IE406 Machine Learning - Assignment 6

202201140 | Harsh Gajjar

1. Generate the set of points A and B in R2, each consisting of 2000 data points from a bi-variate normal distribution. The set A and B has been drawn from the N (μ1, Σ1) and N(μ2, Σ2). Let us fix the μ1 = [-1,-1] and μ2 = [2,1]. Separate the 500 data points from each class as a testing set. Plot the optimal Bayesian decision boundary and compute the testing accuracy on test set for three following cases.

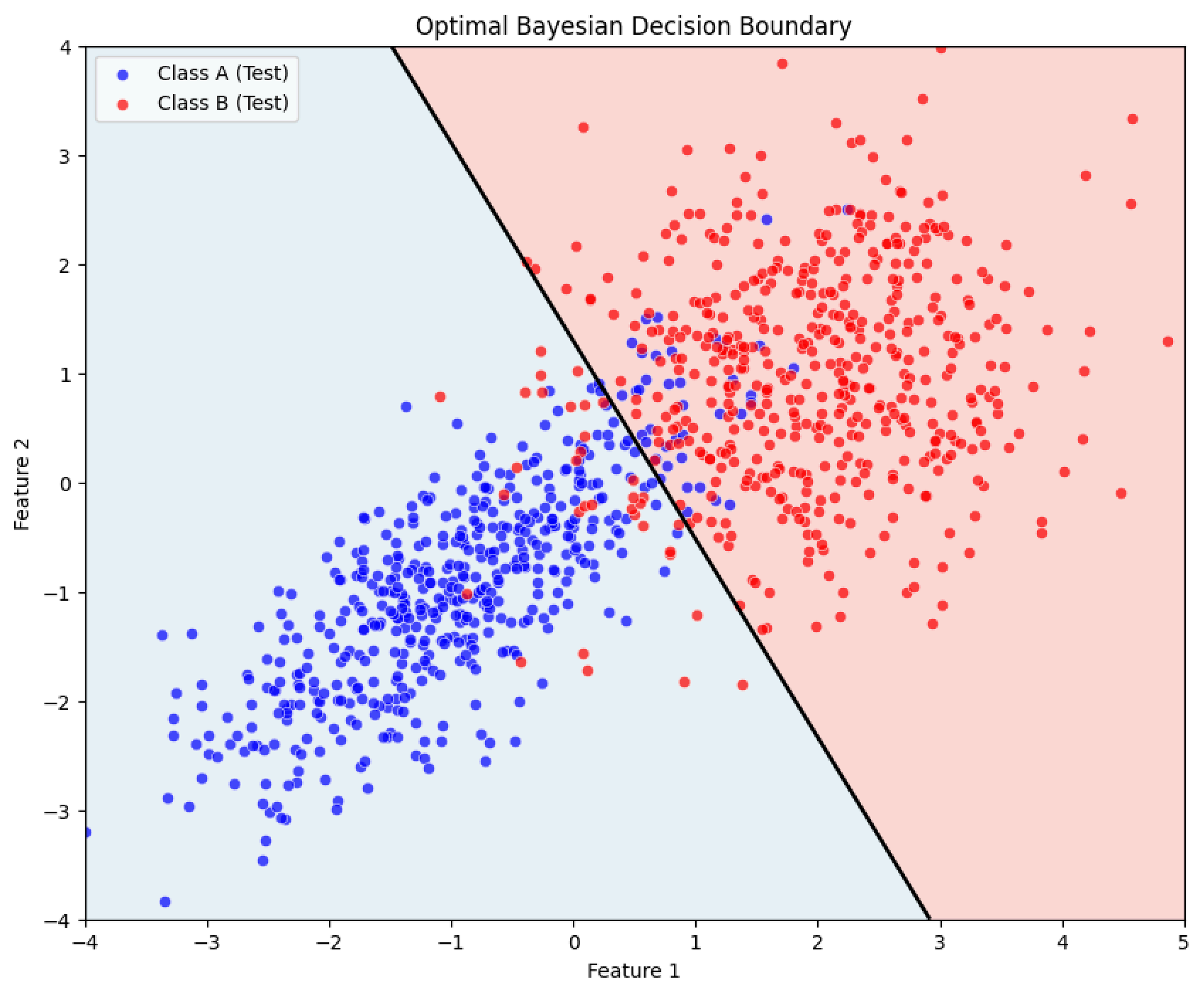
(A) $\Sigma_1 = \Sigma_2 = I.$

(B) $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix}.$

(C) $\Sigma_1 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$ $\Sigma_2 = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}$

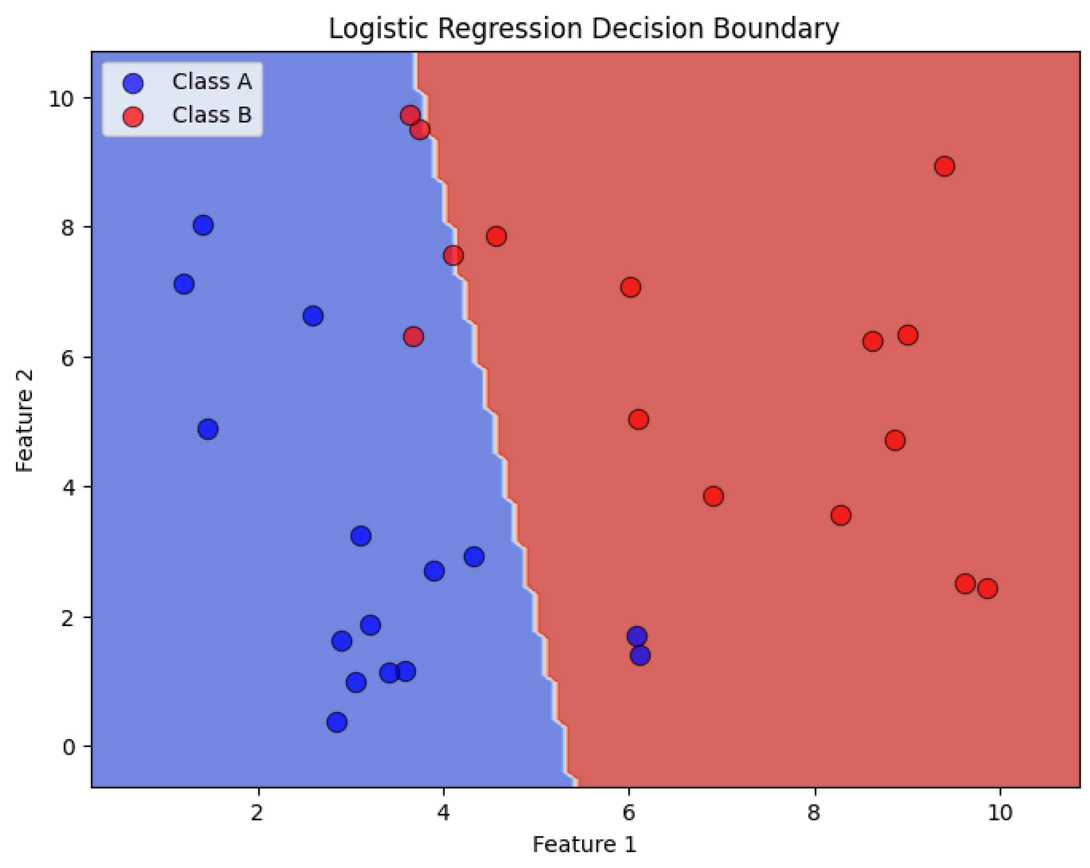
In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
import seaborn as sns

# Set the mean vectors and covariance matrices
mu1 = np.array([-1, -1])
mu2 = np.array([2, 1])
sigma1 = np.array([[1, 0.8], [0.8, 1]])
sigma2 = np.array([[1, 0.1], [0.1, 1]])

# Generate 2000 data points for each class
A = np.random.multivariate_normal(mu1, sigma1, 2000)
B = np.random.multivariate_normal(mu2, sigma2, 2000)

# Separate 500 data points from each class as a testing set
A_train, A_test = A[:1500], A[1500:]
B_train, B_test = B[:1500], B[1500:]

# Combine training and testing sets
X_train = np.vstack((A_train, B_train))
Y_train = np.hstack((np.zeros(1500), np.ones(1500)))
X_test = np.vstack((A_test, B_test))
Y_test = np.hstack((np.zeros(500), np.ones(500)))

# Calculate the optimal Bayesian decision boundary
x_min, x_max = -4, 5
y_min, y_max = -4, 4
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
grid = np.c_[xx.ravel(), yy.ravel()]
inv_sigma1 = np.linalg.inv(sigma1)
inv_sigma2 = np.linalg.inv(sigma2)
w = inv_sigma2 @ mu2 - inv_sigma1 @ mu1
b = -0.5 * (mu2.T @ inv_sigma2 @ mu2 - mu1.T @ inv_sigma1 @ mu1) + np.log(1500 / 1500)
z = grid @ w + b
z = z.reshape(xx.shape)

# Calculate accuracy on the testing set
Y_pred_test = (X_test @ w + b > 0).astype(int)
print(f"Testing Accuracy: {accuracy_score(Y_test, Y_pred_test) * 100:.2f}%")

# Plot testing data points
plt.figure(figsize=(10, 8))
plt.contourf(xx, yy, z, alpha=0.3, levels=[-np.inf, 0], colors='lightblue', zorder=1)
plt.contourf(xx, yy, z, alpha=0.3, levels=[0, np.inf], colors='salmon', zorder=1)
sns.scatterplot(x=A_test[:, 0], y=A_test[:, 1], color='blue', alpha=0.7, label='Class A (Test)', edgecolor='w', linewidth=0.5)
sns.scatterplot(x=B_test[:, 0], y=B_test[:, 1], color='red', alpha=0.7, label='Class B (Test)', edgecolor='w', linewidth=0.5)

# Plot the decision boundary
plt.contour(xx, yy, z, levels=[0], linewidths=2, colors='black', zorder=2)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Optimal Bayesian Decision Boundary')
plt.legend(loc='upper left')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.show()
```

```
Testing Accuracy: 92.70%
```



In [81]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

def generate_data(n_samples):
    np.random.seed(42)
    X = np.random.rand(n_samples, 2) * 10
    Y = (X[:, 0] + X[:, 1] > 10).astype(int)
    return X, Y

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def cost_function(theta, X, Y):
    m = len(Y)
    h = sigmoid(X @ theta)
    return (-1/m) * np.sum(Y * np.log(h + 1e-5) + (1 - Y) * np.log(1 - h + 1e-5))

def gradient_descent(X, Y, theta, alpha, iterations):
    m = len(Y)
    cost_history = []

    for _ in range(iterations):
        h = sigmoid(X @ theta)
        gradient = (X.T @ (h - Y)) / m
        theta -= alpha * gradient
        cost_history.append(cost_function(theta, X, Y))

    return theta, cost_history

def predict(X, theta):
    probabilities = sigmoid(X @ theta)
    return [1 if p >= 0.5 else 0 for p in probabilities]

def plot_decision_boundary(X, Y, theta):
    x1_min, x1_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    x2_min, x2_max = X[:, 2].min() - 1, X[:, 2].max() + 1

    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max, 100), np.linspace(x2_min, x2_max, 100))
    Z = predict(np.c_[np.ones((xx1.ravel().shape[0], 1)), xx1.ravel(), xx2.ravel()], theta)
    Z = np.array(Z).reshape(xx1.shape)

    plt.figure(figsize=(8, 6))
    plt.contourf(xx1, xx2, Z, alpha=0.8, cmap='coolwarm')
    sns.scatterplot(x=X[Y == 0, 1], y=X[Y == 0, 2], alpha=0.7, color='blue', edgecolor='k', marker='o', s=80)
    sns.scatterplot(x=X[Y == 1, 1], y=X[Y == 1, 2], alpha=0.7, color='red', edgecolor='k', marker='o', s=80)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Logistic Regression Decision Boundary')
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.legend(loc='upper left', labels=['Class A', 'Class B'])
    plt.show()

def logistic_regression(X_train, Y_train, X_test, Y_test, alpha=0.01, iterations=1000):
    X_train = np.insert(X_train, 0, 1, axis=1)
    X_test = np.insert(X_test, 0, 1, axis=1)
    theta = np.zeros(X_train.shape[1])
    theta, cost_history = gradient_descent(X_train, Y_train, theta, alpha, iterations)
    predictions = predict(X_test, theta)

    accuracy = np.mean(predictions == Y_test) * 100
    print(f'Test Accuracy: {accuracy:.2f}%')
    plot_decision_boundary(X_test, Y_test, theta)

X, Y = generate_data(n_samples=100)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
logistic_regression(X_train, Y_train, X_test, Y_test)
```

```
Test Accuracy: 80.00%
```


Logistic Regression Decision Boundary
Class A
Class B
Feature 2
Feature 1
0
2
4
6
8
10
2
4
6
8
10