7th Homework

Theofanis Nitsos - p3352325

Exercise 1

3, 4

Exercise 2

2,3

Exercise 3

1,3

Exercise 4

1,3

Exercise 5

2

Exercise 6

3

Exercise 7

2

Exercise 8

1

Exercise 9

2

Exercise 10

4

Exercise 11

4

Exercise 12

1

Exercise 13

1,2 (not sure)

Exercise 14

4

Exercise 15

4

Exercise 16

4

Exercise 17

2

Exercise 18

2,3

Exercise 19

1,4

Exercise 20

2

Exercise 21

2,4

Exercise 22

2,4

Exercise 23

4

Exercise 24

3

Exercise 25

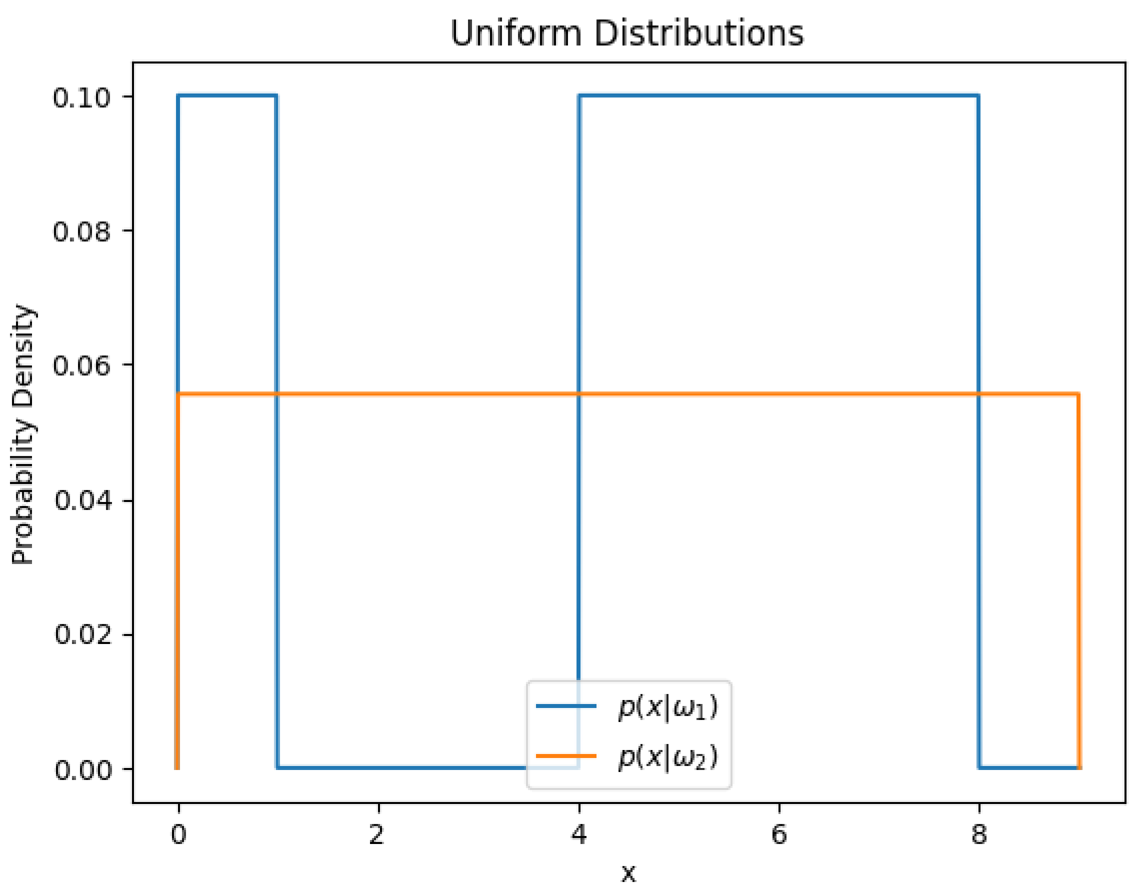
2,3

Exercise 26

(I)

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the probability density functions
def p_x_given_w1(x):
    if (0 < x < 1) or (4 < x < 8):
        return 1/10
    else:
        return 0

def p_x_given_w2(x):
    if 0 < x < 9:
        return 1/18
    else:
```

```
        return 0

# Generate x values for the plot
x_values = np.linspace(0, 9, 1000)

# Calculate the corresponding y values for each distribution
y_values_w1 = [p_x_given_w1(x) for x in x_values]
y_values_w2 = [p_x_given_w2(x) for x in x_values]

# Plot the distributions
plt.plot(x_values, y_values_w1, label=r'$p(x|\omega_1)$')
plt.plot(x_values, y_values_w2, label=r'$p(x|\omega_2)$')

# Add labels and legend
plt.xlabel('x')
plt.ylabel('Probability Density')
plt.title('Uniform Distributions')
plt.legend()

# Show the plot
plt.show()
```



- for 0<x<1 and 4<x<8 the points are assigned to the ω_1
- for 1<x<4 and 8<x<9 the points are assigned to the ω_2
- the point x=3.5 is classified to ω_2

(II)

(i) For the x = 5 to be assigned to ω_1 the following must stand

$P(\omega_1)p(x=5|\omega_1) < P(\omega_2)p(x=5|\omega_2) \Rightarrow P(\omega_1)p(x=5|\omega_1) < (1-P(\omega_1))p(x=5|$

$\Rightarrow P(\omega_1) < \frac{1/9}{1/9+1/5} \approx 0.36$

or $\Rightarrow 1-P(\omega_2) < 0.36 \Rightarrow P(\omega_2) > 0,64$

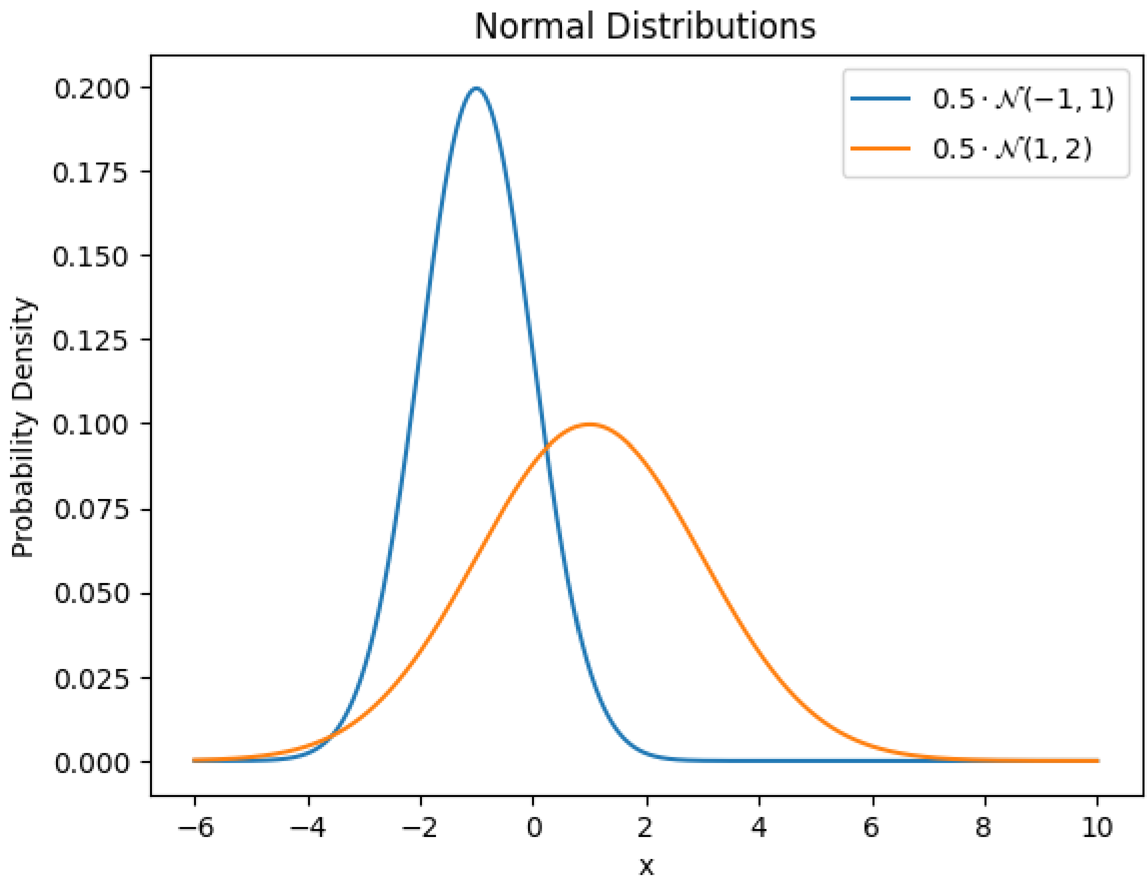
(ii) No since p(x=3|\omega_1) = 0 and for the classifier to classify x=3 to ω_1 the following should stand

$P(\omega_1)p(x=3|\omega_1) > P(\omega_2)p(x=3|\omega_2) \Rightarrow 0 > P(\omega_2)p(x=3|\omega_2)$ which can't be true.

Exercise 27

In [7]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Define the parameters for the normal distributions
mu1, sigma1 = -1, 1
mu2, sigma2 = 1, 2  # Adjusted the standard deviation to 2 for better visibi

# Generate x values for the plot
x_values = np.linspace(-6, 10, 1000)

# Calculate the corresponding y values for each distribution
pdf1 = 0.5 * norm.pdf(x_values, mu1, sigma1)
pdf2 = 0.5 * norm.pdf(x_values, mu2, sigma2)

# Plot the normal distributions
plt.plot(x_values, pdf1, label=r'$0.5 \cdot \mathcal{N}(-1,1)$')
plt.plot(x_values, pdf2, label=r'$0.5 \cdot \mathcal{N}(1,2)$')

# Add labels and legend
plt.xlabel('x')
plt.ylabel('Probability Density')
plt.title('Normal Distributions')
plt.legend()

# Show the plot
plt.show()
```



The intersection points are calculated through the equation

$P(\omega_1)p(x|\omega_1) = P(\omega_2)p(x|\omega_2)$. Since $P(\omega_1) = P(\omega_2) = 0.5$
$p(x|\omega_1) = p(x|\omega_2) \Rightarrow \frac{1}{sqrt2pi}e^{-\frac{1}{2}(\frac{x+1}{1})^2} = \frac{1}{2sqrt2pi}e^{-\frac{1}{2}(\frac{x-1}{2})^2}$
The solutions of the above equation are $x = -3.6$ and $x = 0.24$

Thus:

- $R_1 \ is \ (-\infty, -3.6) \ and \ (0.24, +\infty)$
- $R_2 \ is \ (-3.6, 0.24)$

Exercise 28

(a)

We can calculate the a-posteriori probabilities to classify these points.

$P(\omega_j|x_i) = \frac{p(x_i|\omega_j)P(\omega_j)}{\sum_{k=1}^{m} p(x_i|\omega_k)P(\omega_k)}$ since the denominator is the same it suffices to calculate the below:
$p(x_i|\omega_j)P(\omega_j)$ considering that we have 2 equiprobable classes $P(\omega_1) = P(\omega_2)$, thus we can simply calculate $p(x_i|\omega_j)$

- For x_1 $p(x_1|\omega_1) = 2.67e - 05$, $p(x_2|\omega_2) = 0.011$ belongs to ω_2
- For x_2 $p(x_2|\omega_1) = 0.01$, $p(x_2|\omega_2) = 2.67e - 05$ belongs to ω_1
- For x_3 $p(x_3|\omega_1) = 0.00054$, $p(x_3|\omega_2) = 0.00054$ the same probability for both ω_1 and ω_2

(b) $P(\omega_1)p(x|\omega_1) = P(\omega_2)p(x|\omega_2)$ since ω_1 , ω_2 are equiprobable
$\Rightarrow p(x|\omega_1) = p(x|\omega_2) \Rightarrow \frac{1}{2\pi|\Sigma_1|^{1/2}} exp\left(-\frac{(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)}{2}\right) = \frac{1}{2\pi|\Sigma_2|^{1/2}} exp\left(-\frac{(x-\mu_2)^T\Sigma_2^{-1}(}{2}\right.$
$(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) = (x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) \Rightarrow$
$\Rightarrow \begin{bmatrix} x_1 - 6 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 - 6 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 - 6 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 - 6 \end{bmatrix}$
$\Rightarrow x_1^2 - 12x_1 + x_2^2 + 36 = x_1^2 + x_2^2 - 12x_2 + 36$
$\Rightarrow x_1 = x_2$

Exercise 29

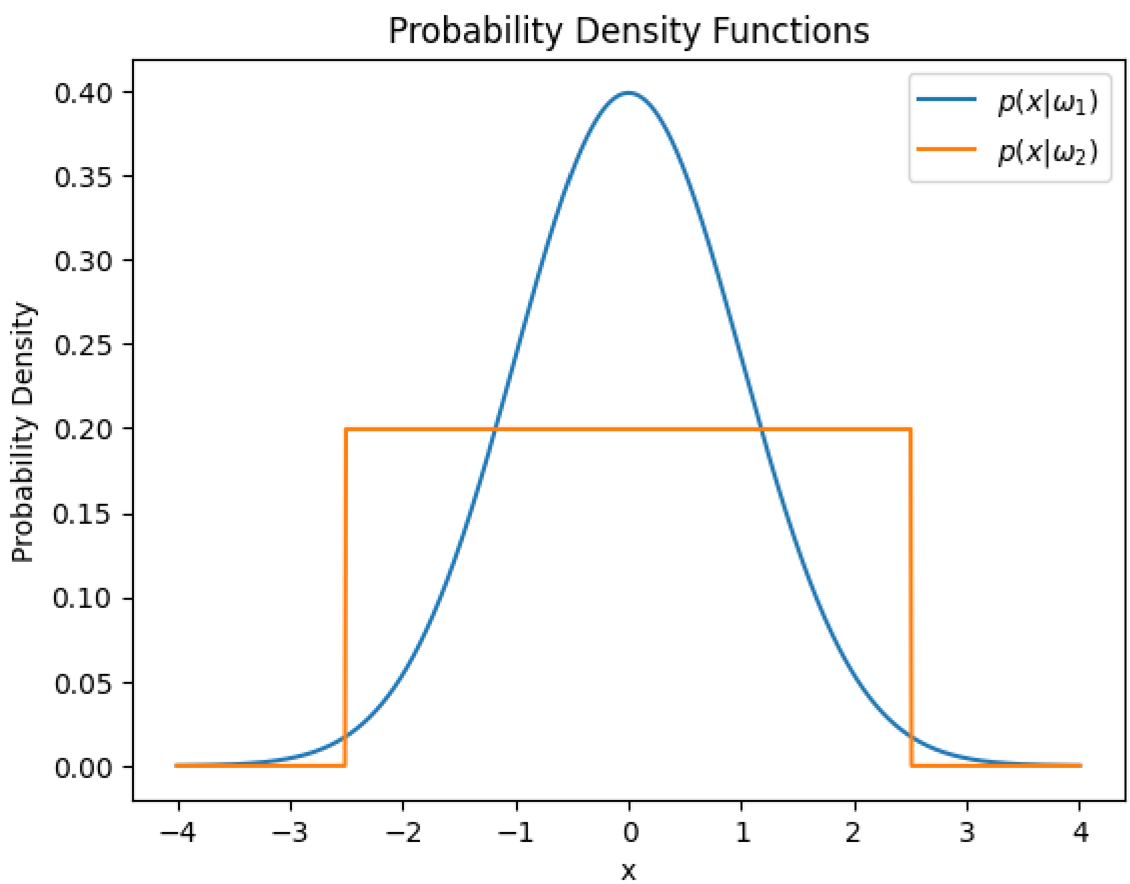
(a) Similar to the above exercise :
$P(\omega_1)p(x|\omega_1) = P(\omega_2)p(x|\omega_2)$ since ω_1 , ω_2 are equiprobable
$\Rightarrow p(x|\omega_1) = p(x|\omega_2) \Rightarrow \frac{1}{2\pi|\Sigma|^{1/2}} exp\left(-\frac{(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)}{2}\right) = \frac{1}{2\pi|\Sigma|^{1/2}} exp\left(-\frac{(x-\mu_2)^T\Sigma^{-1}(x-}{2}\right.$
$(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) = (x-\mu_2)^T\Sigma^{-1}(x-\mu_2) \Rightarrow$ since Σ are diagonal this expression can be written as
$(x-\mu_1)^T(x-\mu_1) = (x-\mu_2)^T(x-\mu_2)$
$\Rightarrow ||x-\mu_1||^2 = ||x-\mu_2||^2$ which is the equation of the perpendicular bisector of the line segment between μ_1 and μ_2

(b) If Σ is not diagonal then the σ_{ij} terms would remain in the equation and be multiplied by the respective $\mu_i s$ and $x_i s$ creating a curve.

Exercise 30

In [58]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Define the range of x values
x_values = np.linspace(-4, 4, 1000)

# Define the probability density functions
pdf_w1 = 1/np.sqrt(2*np.pi) * np.exp(-x_values**2 / 2)

def p_x_given_w2(x):
    if (-np.sqrt(2*np.pi) < x < np.sqrt(2*np.pi)) :
        return 1/(2*np.sqrt(2*np.pi))
    else:
        return 0
```

```
y_values_w2 = [p_x_given_w2(x) for x in x_values]

# Plot the probability density functions
plt.plot(x_values, pdf_w1, label=r'$p(x|\omega_1)$')
plt.plot(x_values, y_values_w2, label=r'$p(x|\omega_2)$')

# Add labels and legend
plt.xlabel('x')
plt.ylabel('Probability Density')
plt.title('Probability Density Functions')
plt.legend()

# Show the plot
plt.show()
```



The 2 plots intersect at -2π and 2π and at $p(x|\omega_1) = p(x|\omega_2)$ solving the equation $\frac{1}{\sqrt{2\pi}} exp(-\frac{x^2}{2}) = \frac{1}{2\sqrt{2\pi}} \Rightarrow x = -1.177\ and\ x = 1.177$

Thus:

$R_1 = (-\infty, -\sqrt{2\pi})\ and\ (-1.177, 1.177)\ and\ (\sqrt{2\pi}, +\infty)$

$R_2 = (-\sqrt{2\pi}, -1.177)\ and\ (1.177, \sqrt{2\pi})$

(b) Since the errors that stem from ω_2 are twice more serious we will try to reduce these errors by accepting more ω_1 errors. As such we can modify the equation as:

$$\Lambda = \begin{bmatrix} 0 & 0.5\lambda \\ \lambda & 0 \end{bmatrix}$$

$\lambda_{12} P(\omega_1) p(x|\omega_1) = \lambda_{21} P(\omega_2) p(x|\omega_2) \Rightarrow$
$P(\omega_1) p(x|\omega_1) = 2P(\omega_2) p(x|\omega_2) \Rightarrow x = 0$
Thus:
$R_1 = (-\infty, -\sqrt{2\pi})\ and\ (\sqrt{2\pi}, +\infty)$
$R_2 = (-\sqrt{2\pi}, \sqrt{2\pi})$

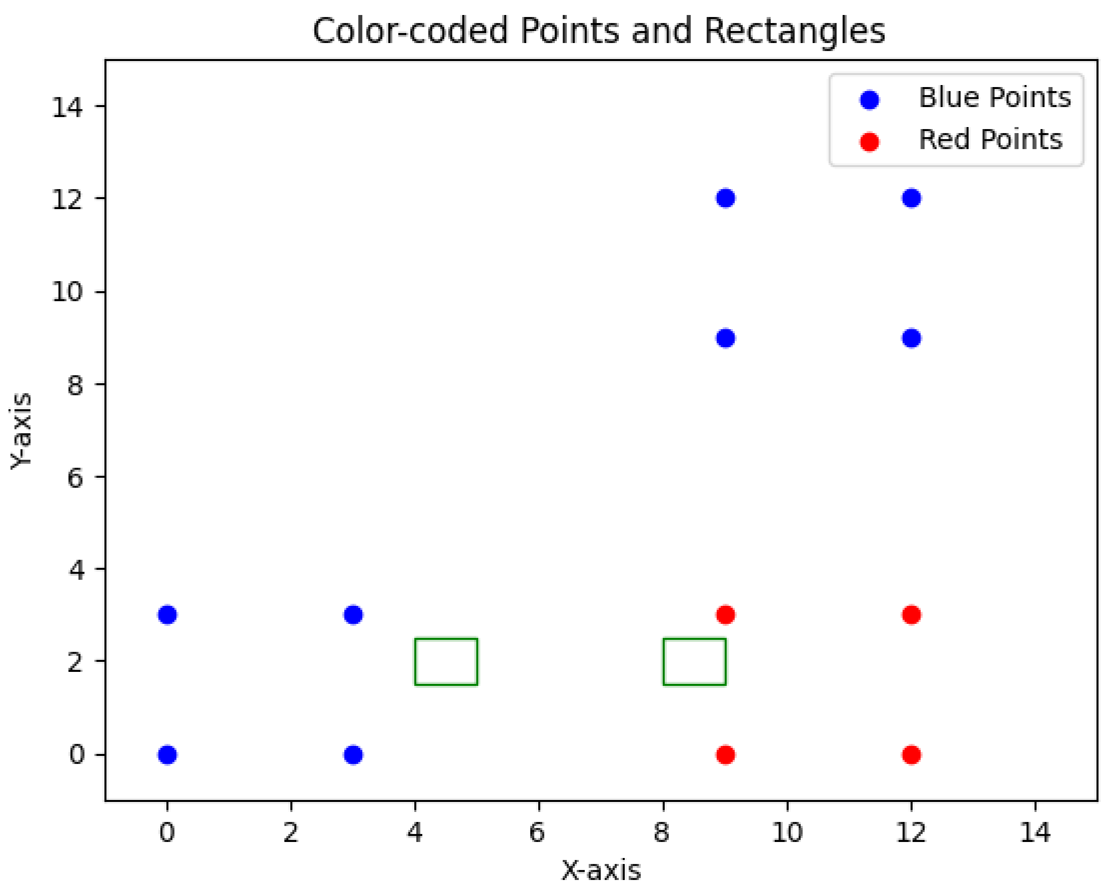
The average risk will be:
$r = \sum_{k=1}^{M} \left(\sum_{k=1}^{M} \lambda_{ki} \int_{R_i} p(x|\omega_k) dx \right) P(\omega_k) =$
$= 0.5\lambda \int_{-\sqrt{2\pi}}^{\sqrt{2\pi}} \frac{1}{2\pi} exp(-\frac{x^2}{2}) dx\, 0.5 = 0.247\lambda$

(c)
The decision regions change significantly because we considered the ω_2 errors more significant. Although the first classifier is more accurate depending on the significance of the errors using Λ we can manipulate the decision regions to ensure that one error is highly unlikely.

Exercise 31

In [65]:
```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Define points
blue_points = np.array([[0, 0], [3, 0], [0, 3], [3, 3], [9, 9], [12, 9], [9,
red_points = np.array([[9, 0], [12, 0], [9, 3], [12, 3]])
rect_points = np.array([[4, 1.5], [8, 1.5]])

# Create a figure and axis
fig, ax = plt.subplots()

# Plot blue points
ax.scatter(blue_points[:, 0], blue_points[:, 1], color='blue', label='Blue P

# Plot red points
ax.scatter(red_points[:, 0], red_points[:, 1], color='red', label='Red Point

# Plot rectangle points and draw rectangle around them
for point in rect_points:
    rect = patches.Rectangle((point[0], point[1]), 1, 1, linewidth=1, edgecc
    ax.add_patch(rect)

# Set axis limits
ax.set_xlim(-1, 15)
ax.set_ylim(-1, 15)

# Set labels and title
ax.set_xlabel('X-axis')
```

```python
ax.set_ylabel('Y-axis')
ax.set_title('Color-coded Points and Rectangles')

# Add legend
ax.legend()

# Show the plot
plt.show()
```



(a) To classify the data points we will use the posterior probabilities.

$P(\omega_1|x) = \frac{p(x|\omega_1)P(\omega_1)}{p(x)}$

$P(\omega_2|x) = \frac{p(x|\omega_2)P(\omega_2)}{p(x)}$

since the denominator is common we can simply compare the numerator

The $p(x|\omega_j)$ is calculated below using python.

$p(x|\omega_1)P(\omega_1) = p(x|\omega_1)\frac{8}{12} = \frac{0.04}{3}$

$p(x|\omega_2)P(\omega_2) = p(x|\omega_2)\frac{4}{12} = \frac{2e-7}{3}$

Thus x is classified to ω_1

$p(x'|\omega_1)P(\omega_1) = p(x'|\omega_1)\frac{8}{12} = \frac{2.4e-7}{3}$

$p(x'|\omega_2)P(\omega_2) = p(x'|\omega_2)\frac{4}{12} = \frac{0.04}{3}$

Thus x' is classified to ω_2

(b) The ω_1 class could be two 2D gaussian distribution and the ω_2 one 2D gaussian distribution.

In [108…

```
import numpy as np

# Given data points for class ω1 and ω2
class_1_points = np.array([[0, 0], [3, 0], [0, 3], [3, 3], [9, 9], [12, 9],
class_2_points = np.array([[9, 0], [12, 0], [9, 3], [12, 3]])

# Points to classify
x = np.array([4, 1.5])
x_prime = np.array([8, 1.5])

# Bandwidth (window size)
h = 1

# Gaussian kernel function
def gaussian_kernel(u):
    return (1 / np.sqrt(2 * np.pi)) * np.exp(-0.5 * u**2)

# Calculate PDF estimates for class ω1
pdf_w1_x = np.mean(gaussian_kernel(np.linalg.norm(x - class_1_points, axis=1
pdf_w1_x_prime = np.mean(gaussian_kernel(np.linalg.norm(x_prime - class_1_po

# Calculate PDF estimates for class ω2
pdf_w2_x = np.mean(gaussian_kernel(np.linalg.norm(x - class_2_points, axis=1
pdf_w2_x_prime = np.mean(gaussian_kernel(np.linalg.norm(x_prime - class_2_po

print(pdf_w1_x, pdf_w1_x_prime)
print(pdf_w2_x, pdf_w2_x_prime)
```

```
0.019649960264157284 1.206668085584657e-07
2.4133358018426484e-07 0.03929992052831456
```

Exercise 32

In [109…

```
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt

Dataset = sio.loadmat('HW8.mat')
train_x = Dataset['train_x']
train_y= Dataset['train_y']

test_x = Dataset['test_x']
test_y = Dataset['test_y']
```

In [110…

```
import numpy as np
from scipy.stats import multivariate_normal

# Given data for training
train_y = np.squeeze(train_y)

# Separate data into classes
```

```python
class_1_data = train_x[train_y == 1]
class_2_data = train_x[train_y == 2]

# Estimate class priors
P_omega_1 = len(class_1_data) / len(train_x)
P_omega_2 = len(class_2_data) / len(train_x)

# Estimate means
mu_1 = np.mean(class_1_data, axis=0)
mu_2 = np.mean(class_2_data, axis=0)

# Estimate covariance matrices
Sigma_1 = np.cov(class_1_data, rowvar=False)
Sigma_2 = np.cov(class_2_data, rowvar=False)

# Print the estimated parameters
print("Estimated Parameters:")
print(f"P(omega_1): {P_omega_1}")
print(f"P(omega_2): {P_omega_2}")
print(f"mu_1: {mu_1}")
print(f"mu_2: {mu_2}")
print(f"Sigma_1: {Sigma_1}")
print(f"Sigma_2: {Sigma_2}")
```

```
Estimated Parameters:
P(omega_1): 0.5
P(omega_2): 0.5
mu_1: [0.14549472 0.11840199]
mu_2: [ 2.07024339 -1.89136529]
Sigma_1: [[3.63737014 1.74128017]
 [1.74128017 4.22056748]]
Sigma_2: [[4.71777486 2.6006903 ]
 [2.6006903  4.37763924]]
```

In [127…

```python
# Initialize the column vector for class labels
Btest_y = np.zeros((len(test_x), 1), dtype=int)

# Apply the Bayes classifier to each test vector
for i in range(len(test_x)):
    x_i = test_x[i]

    # Calculate posterior probabilities using Bayes classifier
    posterior_prob_1 = P_omega_1 * multivariate_normal.pdf(x_i, mu_1, Sigma_
    posterior_prob_2 = P_omega_2 * multivariate_normal.pdf(x_i, mu_2, Sigma_

    # Assign the class label based on the maximum posterior probability
    Btest_y[i] = 1 if posterior_prob_1 > posterior_prob_2 else 2

#print(Btest_y)
```

In [112…

```python
# Ensure test_y is a 1D array
test_y = np.squeeze(test_y)

# Count the number of misclassifications
misclassifications = np.sum(test_y != Btest_y.flatten())
```

```
# Calculate the error classification probability
error_probability = misclassifications / len(test_y)

# Print the results
print(f"Number of Misclassifications: {misclassifications}")
print(f"Error Classification Probability: {error_probability * 100:.2f}%")
```

```
Number of Misclassifications: 30
Error Classification Probability: 15.00%
```

In [103…

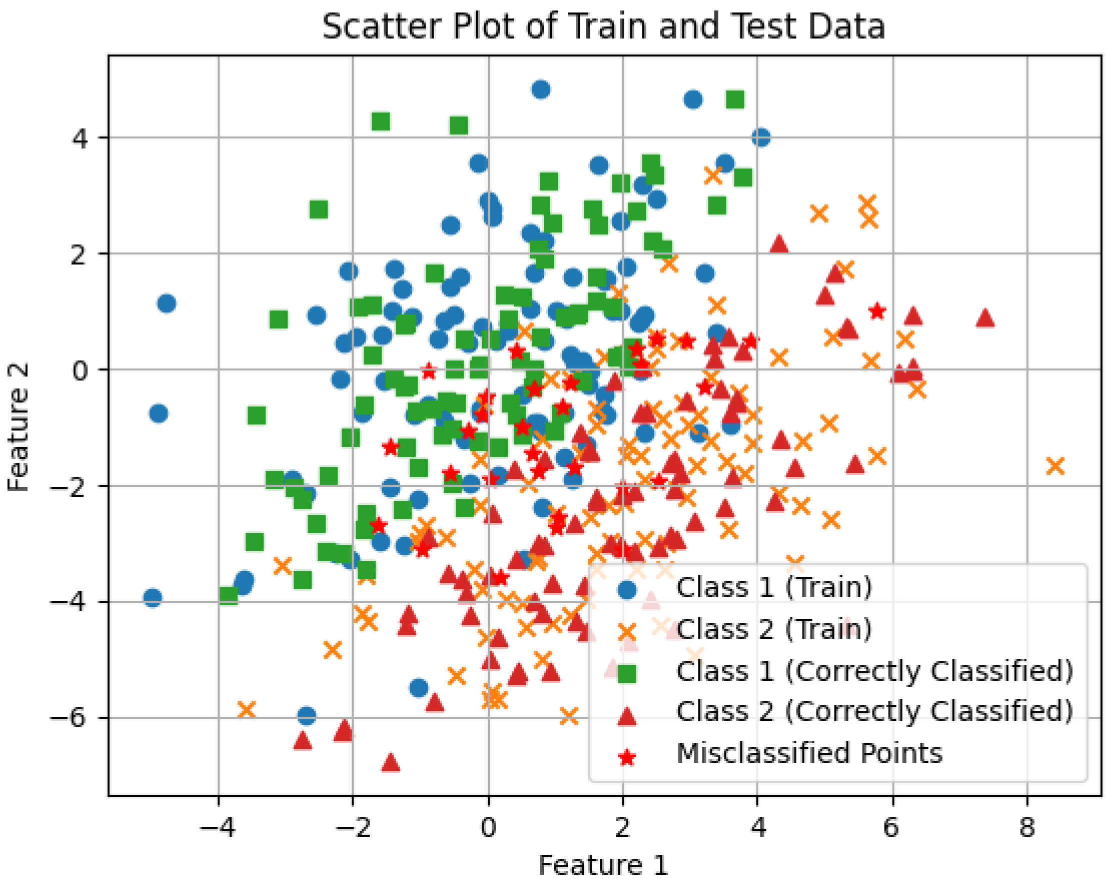
```
test_x.shape
```

Out[103…

```
(200, 2)
```

In [120…

```
import matplotlib.pyplot as plt

# Initialize a list to store the indices of misclassified points
misclassified_indices = []
Btest_y = np.squeeze(Btest_y)

# Find the indices of misclassified points
for i in range(len(test_y)):
    if test_y[i] != Btest_y[i]:
        misclassified_indices.append(i)

# Scatter plot for training data
plt.scatter(train_x[train_y == 1][:, 0], train_x[train_y == 1][:, 1], label=
plt.scatter(train_x[train_y == 2][:, 0], train_x[train_y == 2][:, 1], label=

# Scatter plot for correctly classified test data
plt.scatter(test_x[(Btest_y == 1) & (test_y == 1)][:, 0], test_x[(Btest_y ==
            label='Class 1 (Correctly Classified)', marker='s')
plt.scatter(test_x[(Btest_y == 2) & (test_y == 2)][:, 0], test_x[(Btest_y ==
            label='Class 2 (Correctly Classified)', marker='^')

# Scatter plot for misclassified test data
plt.scatter(test_x[misclassified_indices][:, 0], test_x[misclassified_indice
            label='Misclassified Points', marker='*', color='red')

# Customize the plot
plt.title('Scatter Plot of Train and Test Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



Scatter Plot of Train and Test Data
Feature 2
Feature 1
4
2
0
−2
−4
−6
−4
−2
0
2
4
6
8
Class 1 (Train)
Class 2 (Train)
Class 1 (Correctly Classified)
Class 2 (Correctly Classified)
Misclassified Points