np.exp(-(np.linalg.solve(covariance, x_m).T.dot(x_m)) / 2))

# Plot bivariate distribution
def generate_surface(mean, covariance, d):
    """Helper function to generate density surface."""
    nb_of_x = 100 # grid size
    x1s = np.linspace(-7.0, 7, num=nb_of_x)
    x2s = np.linspace(-7.0, 7, num=nb_of_x)
    x1, x2 = np.meshgrid(x1s, x2s) # Generate grid
    pdf = np.zeros((nb_of_x, nb_of_x))
    # Fill the cost matrix for each combination of weights
    for i in range(nb_of_x):
        for j in range(nb_of_x):
            pdf[i,j] = multivariate_normal(
                np.matrix([[x1[i,j]], [x2[i,j]]]),
                d, mean, covariance)
    return x1, x2, pdf  # x1, x2, pdf(x1,x2)
# Make a 3D plot
fig = plt.figure(figsize=[12,8])

ax = fig.add_subplot(111, projection='3d')
d = 2

bivariate_mean = np.matrix([[0.], [0.]])  # Mean
bivariate_covariance = np.matrix([
    [0.25, 0.3],
    [0.3, 1.]])  # Covariance
x1, x2, p = generate_surface(
    bivariate_mean, bivariate_covariance, d)
#ax.contourf(x1, x2, p, 100, alpha=0.4,  cmap=cm.YlGnBu)
surf = ax.plot_surface(x1, x2, p, rstride=8, cstride=1,alpha=0.4, cmap=cm.YlGnBu)
cset = ax.contour(x1, x2, p,offset=-0.3, cmap=cm.YlGnBu)

bivariate_mean = np.matrix([[2.], [2.]])  # Mean
bivariate_covariance = np.matrix([
    [0.5, 0.],
    [0., 0.5]])  # Covariance
x1, x2, p = generate_surface(
    bivariate_mean, bivariate_covariance, d)

surf = ax.plot_surface(x1, x2, p, rstride=8, cstride=1, alpha=0.3, cmap=cm.YlGnBu)
cset = ax.contourf(X, Y, db, zdir='z', offset=-0.3, cmap=cm.YlGnBu)

for i in range(len(xd)):
    if datacl[i][2]==1:
        xc1=ax.scatter(data[i,0], data[i,1], color='r')
    else:
        xc2=ax.scatter(data[i,0], data[i,1], marker='*', color='#184DD5')
plt.legend([xc1, xc2], ["Train class-1", "Train class-2"])

ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlim(-0.3, 0.7)
ax.set_zlabel('Z axis')
ax.view_init(elev=20, azim=5)
plt.show()
```