Implementing Minimum Error Rate Classifier

Md. Ashraful Haque
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204101@aust.edu

***Abstract*—Minimum error rate classifier is a probabilistic Generative method of classification which uses the Bayesian probability theory at it's core.**

I. Introduction

The primary objective of the algorithm is to 1st find the **Gaussian distribution** of all of the classes and then to classify the data according to it.

The basic components of the algorithm are the **test set** , **priors**, **Mean of the classes** and **Variance of the classes**. The test data will be used to generate the multiplication of the **likelihood** and **prior** with the help of the mean and variance of each class.

II. Experimental Design / Methodology

step 1 Collecting the test data. In our case since it is a generative model there was 6 test data point with out labels.

step 2

Send the data on by one to the function which calculates the the multiplication of the prior and likelihood of a certain function. The give the decision, to which class does the point belong. The decision rule is,

$$\text{if } P(\omega1|x) > P(\omega2|x) \text{ then } x \in \omega1$$

$$\text{else } x \in \omega2$$

here $P(\omega1|x)$ is the posterior probability.

$$P(\omega i|x) = P(x|\omega i)P(\omega i)$$

So, the decision rule becomes,

$$\text{if } P(x|\omega1)P(\omega1) > P(x|\omega2)P(\omega2) \text{ then } x \in \omega1$$

$$\text{else } x \in \omega2$$

here $P(wi)$ is the prior of the class i.

Now, the probability distribution function that is used in the **likelihood** $P(x|\omega i)$ estimation is the **multivariate Gaussian distribution** equation which is,

$$P(x|\omega i) = \frac{1}{2\pi^{\frac{d}{2}}|\sum|_i^{\frac{1}{2}}} e^{(-.5(x-\mu_i)^T(\sum)_i^{-1}(x-\mu_i))}$$

here,
d = number of dimentions
$\sum$ = Covrience Matrix of class i
μ = Mean of class i

step 3

For decision boundary,

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

$$\Rightarrow P(x|\omega1) = P(x|\omega2)$$

$$\Rightarrow P(x|\omega1)P(\omega1) = P(x|\omega2)P(\omega2)$$

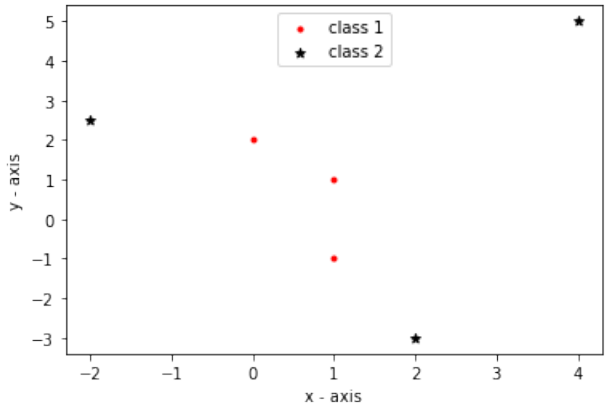
$$\Rightarrow P(x|\omega1)P(\omega1) - P(x|\omega2)P(\omega2) = 0$$

III. Result Analysis

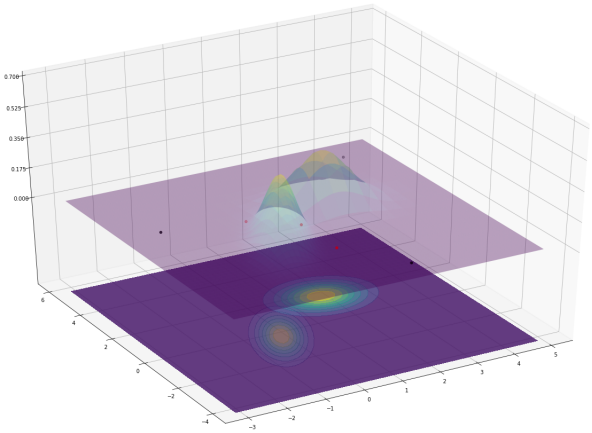
Task 1 Classification

x-value	y-value	Class
1	1	1
1	-1	1
4	5	2
-2	2.5	2
0	2	1
2	-3	2

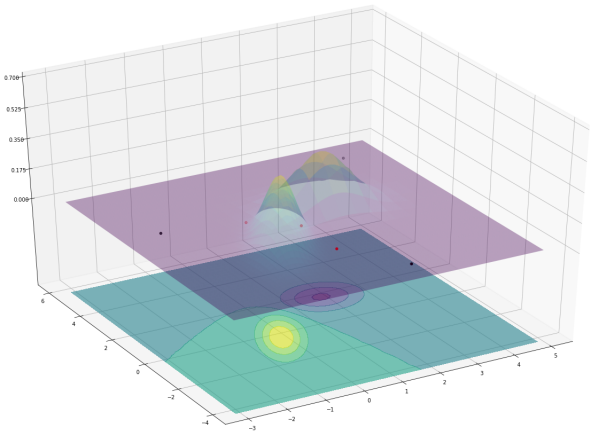
Task 2 plot



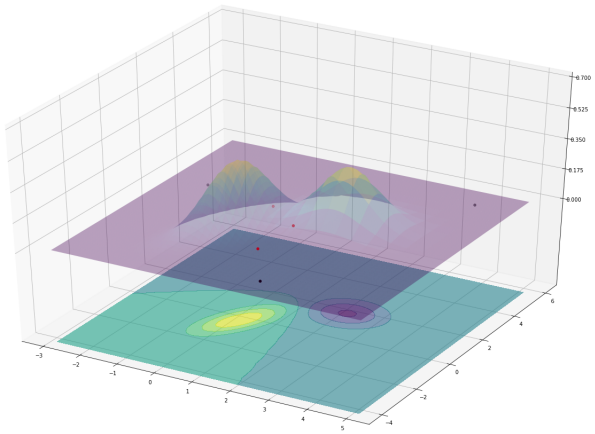
Task 3 contour plot



Task 4 with decision boundary
from 300 degree rotation



From 240 degree rotation



IV. Conclusion

So, by these experiments we can come to the conclusion that by variations of the different kind, we can optimize the performance of the classifier.

V. Algorithm Implementation / Code

Task 1:

```
def normalDistribution(x ,mu , sigma ):
    size = len(x)
    determinantofSigma = np.linalg.det(sigma)

    norm_const = 1.0/ ( math.pow((2*math.pi),float(
    size)/2) * math.pow(determinantofSigma,1.0/2) )
    x_mu = np.matrix(x - mu)
    inv = np.matrix(sigma).I


    result = math.pow(math.e, -0.5 * (x_mu * inv *
    x_mu.T))

    value = norm_const * result

    return value


classified = {}

for i in testData:
    print(i , normalDistribution(i ,miu1 , sigma1  )
     , normalDistribution(i ,miu2 , sigma2  ))
    if(normalDistribution(i ,miu1 , sigma1  ) *
    prior1 >= normalDistribution(i ,miu2 , sigma2  )
     * prior2):
        print("class 1")
        classified[tuple(i)] = 1

    else:
        print("class 2")
        classified[tuple(i)] = 2

classified
```

Task 2:

```
trainData1x = []
trainData1y = []
trainData2x = []
trainData2y = []

for coor , classinfo in classified.items():
    if(classinfo == 1):
        trainData1x.append(coor[0]);
        trainData1y.append(coor[1]);
    else:
        trainData2x.append(coor[0]);
        trainData2y.append(coor[1]);

print(trainData1x , trainData1y)
print(trainData2x , trainData2y)



plt.scatter(trainData1x, trainData1y, color = 'r',
    marker = ".", label = 'class 1')

plt.scatter(trainData2x, trainData2y, color = 'k',
    marker = "*", label = 'class 2')

plt.xlabel('x - axis')
plt.ylabel('y - axis')

plt.legend(loc = 'upper center')

plt.show()
```

Task 3 and 4:

```
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 = 60
X = np.linspace(min(trainData1x + trainData2x)-1,
    max(trainData1x + trainData2x)+1, N)
Y = np.linspace(min(trainData1y + trainData2y)-1,
    max(trainData1y + trainData2y)+1, 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



# The distribution on the variables X, Y packed into
    pos.

Z = finDistribution(pos, miu1, sigma1)



Z2 = finDistribution(pos, miu2, sigma2)


db = Z - Z2


#Z = Z + Z2




# Create a surface plot and projected filled contour
    plot under it.
fig = plt.figure(figsize = (24,16))
ax = fig.gca(projection='3d')



plt1 = ax.plot_surface(X, Y, Z, rstride=1, cstride
    =1, linewidth=1,alpha=.2, antialiased=True,cmap=
    'viridis')

plt2 = ax.plot_surface(X, Y, Z2, rstride=3, cstride
    =3, linewidth=1,alpha=.2, antialiased=True,cmap=
    'viridis')

#ax.hold(True)

#ax.scatter(points2[0], point2[1], point2[2], color
    ='green')

ax.scatter(trainData1x, trainData1y, color = 'r',
    marker = "o", label = 'class 1')

ax.scatter(trainData2x, trainData2y, color = 'k',
    marker = "o", label = 'class 2')
```

```
#cset1 = ax.contourf(X, Y, Z, zdir='z', offset=-.55,
     alpha=.6, cmap='viridis')
#cset2 = ax.contourf(X, Y, Z2, zdir='z', offset
    =-.55, alpha=.6, cmap='viridis')


decisionBoundary = ax.contourf(X, Y, db, zdir='z',
    offset=-0.55, alpha=.6, cmap='viridis')
#cset = ax.contourf(X, Y, Z2, zdir='z', offset=-.2,
    cmap='binary')

#decision boundary
#ax.plot(trainData1x , trainData1y, "--r", label = '
    Decision Boundary')



# Adjust the limits, ticks and view angle
ax.set_zlim(-.5,.5)
ax.set_zticks(np.linspace(0,0.7,5))
ax.view_init(35, 300)

#ax.view_init(35, 240)

plt.show()
```