# Assignment 5

## Question 1

## Apply Bayes Decision Rule for two-class problems

**ALGORITHM: Bayes Decision Rule for Two-Class Problems**

INPUT:
- Training dataset {(X_i, y_i)}, where y_i ∈ {0, 1}
- Test sample X_test
- Prior probabilities $P(C_0)$ and $P(C_1)$

OUTPUT:
- Predicted class label y_pred ∈ {0, 1}

PROCEDURE:
1. Separate training data by class
   $X_0$ = {X_i | y_i = 0}
   $X_1$ = {X_i | y_i = 1}

2. Estimate class parameters
   $\mu_0 = \text{mean}(X_0)$
   $\mu_1 = \text{mean}(X_1)$
   $\Sigma_0 = \text{covariance}(X_0)$
   $\Sigma_1 = \text{covariance}(X_1)$

3. For each test sample x in X_test:
   a. Calculate likelihood for each class
      $P(x|C_0) = N(x; \mu_0, \Sigma_0)$ // Gaussian probability density
      $P(x|C_1) = N(x; \mu_1, \Sigma_1)$

   b. Calculate posterior probabilities using Bayes' theorem
      $P(C_0|x) \propto P(x|C_0) \times P(C_0)$
      $P(C_1|x) \propto P(x|C_1) \times P(C_1)$

   c. Apply decision rule
      If $P(C_1|x) > P(C_0|x)$ then
         Assign x to class $C_1$ (y_pred = 1)

Else
  Assign x to class $C_0$ (y_pred = 0)

RETURN: y_pred

**Complete the code below**
**Note: you have to insert your code in the Red  color section**

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

def bayes_decision_rule(X_train, y_train, X_test, prior_0=0.5, prior_1=0.5):
   """
Steps
   # Separate the data by class
   # Calculate mean vectors
    # Calculate covariance matrices

   # Create probability distribution for each class

   # Calculate likelihoods for each test point

   # Calculate posteriors using Bayes rule

   # Classify based on highest posterior probability
      """
   return predicted_labels

if __name__ == "__main__":
   # Generate some example data
   np.random.seed(42)

   # Class 0 data (100 points)
   mean0 = [0, 0]
   cov0 = [[1, 0], [0, 1]]
   class0_data = np.random.multivariate_normal(mean0, cov0, 100)

   # Class 1 data (100 points)
   mean1 = [3, 3]
   cov1 = [[1, 0], [0, 1]]
   class1_data = np.random.multivariate_normal(mean1, cov1, 100)
```

---

```
# Combine data
X = np.vstack((class0_data, class1_data))
y = np.hstack((np.zeros(100), np.ones(100)))

# Use first 80% for training, last 20% for testing
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Apply Bayes decision rule
y_pred = bayes_decision_rule(X_train, y_train, X_test)
```

Calculate accuracy
Plot results

# Question 2

# Bayes maximum likelihood rule classifier for two class

**ALGORITHM: Bayes Maximum Likelihood Classifier**

INPUT:
- Training dataset {(X_i, y_i)}, where y_i ∈ {0, 1}
- Test samples X_test

OUTPUT:
- Predicted class labels y_pred ∈ {0, 1}

PROCEDURE:
1. Separate training data by class
   $X_0$ = {X_i | y_i = 0}  // All samples from class 0
   $X_1$ = {X_i | y_i = 1}  // All samples from class 1

2. Compute maximum likelihood estimates of parameters
   a. For class 0:
      $\mu_0 = (1/|X_0|) * \sum x$,  for all $x \in X_0$  // Sample mean
      $\Sigma_0 = (1/|X_0|) * \sum (x - \mu_0)(x - \mu_0)^T$,  for all $x \in X_0$  // Sample covariance

   b. For class 1:
      $\mu_1 = (1/|X_1|) * \sum x$,  for all $x \in X_1$  // Sample mean
      $\Sigma_1 = (1/|X_1|) * \sum (x - \mu_1)(x - \mu_1)^T$,  for all $x \in X_1$  // Sample covariance

---

3. For each test sample x in X_test:
   a. Calculate likelihood for each class assuming Gaussian distribution
      $P(x|C_0) = (1/\sqrt{((2\pi)^d |\Sigma_0|)}) * \exp(-(1/2)(x - \mu_0)^T\Sigma_0^{-1}(x - \mu_0))$
      $P(x|C_1) = (1/\sqrt{((2\pi)^d |\Sigma_1|)}) * \exp(-(1/2)(x - \mu_1)^T\Sigma_1^{-1}(x - \mu_1))$

      Where:
      - d is the dimensionality of the feature space
      - $|\Sigma|$ denotes the determinant of matrix $\Sigma$
      - $\Sigma^{-1}$ denotes the inverse of matrix $\Sigma$

   b. Apply maximum likelihood decision rule
      If $P(x|C_1) > P(x|C_0)$ then
        Assign x to class 1 (y_pred = 1)
      Else
        Assign x to class 0 (y_pred = 0)

RETURN: y_pred

**Complete the code below**
**Note: you have to insert your code in the Red color section**

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

def bayes_maximum_likelihood(X_train, y_train, X_test):

    # Separate data by class
    # Compute maximum likelihood estimates
    # For Gaussian, ML estimates are sample mean and sample covariance
    # Create probability distributions
    # Calculate likelihoods P(x|class)
    # Assign to class with maximum likelihood

    return y_pred

# Example usage
if __name__ == "__main__":
    # Generate synthetic data
    np.random.seed(42)

    # Class 0: 100 samples
```

---

```
mean0 = [0, 1]
cov0 = [[1, 0.2], [0.2, 1]]
X0 = np.random.multivariate_normal(mean0, cov0, 100)

# Class 1: 100 samples
mean1 = [2, 3]
cov1 = [[1, -0.2], [-0.2, 1]]
X1 = np.random.multivariate_normal(mean1, cov1, 100)

# Combine datasets
X = np.vstack((X0, X1))
y = np.hstack((np.zeros(100), np.ones(100)))

# Shuffle the data
indices = np.random.permutation(len(X))
X = X[indices]
y = y[indices]

# Split into train/test (80/20)
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Apply the classifier
y_pred = bayes_maximum_likelihood(X_train, y_train, X_test)

# Calculate accuracy

# Plot results
```

# Question 3

# minimum distance classifier for two class

**ALGORITHM: Minimum Distance Classifier for Two Classes**

INPUT:
- Training dataset {(X_i, y_i)}, where y_i ∈ {0, 1}
- Test samples X_test

OUTPUT:
- Predicted class labels y_pred ∈ {0, 1}

------------------------------------------------------------------------------------------------------------------

PROCEDURE:
1. Compute class centroids (prototypes)
   a. For class 0:
      $\mu_0 = (1/|X_0|) * \sum x$, for all $x \in \{X\_i \mid y\_i = 0\}$

   b. For class 1:
      $\mu_1 = (1/|X_1|) * \sum x$, for all $x \in \{X\_i \mid y\_i = 1\}$

2. For each test sample x in X_test:
   a. Calculate Euclidean distance to each centroid
      $d_0 = ||x - \mu_0|| = \sqrt{[(x - \mu_0)^T(x - \mu_0)]}$
      $d_1 = ||x - \mu_1|| = \sqrt{[(x - \mu_1)^T(x - \mu_1)]}$

   b. Apply minimum distance decision rule
      If $d_1 < d_0$ then
         Assign x to class 1 (y_pred = 1)
      Else
         Assign x to class 0 (y_pred = 0)

RETURN: y_pred

**Complete the code below**
**Note: you have to insert your code in the Red color section**

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

def minimum_distance_classifier(X_train, y_train, X_test):

    Minimum Distance Classifier for two classes.

     # Compute class prototypes (centroids)

    # Classify test points based on minimum Euclidean distance
      # Assign to class with minimum distance

    return y_pred

# Example usage
if __name__ == "__main__":
    # Generate synthetic data
    np.random.seed(42)
```

---

```
# Class 0: 100 samples
mean0 = [0, 1]
cov0 = [[1, 0], [0, 1]]
X0 = np.random.multivariate_normal(mean0, cov0, 100)

# Class 1: 100 samples
mean1 = [3, 3]
cov1 = [[1, 0], [0, 1]]
X1 = np.random.multivariate_normal(mean1, cov1, 100)

# Combine datasets
X = np.vstack((X0, X1))
y = np.hstack((np.zeros(100), np.ones(100)))

# Shuffle the data
indices = np.random.permutation(len(X))
X = X[indices]
y = y[indices]

# Split into train/test (80/20)
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Apply the classifier
y_pred = minimum_distance_classifier(X_train, y_train, X_test)

# Calculate accuracy

# Plot results
```