ESE 417 Homework 3

Problem 1 (Create your own Perceptron algorithm)

(1) Use Python to generate a 2D $(x_i \in R^2)$ linearly separable data set with 50 data points. The data set should have approximately half positive and half negative instances. Create a scatter plot to visualize the data set. Split the data set into training set (60%) and test set (40%).

In [106…

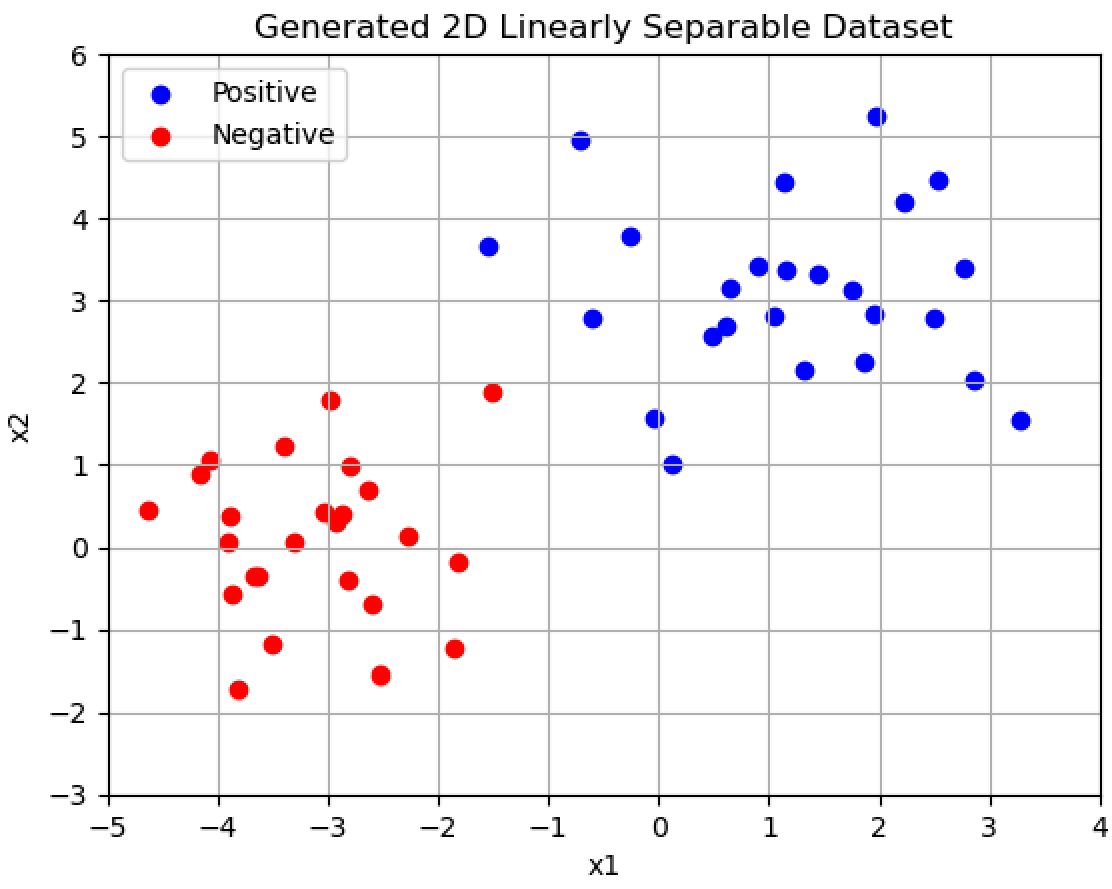
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [106…

```
# Generate 2D linearly separable dataset
np.random.seed(0)

# Create 50 data points with half positive and half negative
point_num = 50 # number of data points

x_positive = np.random.randn(int(point_num/2),2) + np.array([1,3]) # positive data points
y_positive = np.ones(int(point_num/2)) # positive labels

x_negative = np.random.randn(int(point_num/2),2) + np.array([-3,0]) # negative data points
y_negative = -np.ones(int(point_num/2)) # negative labels

x_data = np.vstack((x_positive, x_negative)) # combine positive and negative data points
y_data = np.hstack((y_positive, y_negative)) # combine positive and negative labels
```

In [106…

```
# Scatter plot to vasualize the data set
plt.scatter(x_positive[:,0], x_positive[:,1], color='blue', label='Positive')
plt.scatter(x_negative[:,0], x_negative[:,1], color='red', label='Negative')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Generated 2D Linearly Separable Dataset')
plt.legend()
plt.xlim(-5, 4)
plt.ylim(-3, 6)
plt.grid()
plt.show()
```



In [107…

```
# Split the dataset into training set (60%) and test set (40%)
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.4, random_state
```

(2) Use Python (not sklearn package) to create the Batch Perceptron training algorithm and use the training data set in a.(1) to train a Perceptron model. Plot the error function curve when the training process converges. Create a plot that shows the training instances and the learnt decision boundary.

In [107…

```python
# Batch Perceptron training algorithm
def batch_perceptron_training(x, y, learning_rate=0.01, max_iterations=1000):
    np.random.seed(0)  # set random seed for reproducibility
    num_data = x.shape[0]  # number of data points
    x = np.hstack((np.ones((num_data, 1)), x))  # add bias term to each input vector
    num_features = x.shape[1]  # number of features
    w = np.random.randn(num_features)  # initialize w with random values
    iteration = 0  # initialize iteration counter
    converged = False  # initialize convergence flag
    errors = []  # list to store the perceptron criterion error

    while converged == False and iteration < max_iterations:
        converged = True  # assume convergence
        Mk = []  # set of misclassified data points
        for i in range(num_data):
            if np.dot(w, x[i]) * y[i] <= 0:  # check if the current data point is misclassifie
                Mk.append((x[i], y[i]))  # add the misclassified data point to Mk
                converged = False  # update convergence flag

        # calculate the perceptron criterion error
        error_value = -sum(np.dot(w, x_i) * y_i for x_i, y_i in Mk)
        errors.append(error_value)

        # if there are misclassified points, update the weight vector
        if len(Mk) > 0:
            update = sum([x_i * y_i for x_i, y_i in Mk])  # compute the update vector
            w = w + learning_rate * update # update the weight vector

        iteration = iteration + 1

    return w, errors
```

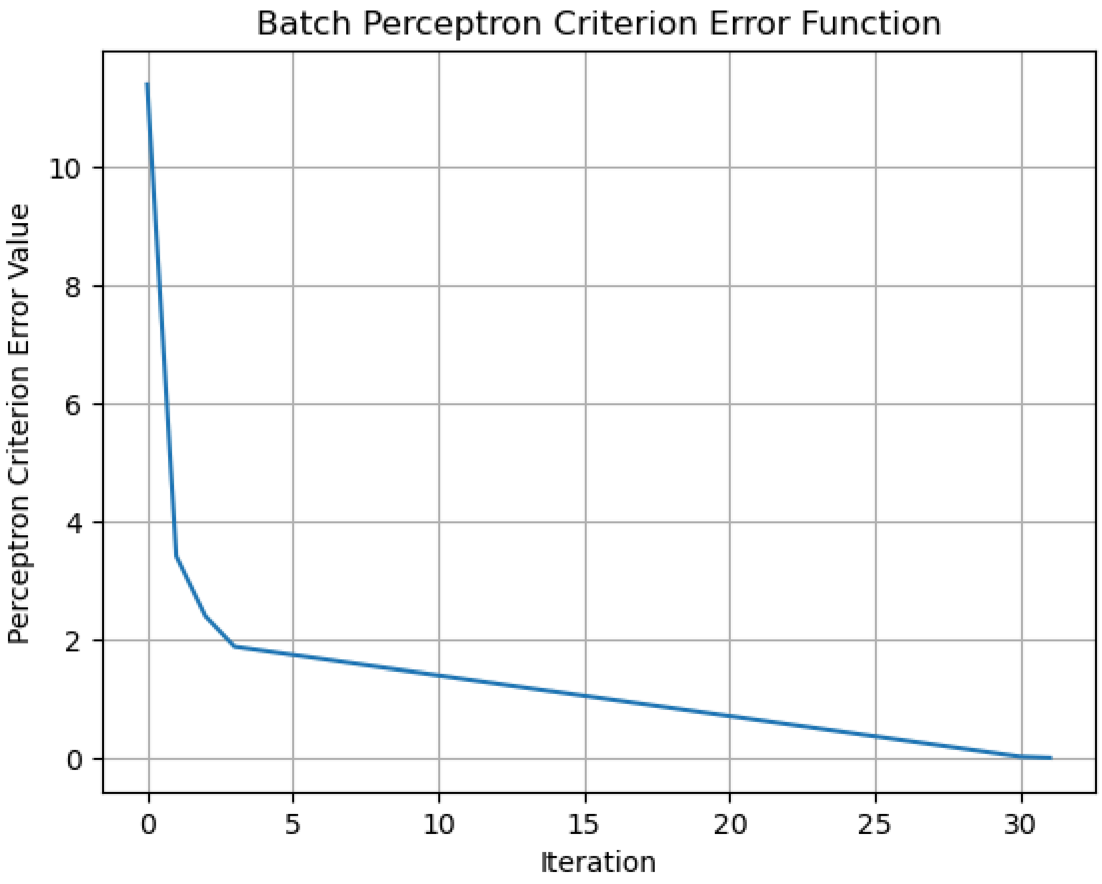
In [107…

```python
# Train the batch perceptron model
w_batch, errors_batch = batch_perceptron_training(x_train, y_train)
print('Batch Perceptron Weight Vector:', w_batch)
# print("Errors in each iteration:", errors_batch)
print(len(errors_batch))
```

```
Batch Perceptron Weight Vector: [1.30405235 1.30564661 0.32697703]
32
```

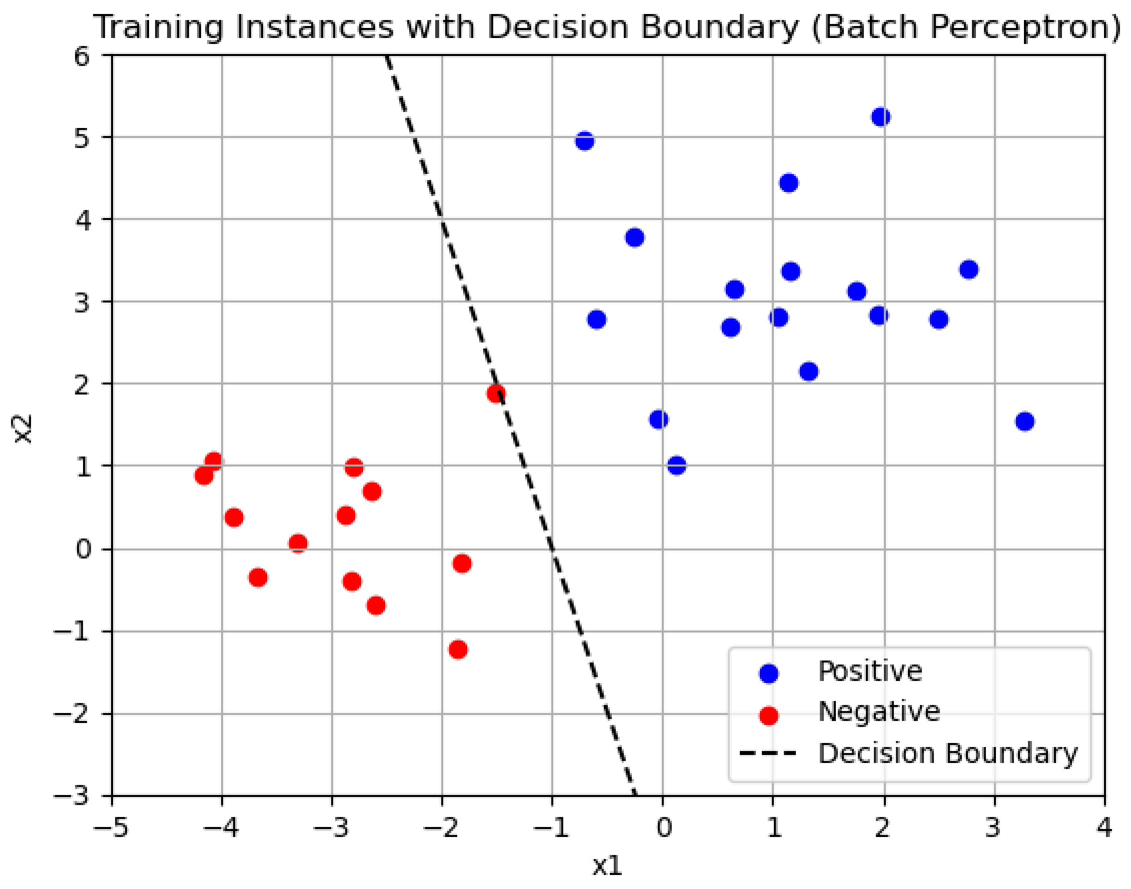
In [107…

```python
# Plot error function cruve
plt.plot(errors_batch)
plt.xlabel('Iteration')
plt.ylabel('Perceptron Criterion Error Value')
plt.title('Batch Perceptron Criterion Error Function')
plt.grid()
plt.show()
```



In [107…

```
# Plot training instances with decision boundary
plt.scatter(x_train[y_train == 1][:, 0], x_train[y_train == 1][:, 1], color='blue', label='Pos
plt.scatter(x_train[y_train == -1][:, 0], x_train[y_train == -1][:, 1], color='red', label='Ne
x_boundary = np.linspace(x_data[:, 0].min(), x_data[:, 0].max(), 100)
y_boundary = - (w_batch[1]/w_batch[2]) * x_boundary - (w_batch[0]/w_batch[2])
plt.plot(x_boundary, y_boundary, 'k--', label='Decision Boundary')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Training Instances with Decision Boundary (Batch Perceptron)')
plt.legend()
plt.xlim(-5, 4)
plt.ylim(-3, 6)
plt.grid()
plt.show()
```



(3) Use the test set in a.(1) to test the trained model in a.(2) and calculate the accuracy (error rate) of the trained model.

In [107…
```
# Define the perceptron prediction function
def perceptron_predict(x, w):
    return np.sign(np.dot(x, w))
```

In [107…
```
# Calculate the accuracy of the batch perceptron model on the test set
y_pred_batch = perceptron_predict(np.hstack((np.ones((x_test.shape[0], 1)), x_test)), w_batch)
accuracy_batch = np.sum(y_pred_batch == y_test) / y_test.shape[0]
print(f'Batch Perceptron Test Accuracy: {accuracy_batch * 100:.2f}%')
```

```
Batch Perceptron Test Accuracy: 100.00%
```

(4) Use Python (not sklearn package) to create the sequential Perceptron training algorithm and use the training data set in a.(1) to train a Perceptron model. Plot the weights vs iterations curve when the training process converges. Create a plot that shows the training instances and the resulting decision boundary.

In [107…
```
def sequential_perceptron(x, y, learning_rate=0.01, max_iterations=1000):
    np.random.seed(0)  # set random seed for reproducibility
    num_data = x.shape[0]  # number of data points
    x = np.hstack((np.ones((num_data, 1)), x))  # add bias term to each input vector
    num_features = x.shape[1]  # number of features
    w = np.random.randn(num_features)  # initialize weight vector with random values
    k = 0  # index of points in the training dataset
    weights_history = [w.copy()]  # track weight values for plotting
    iteration = 0  # initialize iteration counter
    errors = []  # list to store the perceptron criterion error

    while iteration < max_iterations:
        error_value = 0  # initialize the error value for the current iteration
        k = k % num_data  # wrap around index if exceeds dataset size
        if np.dot(w, x[k]) * y[k] <= 0:  # check if the current data point is misclassified
            w = w + learning_rate * y[k] * x[k]  # update weight vector
        k = k + 1 # update k
        weights_history.append(w.copy())  # track weight after update
        # calculate the perceptron criterion error
        for x_i, y_i in zip(x, y):
            if np.dot(w, x_i) * y_i <= 0: # check if the current data point is misclassified
                error_value = error_value - np.dot(w, x_i) * y_i
        errors.append(error_value)
        iteration = iteration + 1  # update iteration counter

    return w, weights_history, errors
```

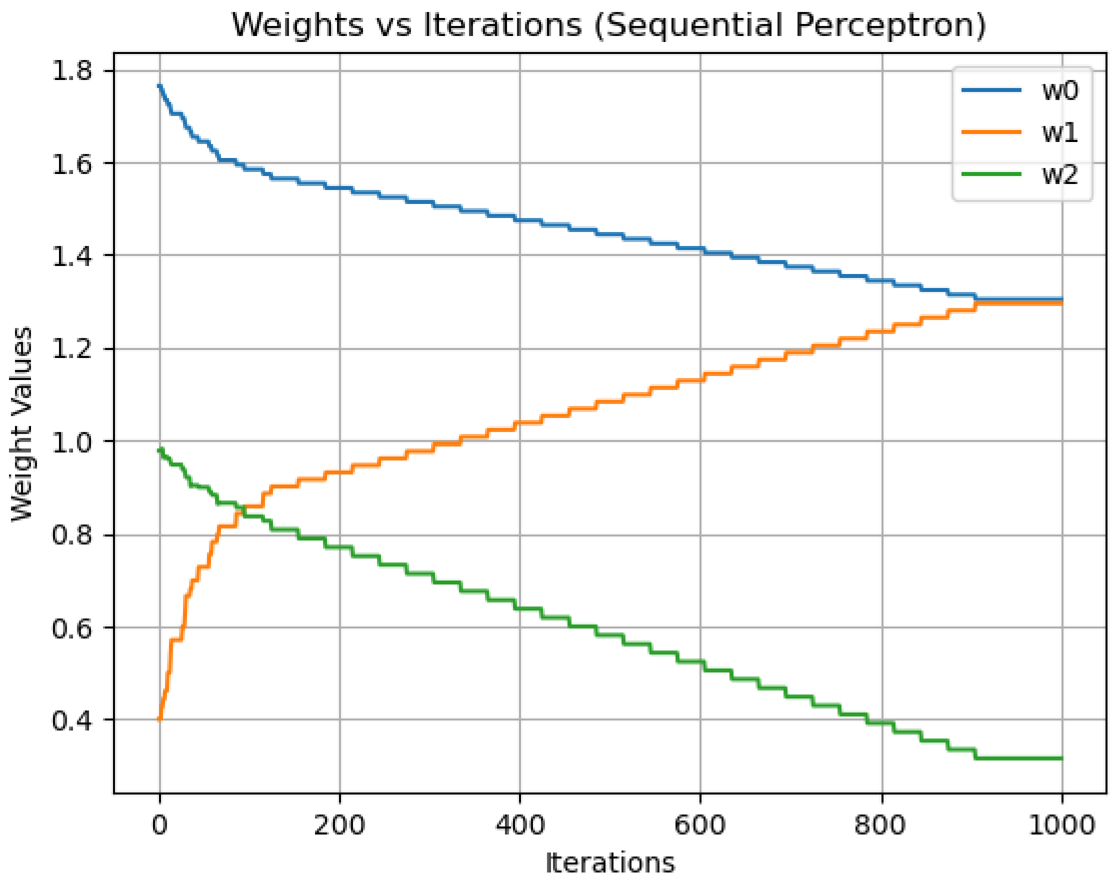
In [107…

```
# Calculate the weight vector and weight history
w_seq, weights_history_seq, errors_seq = sequential_perceptron(x_train, y_train)
print('Sequential Perceptron Weight Vector:', w_seq)
# print(errors_seq)
```

```
Sequential Perceptron Weight Vector: [1.30405235 1.2949785  0.31576927]
```

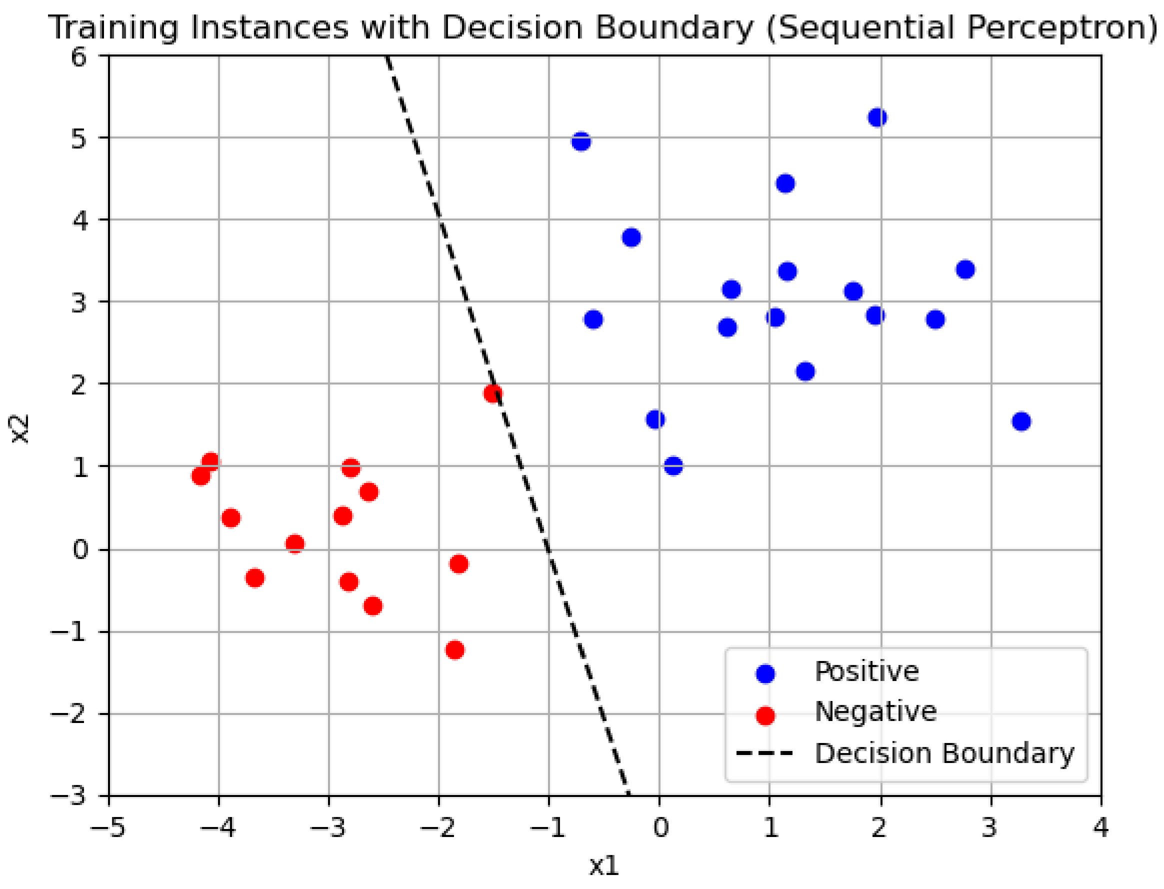
In [107…

```
# Plot weights vs iterations curve for Sequential Perceptron
weights_history_seq = np.array(weights_history_seq)
plt.plot(weights_history_seq[:, 0], label='w0')
plt.plot(weights_history_seq[:, 1], label='w1')
plt.plot(weights_history_seq[:, 2], label='w2')
plt.xlabel('Iterations')
plt.ylabel('Weight Values')
plt.title('Weights vs Iterations (Sequential Perceptron)')
plt.legend()
plt.grid()
plt.show()
```



In [108…

```
plt.scatter(x_train[y_train == 1][:, 0], x_train[y_train == 1][:, 1], color='blue', label='Pos
plt.scatter(x_train[y_train == -1][:, 0], x_train[y_train == -1][:, 1], color='red', label='Ne
x_boundary = np.linspace(x_data[:, 0].min(), x_data[:, 0].max(), 100)
y_boundary = - (w_seq[1]/w_seq[2]) * x_boundary - (w_seq[0]/w_seq[2])
plt.plot(x_boundary, y_boundary, 'k--', label='Decision Boundary')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Training Instances with Decision Boundary (Sequential Perceptron)')
plt.legend()
plt.xlim(-5, 4)
plt.ylim(-3, 6)
plt.grid()
plt.show()
```



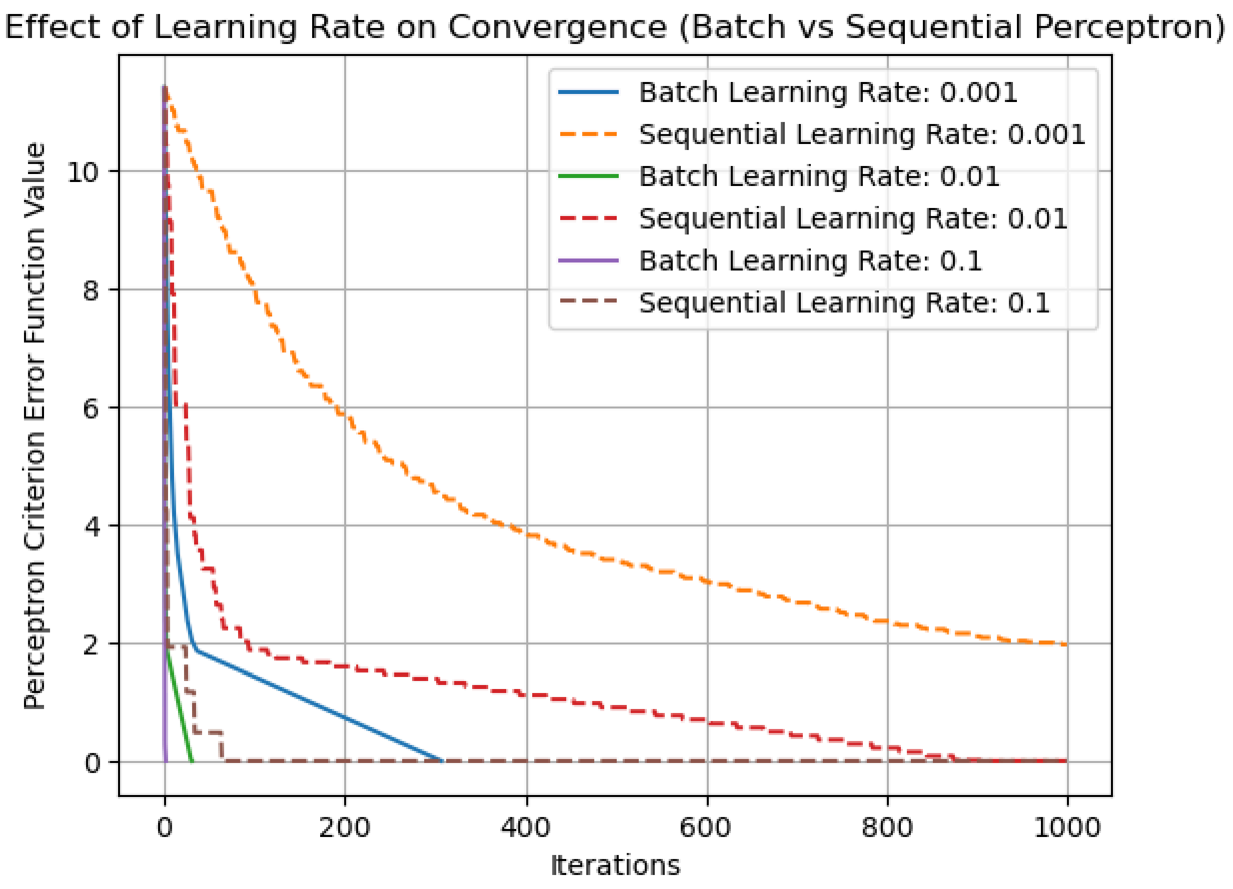
(5) Use the test set in a.(1) to test the trained model in a. (4) and calculate the accuracy (error rate) of the trained model.

In [108…
```
# Test the Sequential Perceptron model
y_pred_seq = perceptron_predict(np.hstack((np.ones((x_test.shape[0], 1)), x_test)), w_seq)
accuracy_seq = np.sum(y_pred_seq == y_test) / y_test.shape[0]
print(f'Sequential Perceptron Test Accuracy: {accuracy_seq * 100:.2f}%')
```

```
Sequential Perceptron Test Accuracy: 100.00%
```

(6) Show how you select learning rate during the training process of a.(2) or a.(4) and demonstrate how the choice of the learning rate is affecting the convergence of the training process.

In [108…
```
learning_rates = [0.001, 0.01, 0.1]  # learning rates to test

for lr in learning_rates:

    # Batch Perceptron training
    w, errors_batch = batch_perceptron_training(x_train, y_train, learning_rate=lr)
    # print(errors_batch)
    plt.plot(errors_batch, label=f'Batch Learning Rate: {lr}')

    # Sequential Perceptron training
    w_seq, weights_history_seq, errors_seq = sequential_perceptron(x_train, y_train, learning_r
    # print(errors_seq)
    plt.plot(errors_seq, linestyle='--', label=f'Sequential Learning Rate: {lr}')

plt.xlabel('Iterations')
plt.ylabel('Perceptron Criterion Error Function Value')
plt.title('Effect of Learning Rate on Convergence (Batch vs Sequential Perceptron)')
plt.legend()
plt.grid()
plt.show()
```



To select an appropriate learning rate, I experimented with three commonly used values: 0.001, 0.01, and 0.1. I plotted the error function values for both the batch and sequential Perceptron methods to visualize their convergence behavior. For the batch Perceptron, a learning rate of 0.1 demonstrates better performance, and for the sequential Perceptron, a learning rate of 0.1 shows superior results.

A small learning rate results in slow convergence, requiring more iterations to minimize error. A moderate learning rate generally provides a good balance convergence speed and stability. A large learning rate can lead to instability, with the error oscillating or even diverging. So we can go through several commonly used learning rates to find a suitable value.

Problem 2 (Use Perceptron model in sklearn to do classification)

Part a:
Use the make_classification method in sklearn to create a binary linearly separable data set with two (2) features and 100 data points. Visualize the data set.
Split the dataset into to training and test data sets using the train_test_split method in sklearn.
Use the training set to train the Perceptron model in sklearn then use the trained model to make prediction on the test set.
Calculate the accuracy of the classification.
Visualize the decision boundary of the trained Perceptron model on top of the test data points.

In [108…

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
```

In [108…

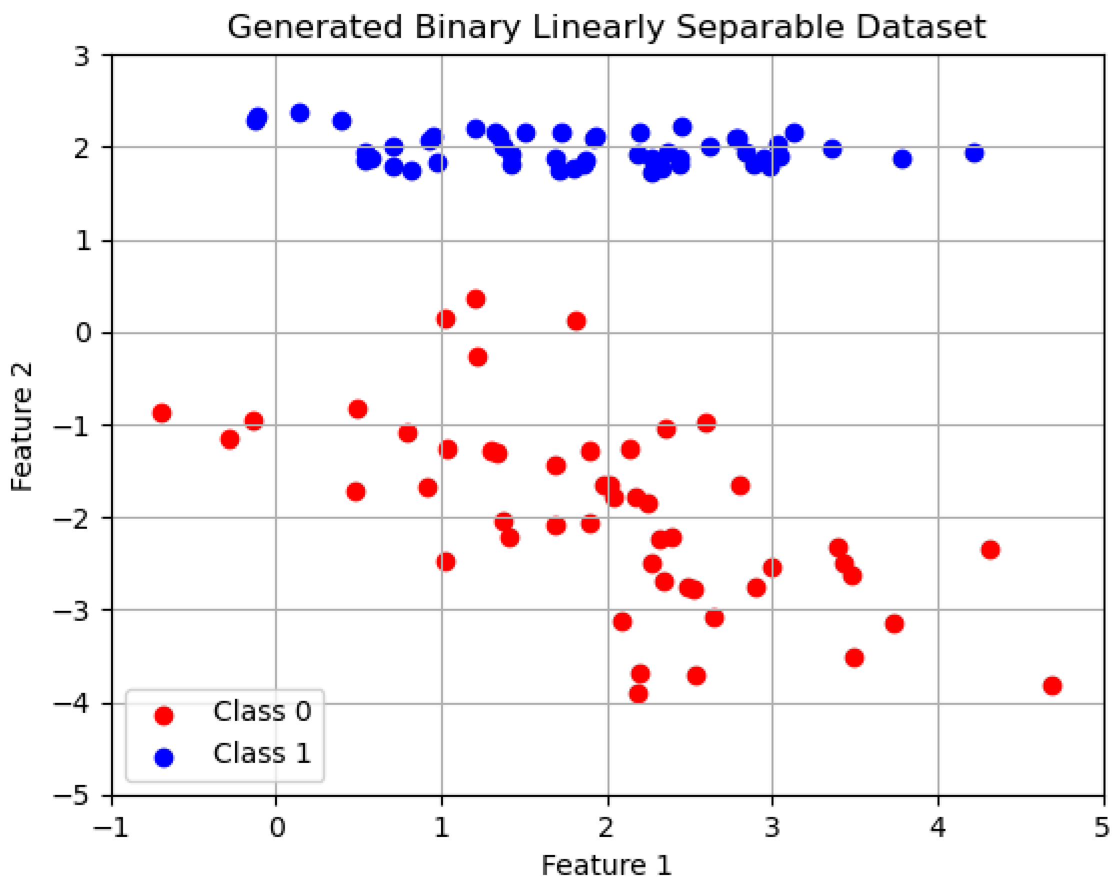
```
# Generate a linearly separable dataset by using make_classification
X, y = make_classification(n_samples=100, # number of samples
                           n_features=2, # number of features
                           n_informative=2, # number of informative features
```

```
                              n_redundant=0, # number of redundant features
                              n_clusters_per_class=1, # number of clusters per class
                              class_sep=2, # separation between classes
                              random_state=0)
print('Generated dataset shape:', X.shape, y.shape)
```

```
Generated dataset shape: (100, 2) (100,)
```

In [108…

```
# Visualize the data set
plt.scatter(X[y == 0, 0], X[y == 0, 1], color='red', label='Class 0')
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='blue', label='Class 1')
plt.title('Generated Binary Linearly Separable Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.xlim(-1, 5)
plt.ylim(-5, 3)
plt.grid()
plt.show()
```



In [108…

```
# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
```

In [108…

```
# Train the Perceptron model in sklearn
model = Perceptron(random_state=0)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

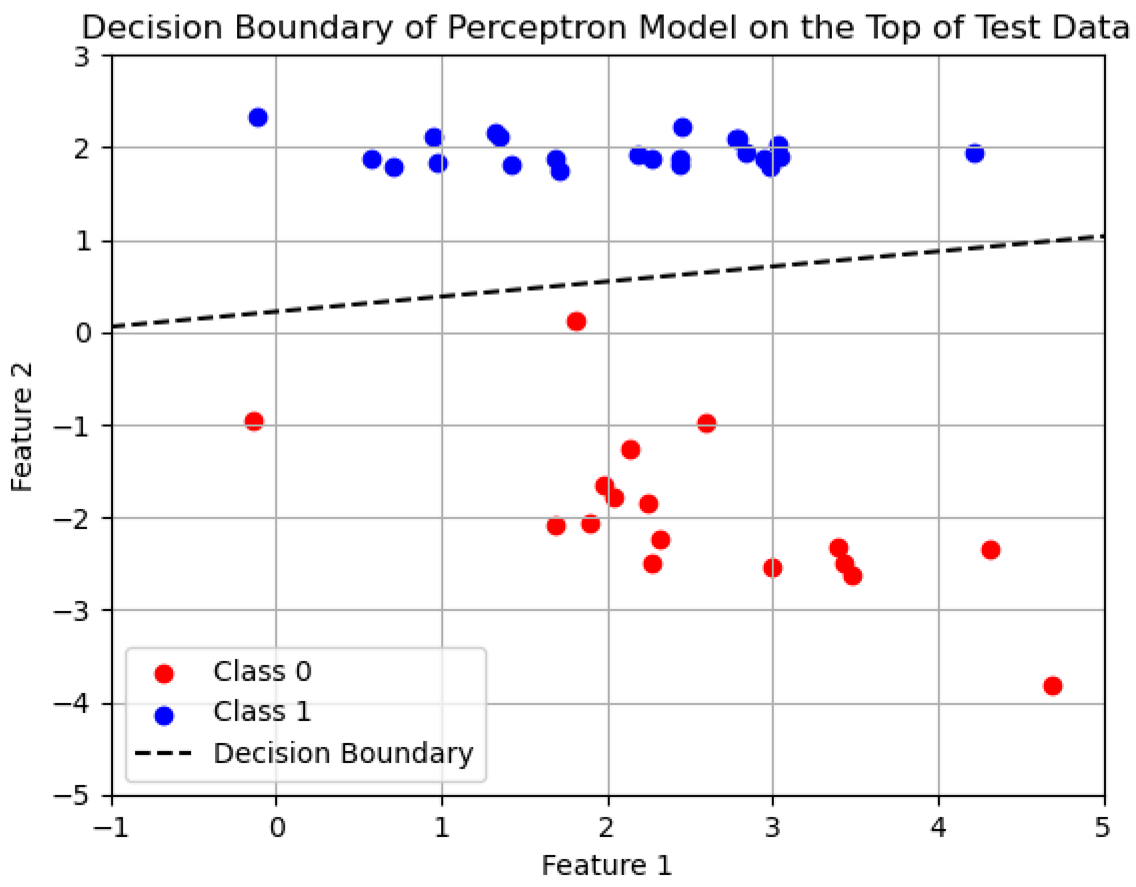
In [108…

```
# Calculate the accuracy of the classification
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Perceptron Model: {accuracy * 100:.2f}%')
```

```
Accuracy of Perceptron Model: 100.00%
```

In [108…

```
# Visualize the decision boundary of the trained Perceptron model on top of the test data point
def plot_decision_boundary(model, X, y):
    # Extract model coefficients
    w = model.coef_[0]
    b = model.intercept_[0]

    # Plot the data points
    plt.scatter(X[y == 0, 0], X[y == 0, 1], color='red', label='Class 0')
    plt.scatter(X[y == 1, 0], X[y == 1, 1], color='blue', label='Class 1')

    # Plot the decision boundary
    x_boundary = np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1, 100)
    y_boundary = -(w[0] * x_boundary + b) / w[1]
    plt.plot(x_boundary, y_boundary, 'k--', label='Decision Boundary')

    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Decision Boundary of Perceptron Model on the Top of Test Data')
    plt.legend()
    plt.xlim(-1, 5)
    plt.ylim(-5, 3)
    plt.grid()
    plt.show()

plot_decision_boundary(model, X_test, y_test)
```



Part b:

Use the make_classification method in sklearn to create a binary non linearly separable data set with two (2) features and 100 data points. The data set should have class imbalance. Visualize the data set.

Split the dataset into to training and test data sets using the train_test_split method in sklearn.

Use the training set to train the Perceptron model in sklearn then use the trained model to make prediction on the test set.

Use sklearn.metrics to show the following on the classification results:

> Accuracy, Precision, Recall, F1 score
>
> The classification_report and the confusion_matrix

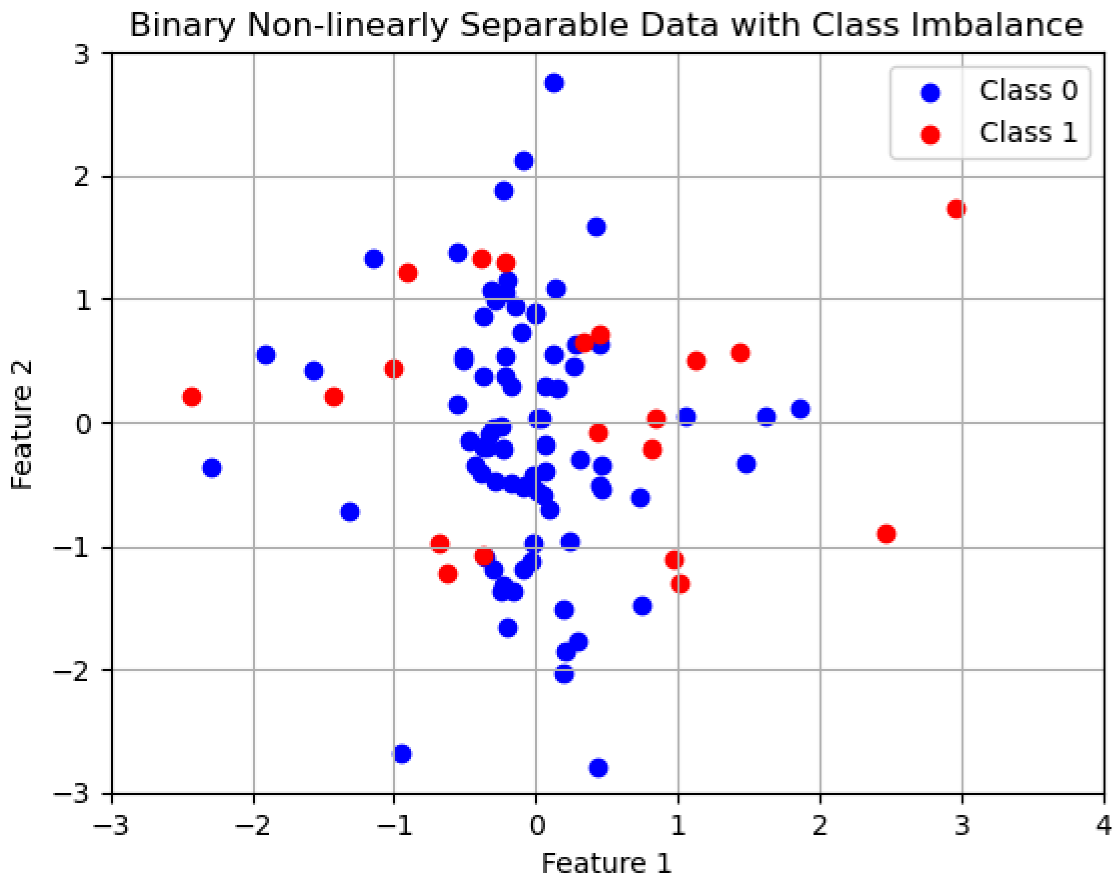
Visualize the decision boundary of the trained Perceptron model on top of the test data points.

```
In [109…  import numpy as np
          import matplotlib.pyplot as plt
          from sklearn.datasets import make_classification
          from sklearn.model_selection import train_test_split
          from sklearn.linear_model import Perceptron
          from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
          from sklearn.metrics import classification_report, confusion_matrix
          import matplotlib.colors as mcolors
```

```
In [109…  # Generate a binary non linearly separable dataset by using make_classification
          X, y = make_classification(n_samples=100, # number of samples
                                     n_features=2, # number of features
                                     n_informative=2, # number of informative features
                                     n_redundant=0, # number of redundant features
                                     n_clusters_per_class=2, # number of clusters per class
                                     class_sep=0.1, # separation between classes
                                     weights=[0.8, 0.2], # create imbalanced classes (80% class 0, 20% cl
                                     random_state=0)
          print('Generated dataset shape:', X.shape, y.shape)
```

```
          Generated dataset shape: (100, 2) (100,)
```

```
In [109…  # Visualize the dataset
          plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='Class 0')
          plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='Class 1')
          plt.title('Binary Non-linearly Separable Data with Class Imbalance')
          plt.xlabel('Feature 1')
          plt.ylabel('Feature 2')
          plt.legend()
          plt.xlim(-3, 4)
          plt.ylim(-3, 3)
          plt.grid()
          plt.show()
```



```
In [109…  # Split the dataset into training set and test set
          x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
In [109…  # Train Perceptron model
          perceptron = Perceptron(random_state=0)
```

```
perceptron.fit(x_train, y_train)

# Make predictions on the test set
y_pred = perceptron.predict(x_test)
```

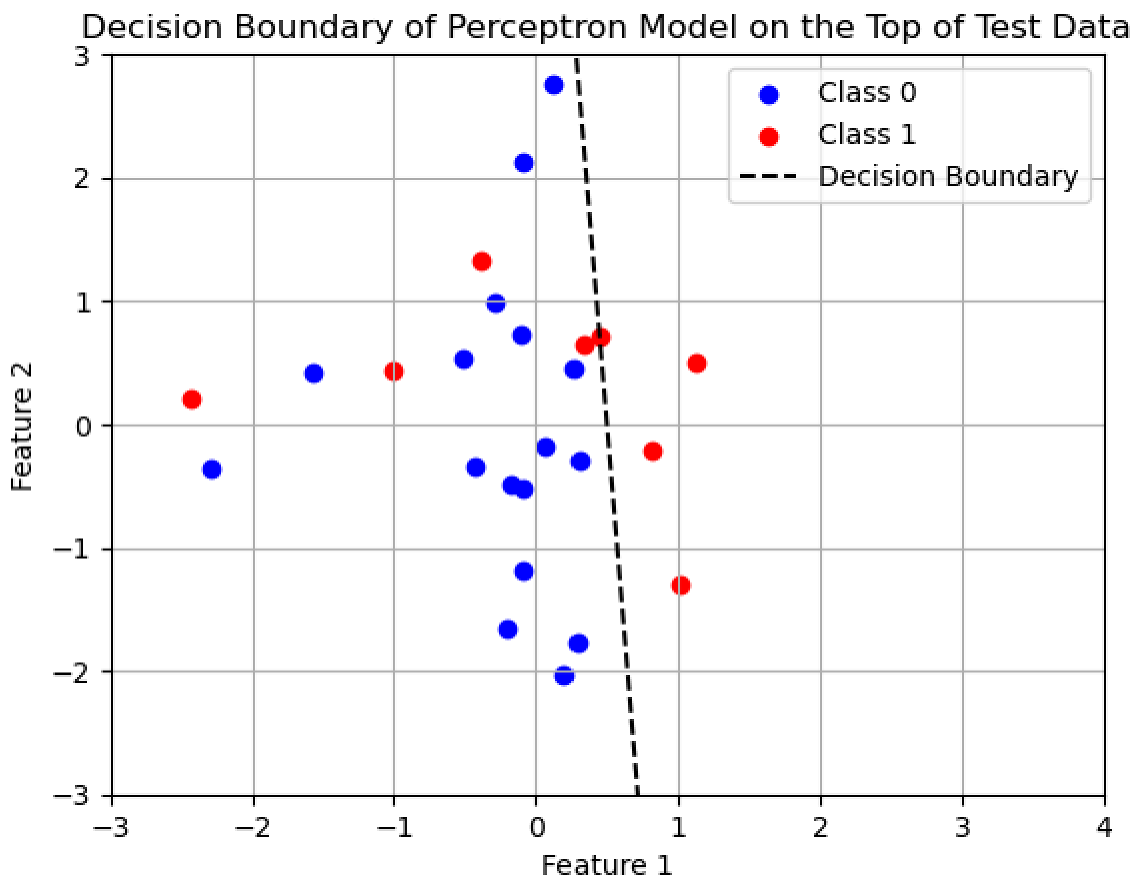
In [109…

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print('\nClassification Report:\n', report)
print('Confusion Matrix:\n', conf_matrix)
```

```
Accuracy: 0.84
Precision: 1.0
Recall: 0.5
F1 Score: 0.6666666666666666

Classification Report:
               precision    recall  f1-score   support

           0       0.81      1.00      0.89        17
           1       1.00      0.50      0.67         8

    accuracy                           0.84        25
   macro avg       0.90      0.75      0.78        25
weighted avg       0.87      0.84      0.82        25

Confusion Matrix:
 [[17  0]
 [ 4  4]]
```

In [109…

```
# Visualize the decision boundary of the trained Perceptron model on top of the test data point
def plot_decision_boundary(model, X, y):
    # Extract model coefficients
    w = model.coef_[0]
    b = model.intercept_[0]

    # Plot the data points
    plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='Class 0')
    plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='Class 1')

    # Plot the decision boundary
    x_boundary = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    y_boundary = -(w[0] * x_boundary + b) / w[1]
    plt.plot(x_boundary, y_boundary, 'k--', label='Decision Boundary')

    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Decision Boundary of Perceptron Model on the Top of Test Data')
    plt.legend()
    plt.xlim(-3, 4)
    plt.ylim(-3, 3)
    plt.grid()
    plt.show()

plot_decision_boundary(perceptron, x_test, y_test)
```


Decision Boundary of Perceptron Model on the Top of Test Data
Class 0
Class 1
Decision Boundary
Feature 1
Feature 2
−3
−2
−1
0
1
2
3
4