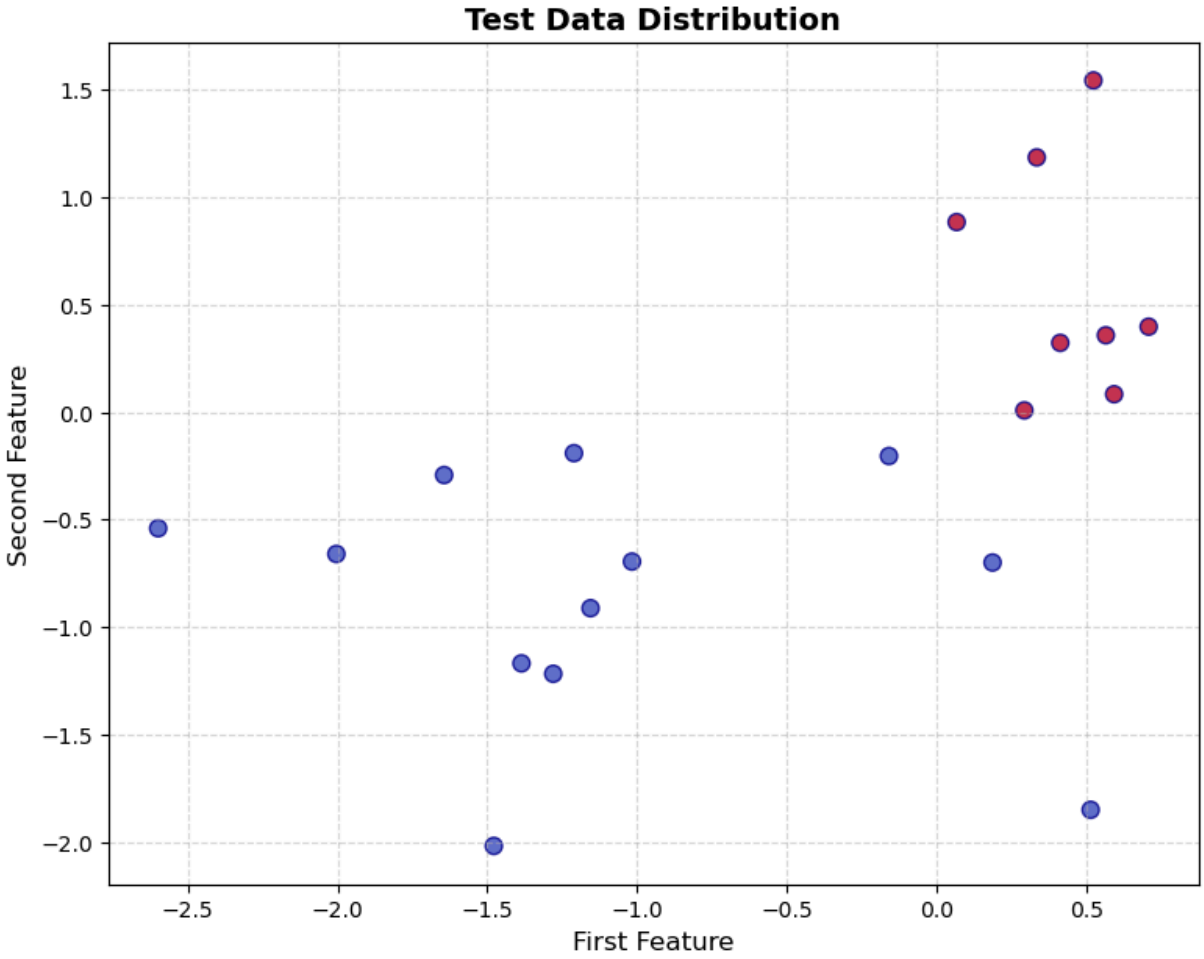
In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

In [ ]:
```
dataset = np.loadtxt('data.txt', delimiter='\t')

X, y = dataset[:, :-1], dataset[:, -1]

print(f"Dimensions of X: {X.shape}")
print(f"Dimensions of y: {y.shape}")

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("Data has been successfully processed.")
```

```
Dimensions of X: (100, 2)
Dimensions of y: (100,)
Data has been successfully processed.
```

In [ ]:
```
# Visualizing the training data
plt.figure(figsize=(9, 7))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='coolwarm', edgecolor='da
plt.title('Training Data Distribution', fontsize=14, fontweight='bold')
plt.xlabel('First Feature', fontsize=12)
plt.ylabel('Second Feature', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()

# Visualizing the test data
plt.figure(figsize=(9, 7))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='coolwarm', edgecolor='darkb
plt.title('Test Data Distribution', fontsize=14, fontweight='bold')
plt.xlabel('First Feature', fontsize=12)
plt.ylabel('Second Feature', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```


Training Data Distribution
Second Feature
First Feature
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Test Data Distribution
Second Feature
First Feature
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5

In [ ]:
```
class Perceptron:
    def __init__(self, num_inputs):
        self.num_inputs = num_inputs
        self.weights = np.zeros(num_inputs)
        self.bias = 0

    def predict(self, inputs):
        linear_combination = np.dot(inputs, self.weights) + self.bias
        return np.where(linear_combination >= 0, 1, 0)

    def compute_error(self, inputs, actual_output):
        predicted_output = self.predict(inputs)
        return actual_output - predicted_output

    def train(self, inputs, targets, epochs):
        for epoch in range(epochs):
            for index in range(targets.shape[0]):
                error = self.compute_error(inputs[index], targets[index])
                self.weights += error * inputs[index]
                self.bias += error
        print("Training finished.")

    def evaluate(self, inputs, targets):
        predictions = self.predict(inputs)
        accuracy = np.mean(predictions == targets) * 100
        return accuracy
```

In [ ]:
```
num_features = X_train.shape[1]
model = Perceptron(num_features)

epochs = 5
model.train(X_train, y_train, epochs)

print("Weights after training:", model.weights)
print("Bias after training:", model.bias)
```

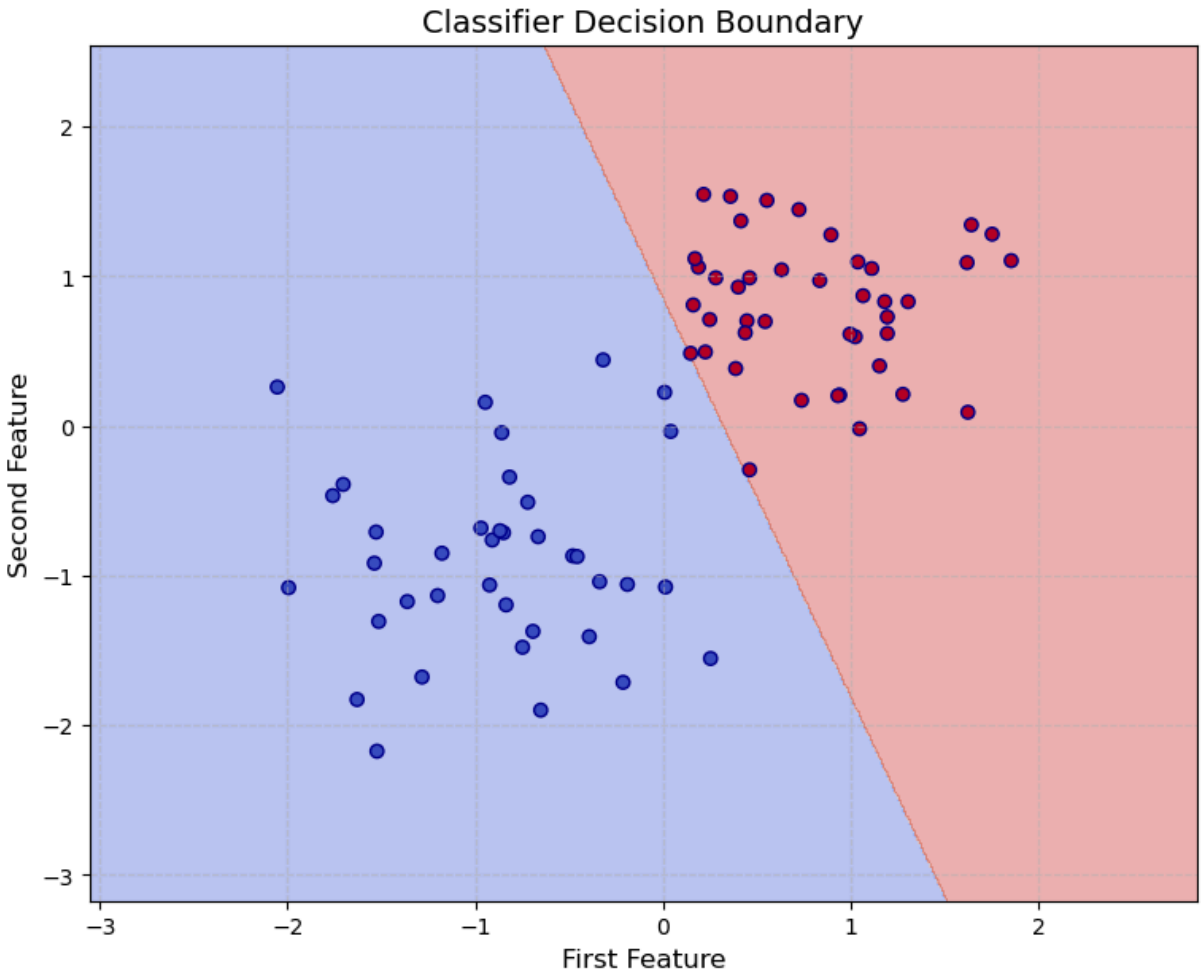
```
Training finished.
Weights after training: [3.14822119 1.18210243]
Bias after training: -1.0
```

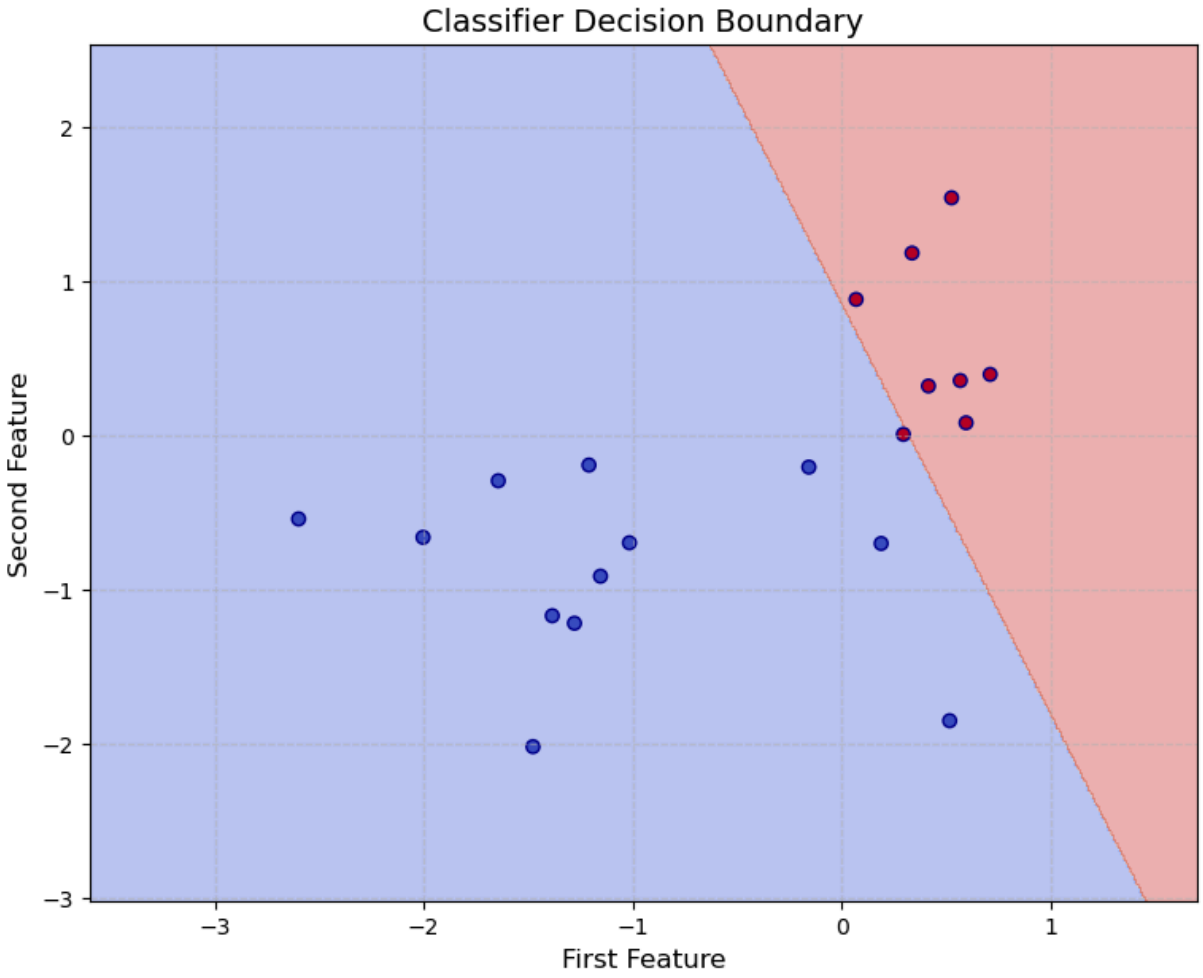
In [ ]:
```
train_accuracy = model.evaluate(X_train, y_train)
print(f'Training Accuracy: {train_accuracy:.2f}%')

test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}%')
```

```
Training Accuracy: 100.00%
Test Accuracy: 95.00%
```

In [ ]:
```
def visualize_decision_boundary(features, labels, classifier):
    x_range_min, x_range_max = features[:, 0].min() - 1, features[:, 0].max() + 1
    y_range_min, y_range_max = features[:, 1].min() - 1, features[:, 1].max() + 1
    xx_grid, yy_grid = np.meshgrid(np.arange(x_range_min, x_range_max, 0.01),
                                   np.arange(y_range_min, y_range_max, 0.01))

    grid_points = np.c_[xx_grid.ravel(), yy_grid.ravel()]
```

```
    predictions = classifier.predict(grid_points)
    predictions = predictions.reshape(xx_grid.shape)

    plt.figure(figsize=(9, 7))
    plt.contourf(xx_grid, yy_grid, predictions, alpha=0.4, cmap='coolwarm')
    plt.scatter(features[:, 0], features[:, 1], c=labels, edgecolor='darkblue', cma
    plt.title('Classifier Decision Boundary', fontsize=14)
    plt.xlabel('First Feature', fontsize=12)
    plt.ylabel('Second Feature', fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.show()

visualize_decision_boundary(X_train, y_train, model)
visualize_decision_boundary(X_test, y_test, model)
```




Classifier Decision Boundary
Second Feature
First Feature
2
1
0
−1
−2
−3
−3
−2
−1
0
1