Implementing Minimum Error Rate Classifier

Md. Zahid Fesabelilla
Computer Science and Engineering. Ahsanullah University of Science and Technology
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204082@aust.edu

***Abstract*—This document is about implementation of the minimum error rate classifier.Draw a figure which should include sample points, and also find decision boundary.**

***Index Terms*—Gaussian normal distribution,probability distribution function ,Posterior,Likelihood,Prior,Sigma,Mean,Coveriance matrix etc.**

I. Introduction

The minimum error rate classifier is a statistical classifier. The minimum error rate classifier or Bayes error rate is the lowest possible error rate for any classifier of a random outcome and is analogous to the irreducible error.[wikipedia]

Given a sample file "test.txt" and our target is to classify the sample point and draw a contour plot which should include these points,and identify the decision boundary.

Minimum Error Rate classifier :

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

II. Experimental Design / Methodology

Firstly,we need to classify all sample points into two classes using a Multivariate normal density and Minimum Error Rate classifier.
We know,
Posterior = Likelihood * Prior.

For class 1 Minimum Error Rate classifier is :
$g_1(x) = -\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) - \frac{d}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_1| + \log P(\omega_1)$

For class 2 Minimum Error Rate classifier is :
$g_2(x) = -\frac{1}{2}(x-\mu_1)^T\Sigma_2^{-1}(x-\mu_2) - \frac{d}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_2| + \log P(\omega_2)$

Using these two classifier we classify 2 classes. If g1(x) greater than g2(x) then the point will go class 1 otherwise class2.

Secondly, Plot all sample points from both classes, then identify the same class using the same color and marker.

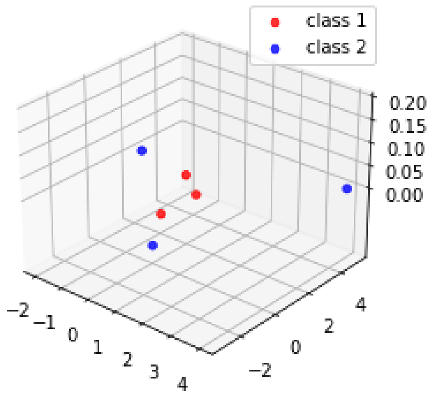

Fig. 1. All sample points from both classes

Thirdly,draw a contour plot using multivariate normal density .

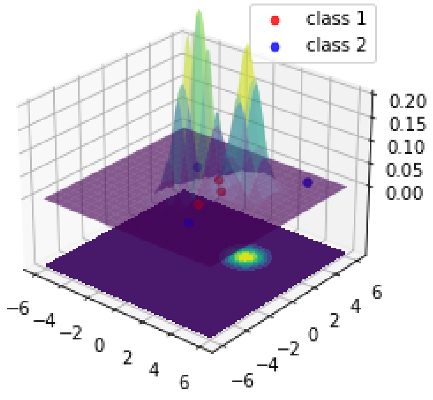

Fig. 2. Draw contour plot

Fourthly, we need to draw a decision boundary between the two classes. For this decision boundary, we need to find the difference between two posterior, The equation is :

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

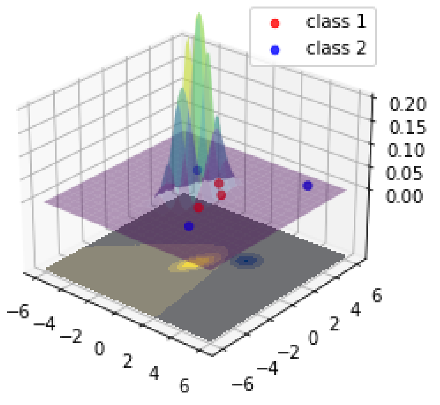

Fig. 3. Draw decision boundary

III. Result Analysis

The decision regions for two normal distributions. Even with such a low number of categories, the shapes of the boundary regions can be rather complex. Maximum points are in the hilltop area.
Here, successfully classified two classes with minimum error rate, draw a contour plot using test data,and successfully draw a decision boundary which is separate these two class.

IV. Conclusion

After using probability density function of the multivariate Gaussian distribution minimize the error and perfectly classify test data into two classes and find them decision boundary.

V. Algorithm Implementation / Code

```
# coding: utf-8

# In[137]:


import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from IPython.display import display
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D


# In[138]:


import numpy as np
import pandas as pd
import math


# In[139]:


train = pd.read_csv('TMERC.txt',header = None)
train


# In[140]:


train = np.array(train)
train


# In[141]:


class1 = []
class2 = []
#prior w1, w2
w1 = .5
w2 = .5
d = train.ndim # (d = 2)


# In[142]:


sigma1 = np.array([[0.25,0.3],[0.3,1]])
sigma2 = np.array([[0.5,0],[0,0.5]])


# In[143]:



mean1 = np.array([0,0])
mean2 = np.array([2,2])


# In[144]:


# for class 1
x1 = []
y1 = []

# for class 2
x2 = []
y2 = []


# In[150]:


n = (train.size)/2 # 12/2 = 6

temp = []

for i in range(int(n)):
    g1 = -0.5*(np.dot(np.dot(train[i] - mean1,np.
    linalg.inv(sigma1)), (train[i] - mean1).T)) - (d
    /2)*np.log(2*np.pi) - (0.5 * np.log(np.linalg.
    det(sigma1))) + w1
    g2 = -0.5*(np.dot(np.dot(train[i] - mean2,np.
    linalg.inv(sigma2)), (train[i] - mean2).T)) - (d
    /2)*np.log(2*np.pi) - (0.5 * np.log(np.linalg.
    det(sigma2))) + w2
    #print(g1,"    ",g2)
    a = train[i][0]
    b = train[i][1]
    if g1 > g2:
        x1.append(a)
        y1.append(b)
        temp.append(a)
        temp.append(b)
        temp.append(1)
        class1.append(temp)
    else:
        x2.append(a)
        y2.append(b)
        temp.append(a)
        temp.append(b)
        temp.append(2)
        class2.append(temp)
    temp = []


# In[151]:


print(x1,y1)
print(x2,y2)


# In[152]:



print("Class 1 : ", class1)
print("Class 2 : ",class2)


# In[158]:


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


# 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, -50)


ax.scatter(x1,y1,0, color = 'r',marker = "o", alpha
    = 0.8,  label = 'class 1')
ax.scatter(x2,y2,0, color = 'b',marker = "o", alpha
    = 0.8,  label = 'class 2')

plt.legend()
plt.show()


# In[153]:


'''
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 60)
y = np.linspace(-6, 6, 60)

X, Y = np.meshgrid(x1, y1)
Z = f(X, Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('3D contour')
plt.show()
'''


# In[162]:


###########
## Reference : https://scipython.com/blog/
    visualizing-the-bivariate-gaussian-distribution
    /?fbclid=IwAR1CrhQe7bpGl-
    LICNjpTkArYxPmsNLDal7kZ67RTq_DAkvWcuSYUtg5DFQ
###########

N = 60
X = np.linspace(-6, 6, 60) ## range -6 to +6 60
    times
Y = np.linspace(-6, 6, 60)
X, Y = np.meshgrid(X, Y) #convert it to numpyarray
#print(X)


# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y

def multivariateGaussian(pos, mu, Sigma):
    n = mu.shape[0] # dimension
    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)

    return np.exp(-fac / 2) / N

# The distribution on the variables X, Y packed into
     pos.
Z1 = multivariateGaussian(pos, mean1, sigma1)
Z2 = multivariateGaussian(pos, mean2, sigma2)


# Create a surface plot and projected filled contour
     plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z1, rstride=3, cstride=3,
    linewidth=1, alpha = 0.4 ,antialiased=True,cmap=
    cm.viridis)
ax.plot_surface(X, Y, Z2, rstride=3, cstride=3,
    linewidth=1, alpha = 0.4 ,antialiased=True,cmap=
    cm.viridis)

cset1 = ax.contourf(X, Y, Z1, zdir='z', offset
    =-0.15, cmap=cm.viridis)
cset2 = ax.contourf(X, Y, Z2, zdir='z', offset
    =-0.15, cmap=cm.viridis)

#decision boundary
db = ax.contourf(X, Y, (Z1 - Z2), zdir='z', offset
    =-0.15, cmap=cm.cividis)


# 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, -50)


ax.scatter(x1,y1,0, color = 'r',marker = "o", alpha
    = 0.8,  label = 'class 1')
ax.scatter(x2,y2,0, color = 'b',marker = "o", alpha
    = 0.8,  label = 'class 2')

plt.legend()
plt.show()
```

```
# In[ ]:
```