```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import plotly.graph_objs as go
import plotly.express as px


main_directory  = '/content/gdrive/MyDrive/Pattern Lab/Assignment2/test-Minimum-Error-Rate

df1 = pd.read_csv(main_directory, sep=" " , header = None)
print(df1)
```

```
   0    1
0  1  1.0
1  1 -1.0
2  4  5.0
3 -2  2.5
4  0  2.0
5  2 -3.0
```

Part1

```
testX = df1.iloc[:, 0:2]
testX = np.array(testX)


# To check if data is successfully retrieved
print(testX)
```

```
[[ 1.   1. ]
 [ 1.  -1. ]
 [ 4.   5. ]
 [-2.   2.5]
 [ 0.   2. ]
 [ 2.  -3. ]]
```

Declaring Normal Distribution Values

Following the question description,

For Class ω_1:

$$\mu_1 = \begin{bmatrix} 0. & 0. \end{bmatrix} \quad \Sigma_1 = \begin{bmatrix} 0.25 & 0.30 \\ 0.30 & 1.00 \end{bmatrix} \quad P(\omega_1) = 0.5$$

For Class ω_2:

$$\mu_2 = \begin{bmatrix} 2. & 2. \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 0.50 & 0.00 \\ 0.00 & 0.50 \end{bmatrix} \quad P(\omega_2) = 0.5$$

```
priorOmega1 = 0.5
meanClass1 = np.array([0., 0.])
varainceClass1 = np.array([[0.25, 0.30],
                           [0.30, 1.00]])
priorOmega2 = 0.5
meanClass2 = np.array([2., 2.])
varainceClass2 = np.array([[0.50, 0.00],
                           [0.00, 0.50]])
```

Implementing Formula

```
def normal(x, mean, variance):
  out = np.zeros((x.shape[0],))
  for i in range(x.shape[0]):
    _x = x[i, :]
    out[i,] = np.exp(- 0.5 * np.dot(_x - mean, np.dot(np.linalg.inv(variance), (_x - mean)
  return out
```

Generating Values

Generating value of both classes and for all provided points using [Normal Distribution Formula]

```
norm1 = normal(testX, meanClass1, varainceClass1) * priorOmega1
norm2 = normal(testX, meanClass2, varainceClass2) * priorOmega2

print(norm1)
print(norm2)
```

```
[2.61089678e-02 6.14024070e-04 2.44317877e-15 4.76628550e-13
 8.74540876e-03 8.52753181e-15]
[2.15502043e-02 7.22928818e-06 3.59925065e-07 1.39558093e-08
 2.91650301e-03 2.21145603e-12]
```

Part2

Generating Points for Plot

```
dataPoint = pd.DataFrame(columns = ['X1', 'X2', 'Probability Density Function', 'Size', 'S
symbol = []

norm1 = normal(testX, meanClass1, varainceClass1)
norm2 = normal(testX, meanClass2, varainceClass2)

for i in range(testX.shape[0]):
  print(testX[i, :], 'are in', end = ' ')
  if norm1[i] > norm2[i]:
    dataPoint.loc[len(dataPoint)] = [testX[i, 0], testX[i, 1], norm1[i], 0.5, 'Class 1', '
    print('Class 1')
    symbol.append('x')
  else:
    dataPoint.loc[len(dataPoint)] = [testX[i, 0], testX[i, 1], norm2[i], 0.5, 'Class 2', '
    print('Class 2')
    symbol.append('square')
```

```
[1. 1.] are in Class 1
[ 1. -1.] are in Class 1
[4. 5.] are in Class 2
[-2.   2.5] are in Class 2
[0. 2.] are in Class 1
[ 2. -3.] are in Class 2
```

```
pointsX1 = np.linspace(-6, 6, 500)
pointsX2 = np.linspace(-6, 6, 500)
pointsPDF = np.zeros((500, 500))

for i in range(len(pointsX1)):
  for j in range(len(pointsX2)):
    nm1 = normal(np.array([[pointsX1[i], pointsX2[j]]]), meanClass1, varainceClass1)
    nm2 = normal(np.array([[pointsX1[i], pointsX2[j]]]), meanClass2, varainceClass2)

    pointsPDF[i][j] = max(nm1[0], nm2[0])
```

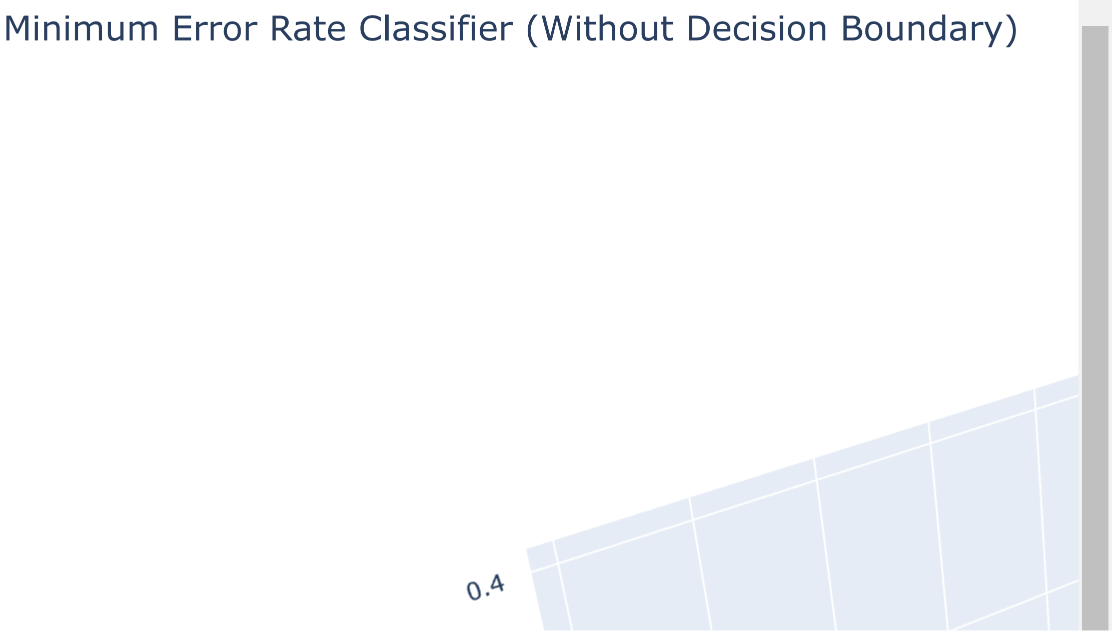
```
dataPoint
```

	X1	X2	Probability Density Function	Size	Symbol	Color
0	1.0	1.0	5.221794e-02	0.5	Class 1	#00bfff
1	1.0	-1.0	1.228048e-03	0.5	Class 1	#00bfff
2	4.0	5.0	7.198501e-07	0.5	Class 2	#8fbc8f
3	-2.0	2.5	2.791162e-08	0.5	Class 2	#8fbc8f
4	0.0	2.0	1.749082e-02	0.5	Class 1	#00bfff
5	2.0	-3.0	4.422912e-12	0.5	Class 2	#8fbc8f

Part3

Plotting Data

```
layout = go.Layout(title = 'Minimum Error Rate Classifier (Without Decision Boundary)', wi

data = [go.Surface(name = 'Gaussian Distribution', z = pointsPDF, x = pointsX1, y = points

fig = go.Figure(data = data, layout = layout)

fig.update_traces(contours_z = dict(show=True, usecolormap=True, highlightcolor="limegreen

fig.add_scatter3d(name = 'Test Point Location', x = dataPoint['X1'].to_list(), y = dataPoi
                  showlegend = True, mode = 'markers', marker = dict(size = 10, color = da

fig.show()
```



Part4

Find Decision Boundary Points

Drawing the decision boundary by utilizing the equation from here

$$log(p(x|\omega_1)) - log(p(x|\omega_2)) = log(P(\omega_1)) - log(P(\omega_2))$$

However, for the display to be clear, a minor error of ± 0.05 was considered. The change of this error value will provide a slight change in the decision boundary.

```
dbZ = np.zeros((500, 500))
for i in range(len(pointsX1)):
  for j in range(len(pointsX2)):
    nm1 = normal(np.array([[pointsX1[i], pointsX2[j]]]), meanClass1, varainceClass1)
    nm2 = normal(np.array([[pointsX1[i], pointsX2[j]]]), meanClass2, varainceClass2)

    # Try different values like 1.0, 0.5, 0.05 etc and see the decision boundaries drawn f
    if abs((np.log10(nm1[0]) - np.log10(nm2[0])) - np.log10(priorOmega1) + np.log10(priorO
      dbZ[i][j] = 0.5
    else:
      dbZ[i][j] = pointsPDF[i][j]
```

Draw Decision Boundary

```
layout = go.Layout(title = 'Minimum Error Rate Classifier', showlegend = True, width = 150

data = [go.Surface(name = 'Gaussian Distribution', z = pointsPDF, x = pointsX1, y = points

fig = go.Figure(data = data, layout = layout)
```

```
fig.update_traces(contours_z = dict(show=True, usecolormap=True, highlightcolor="limegreen

fig.add_scatter3d(name = 'Points', x = dataPoint['X1'].to_list(), y = dataPoint['X2'].to_l
                  z = dataPoint['Probability Density Function'].to_list(), showlegend = Tr
                  marker = dict(size = 10, color = dataPoint['Color'], symbol = symbol))
fig.add_surface(name = 'Decision Boundary', z = dbZ,  x = pointsX1, y = pointsX2,
                colorscale = 'greens', showscale = False, opacity = 0.9)

fig.show()
```

✓ 9s completed at 12:55 PM