MATH-CSCI485:Assignment 4: Perceptron

By: Akshay Aralikatti | 012624106

Part 1: Heuristic Approach

Procedure:

1.Data Loading and Preprocessing:

The CSV is loaded and features are scaled using StandardScaler for improved training behavior.

2.Hyperparameters and Initialization:

The learning rate are varied between 0.05, 0.02, 0.1 and the maximum number of iterations to 100. Weights and bias are initialized using a fixed seed for reproducibility.

3.Training and Boundary Plotting:

- The training loop uses the heuristic update rule.
- After each weight update, the current decision boundary is computed and drawn as a dashed green line over the fixed scatter plot.
- The initial decision boundary is drawn in red before any updates.

4. Training Summary:

After the loop finishes (either due to convergence or reaching the iteration cap), the training accuracy, final weights, bias, and total iterations are printed.

5. Code Snippet:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler




# Function: Compute Decision Boundary Line

def decision_boundary_line(w, b, x_vals):
```

```python
    # Avoid division by zero if w[1] is very small
    if np.abs(w[1]) < 1e-6:
        return np.full_like(x_vals, -b)
    return -(b + w[0] * x_vals) / w[1]


# 1. Load and Preprocess the Dataset
# Change the file path below as needed
data = pd.read_csv('/content/drive/MyDrive/485/Perceptron/data.csv')
# Assume the first two columns are features and the third column is the label
X = data.iloc[:, :2].values
y = data.iloc[:, 2].values


# Feature Scaling for better convergence
scaler = StandardScaler()
X = scaler.fit_transform(X)



# 2. Set Hyperparameters
learning_rate = 0.1
max_iterations = 65
iterations_count = 0



# 3. Initialize Weights and Bias
np.random.seed(0)
```

```
weights = np.random.rand(2)

bias = np.random.rand(1)[0]


# 4. Train the Perceptron and Plot Decision Boundaries


# Create a figure for the training visualization

plt.figure(figsize=(8, 6))


# Plot the fixed 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='orange', label='Class 1')


# Define x-values to compute decision boundaries

x_vals = np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1, 100)


# Plot the initial decision boundary (red)

y_initial = decision_boundary_line(weights, bias, x_vals)

plt.plot(x_vals, y_initial, color='red', linewidth=2, label='Initial Boundary')


# Training loop

converged = False

while iterations_count < max_iterations and not converged:

    error_count = 0

    for i in range(len(X)):
```

```
        if iterations_count >= max_iterations:
            break
        # Compute linear output and make prediction using step function
        z = np.dot(X[i], weights) + bias
        prediction = 1 if z >= 0 else 0
        error = y[i] - prediction

        # Count errors if any
        if error != 0:
            error_count += 1

        # Update weights and bias (heuristic update rule)
        weights += learning_rate * error * X[i]
        bias += learning_rate * error
        iterations_count += 1

        # Plot the decision boundary after each update (dashed green)
        y_boundary = decision_boundary_line(weights, bias, x_vals)
        plt.plot(x_vals, y_boundary, linestyle='--', color='green',
alpha=0.5)

    # If no errors in the entire pass, we consider the model converged
    if error_count == 0:
        converged = True
```

```
# 5. Compute Training Accuracy

predictions = np.array([1 if np.dot(x, weights) + bias >= 0 else 0 for x
in X])

accuracy = np.mean(predictions == y) * 100



# Print a summary of the training results

print("Training with learning rate: {}".format(learning_rate))

print("Iterations to converge (or maximum reached):
{}".format(iterations_count))

print("Final weights: {}".format(weights))

print("Final bias: {}".format(bias))

print("Accuracy: {:.2f}%".format(accuracy))




# 6. Plot the Final Decision Boundary in a Separate Figure

y_final = decision_boundary_line(weights, bias, x_vals)

plt.plot(x_vals, y_final, color='black', linewidth=2, label='Final
Boundary')

plt.xlabel('x1 (scaled)')

plt.ylabel('x2 (scaled)')

plt.title('Perceptron Decision Boundary')

plt.xlim(-2, 2)

plt.ylim(-2, 2)

plt.legend()

plt.show()
```

6. Analysis:

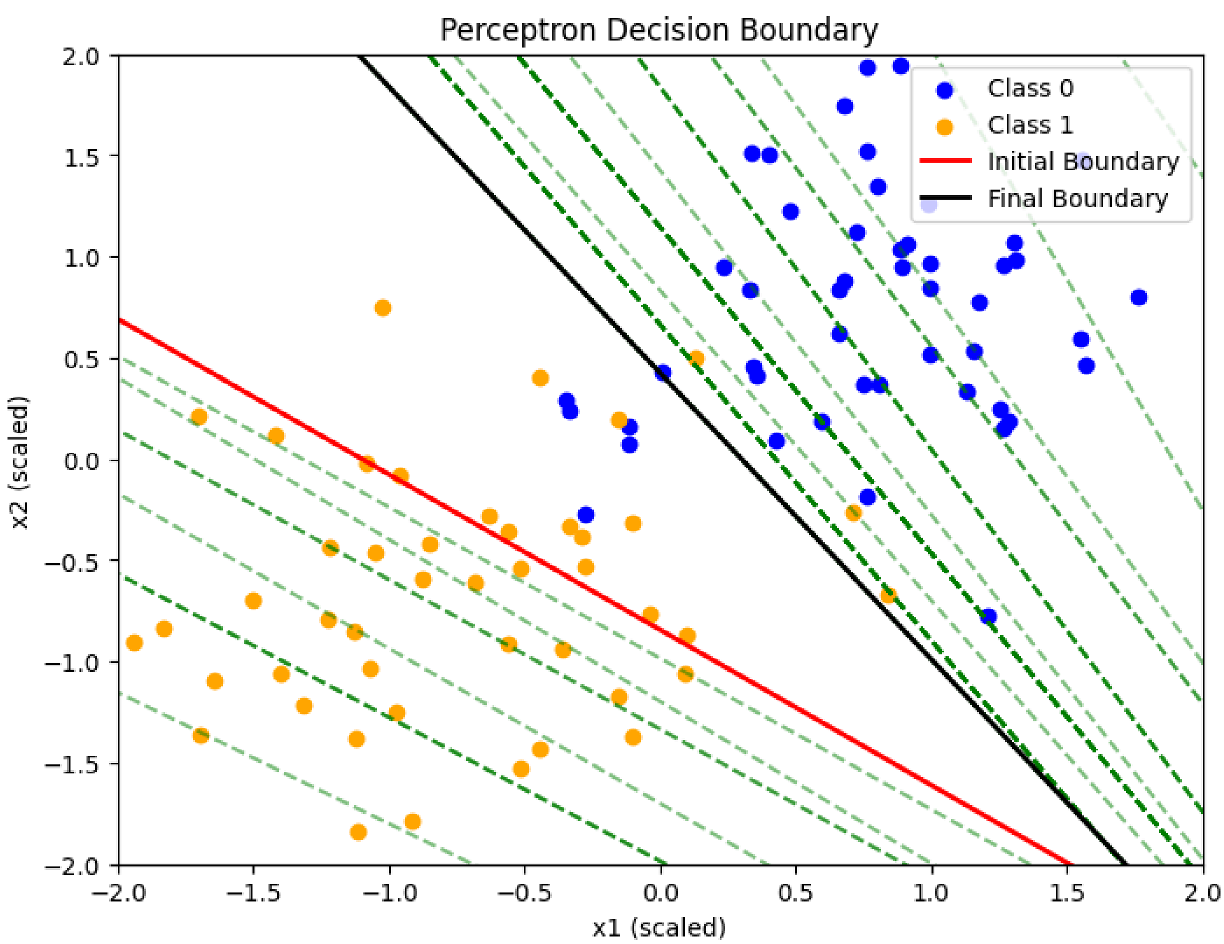
Using the heuristic perceptron, the model updates weights and bias via a discrete step function. Initially, the decision boundary is randomly set and plotted in red. With each misclassified sample, the boundary gradually adjusts as weight updates are applied and intermediate boundaries are shown in dashed green. This approach converges rapidly on linearly separable data but remains sensitive to the learning rate and update limit. The final boundary, drawn in black, clearly separates the two classes. Overall, the heuristic method is simple and effective for binary classification, although it can be unstable with noisy or non-separable data and shows promise.

7. Final Visualization:

A new figure plots the data with the final decision boundary (in black) for clarity.

1st trial:

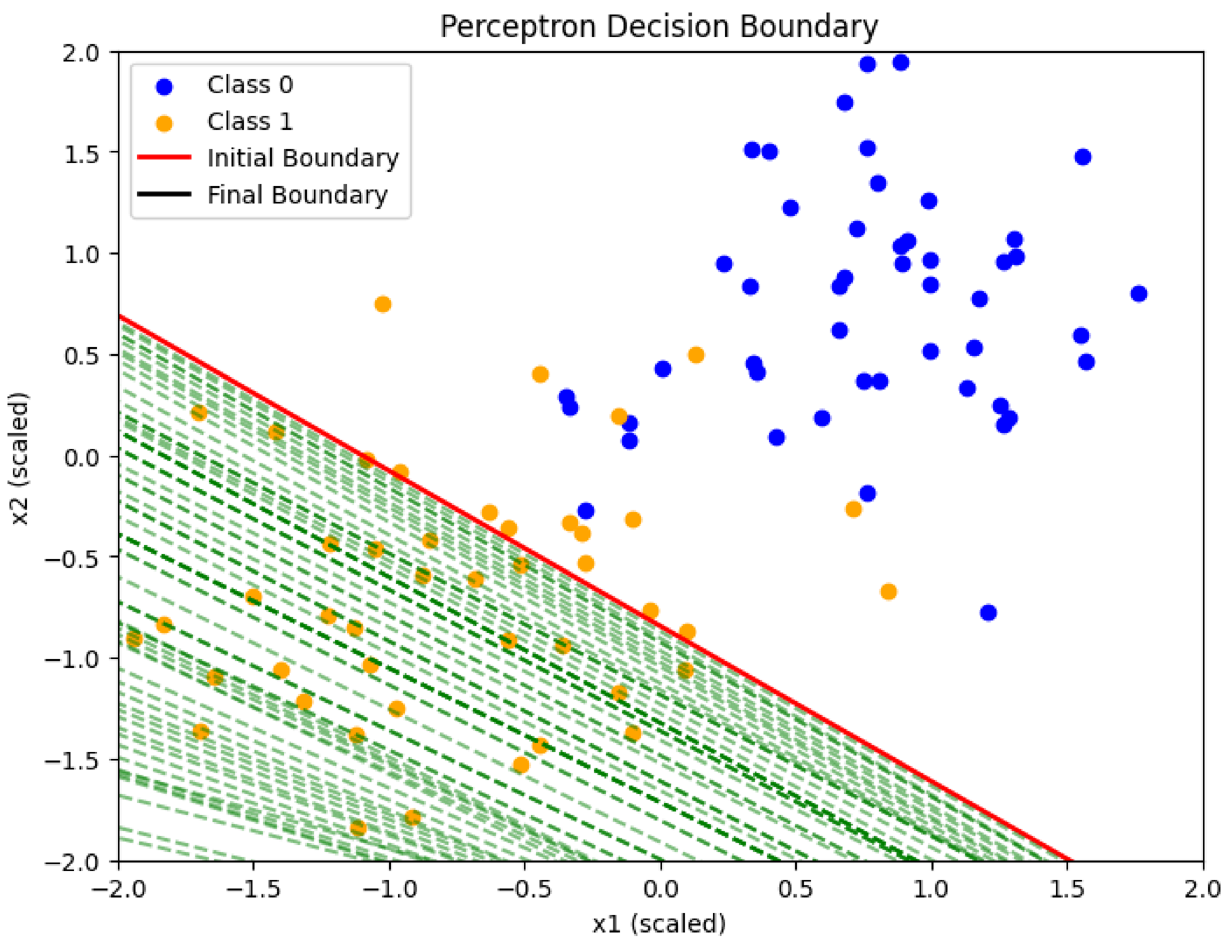
- Training with learning rate: **0.05**
- Iterations to converge (or maximum reached): **100**
- Final weights: **[-0.34208439 -0.24193413]**
- Final bias: **0.10276337607164386**
- Accuracy: **91.92%**



2nd trial:

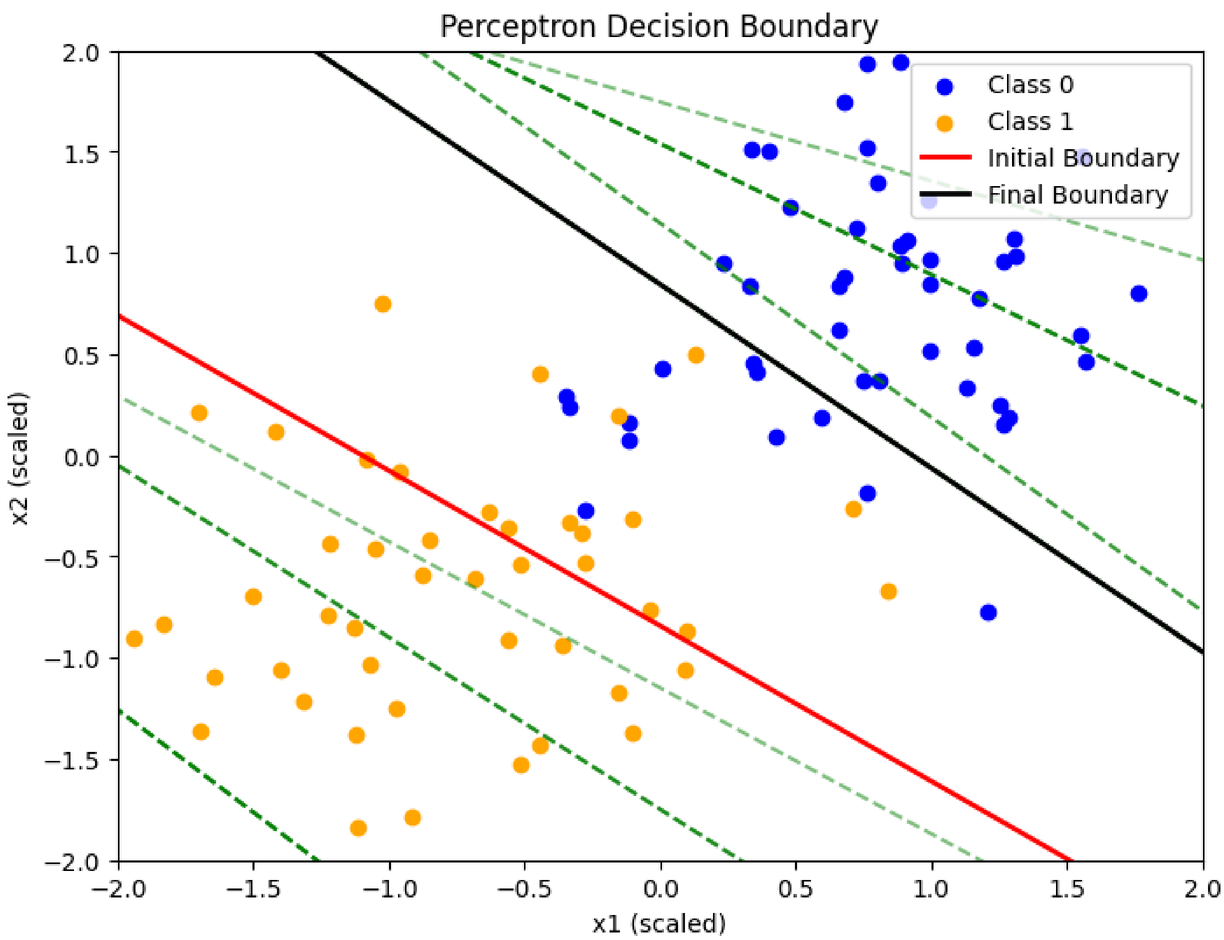
- Training with learning rate: **0.01**

- Iterations to converge (or maximum reached): **150**
- Final weights: **[-0.13116605 0.00210726]**
- Final bias: **0.33276337607164364**
- Accuracy: **49.49%**



3rd trial

- Training with learning rate: **0.1**
- Iterations to converge (or maximum reached): **65**
- Final weights: **[-0.32543339 -0.35854235]**
- Final bias: **0.30276337607164405**
- Accuracy: **87.88%**



Part 2: Gradient Descent Approach

1. Data Loading & Preprocessing:
The code reads in data.csv (assumed to have columns for two features and a binary label). StandardScaler is applied to center and scale the features. (You may choose MinMaxScaler if you prefer to confine the feature values to [0,1].)

2. Setting Hyperparameters:
You can adjust the learning rate and number of epochs. In this example, a learning rate of 0.1 and 50 epochs are used. The log loss is recorded per epoch for later plotting.

3. Initialization:
Weights and biases are randomly initialized with a fixed seed to ensure reproducibility.

4. Plotting the Decision Boundaries During Training:

- The fixed scatter plot of the data points is drawn first.
- The initial decision boundary (based on the random weights) is plotted in red.
- During each epoch, after processing the full dataset, the current decision boundary is computed and plotted as a dashed green line.
- Finally, after training, the final decision boundary is plotted in black.

5. Log Loss Graph:

After training, a separate plot shows the evolution of the log loss over epochs. The x-axis shows epochs (with markers at every 10 epochs) and the y-axis displays the log loss error.

6. Code snippets:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Helper Functions
def sigmoid(z):
   """Compute the sigmoid function."""
   return 1.0 / (1.0 + np.exp(-z))

def log_loss(y_true, y_pred):
   """Compute binary cross-entropy loss."""
   eps = 1e-15  # avoid log(0)
   y_pred = np.clip(y_pred, eps, 1 - eps)
   return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 -
y_pred))

def decision_boundary_line(w, b, x_vals):
   """
   Given weights w and bias b, compute the corresponding x2 values for the
decision boundary.
   For logistic regression, the decision boundary is when w·x + b = 0.
   """
   # If w[1] is nearly zero, return a constant line.
   if np.abs(w[1]) < 1e-6:
       return np.full_like(x_vals, -b)
   return -(b + w[0] * x_vals) / w[1]


# 1. Load and Preprocess the Dataset
# Assume data.csv contains three columns: x1, x2, label
# data = pd.read_csv('data.csv')
X = data.iloc[:, :2].values    # features: x1 and x2
y = data.iloc[:, 2].values     # label: 0 or 1

# Optional: Scale features for better convergence.
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
# 2. Set Hyperparameters
learning_rate = 0.05
epochs = 60
losses = []


# 3. Initialize Weights and Bias
np.random.seed(0)
weights = np.random.rand(2)  # Two features: one weight per input
dimension
bias = np.random.rand(1)[0]


# 4. Plot Data and Training Evolution of Decision Boundaries
plt.figure(figsize=(8, 6))

# Plot fixed 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='orange', label='Class 1')

# x-values for computing decision boundaries
# (Setting the range based on the data spread)
x_vals = np.linspace(X[:,0].min()-1, X[:,0].max()+1, 100)

# Plot the initial decision boundary in red (before any updates)
y_initial = decision_boundary_line(weights, bias, x_vals)
plt.plot(x_vals, y_initial, color='red', linewidth=2, label='Initial
Boundary')

# 5. Train the Perceptron with Gradient Descent
for epoch in range(epochs):
   # For each epoch, loop over each sample (stochastic gradient descent
update)
   for i in range(len(X)):
       # Compute the linear combination and the sigmoid activation
       z = np.dot(X[i], weights) + bias
       y_pred = sigmoid(z)
       error = y[i] - y_pred

       # Update weights and bias using the gradient descent rule
       weights += learning_rate * error * X[i]
       bias += learning_rate * error
```

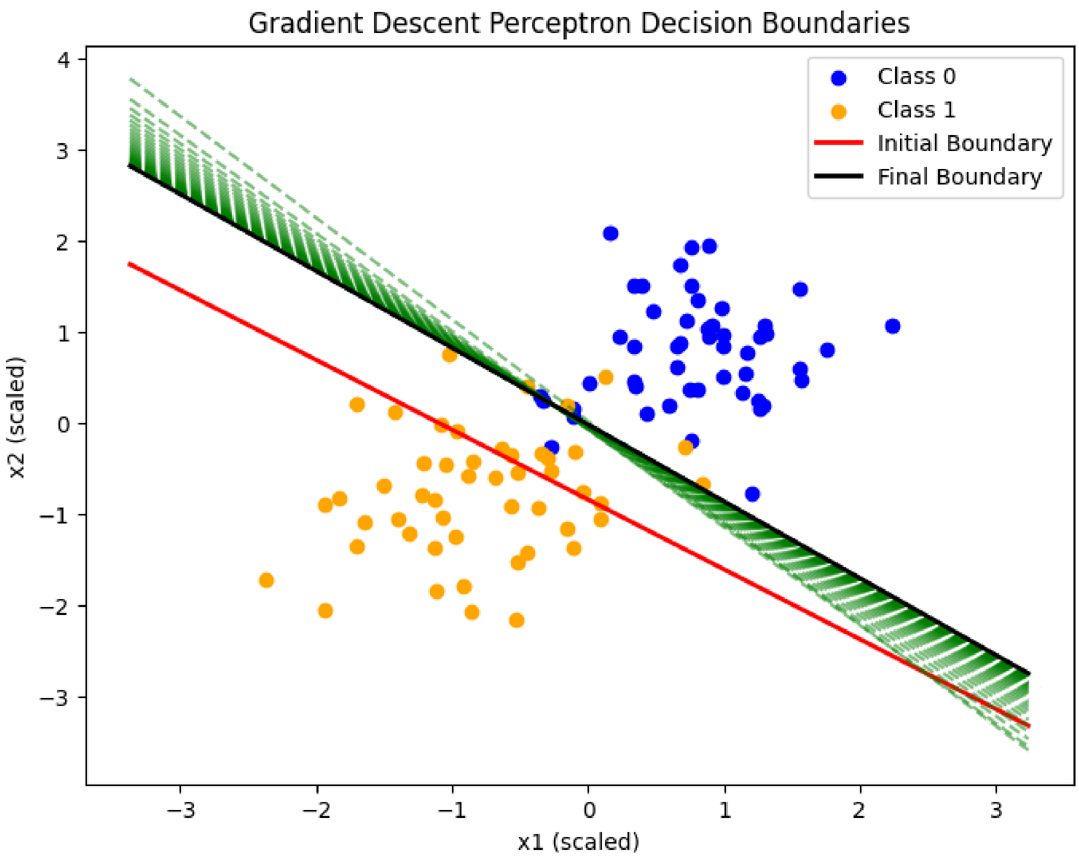
```
    # Compute predictions over the whole dataset and log loss for this
epoch
    preds = sigmoid(np.dot(X, weights) + bias)
    epoch_loss = log_loss(y, preds)
    losses.append(epoch_loss)

    # After each epoch, plot the current decision boundary (dashed green)
    y_boundary = decision_boundary_line(weights, bias, x_vals)
    plt.plot(x_vals, y_boundary, linestyle='--', color='green', alpha=0.5)

    # Optionally, print the log loss every 10 epochs
    if (epoch+1) % 10 == 0:
        print(f"Epoch {epoch+1}, Log Loss: {epoch_loss:.4f}")

# Plot the final decision boundary in black after training
y_final = decision_boundary_line(weights, bias, x_vals)
plt.plot(x_vals, y_final, color='black', linewidth=2, label='Final
Boundary')

plt.xlabel('x1 (scaled)')
plt.ylabel('x2 (scaled)')
plt.title('Gradient Descent Perceptron Decision Boundaries')
plt.legend()
plt.show()


# 6. Plot the Log Loss vs. Epoch Graph
plt.figure(figsize=(8,6))
plt.plot(range(1, epochs+1), losses, marker='o')
plt.xlabel('Epoch')
plt.ylabel('Log Loss')
plt.title('Log Loss vs. Epochs')
plt.xticks(range(1, epochs+1, 10))
plt.grid(True)
plt.show()
```

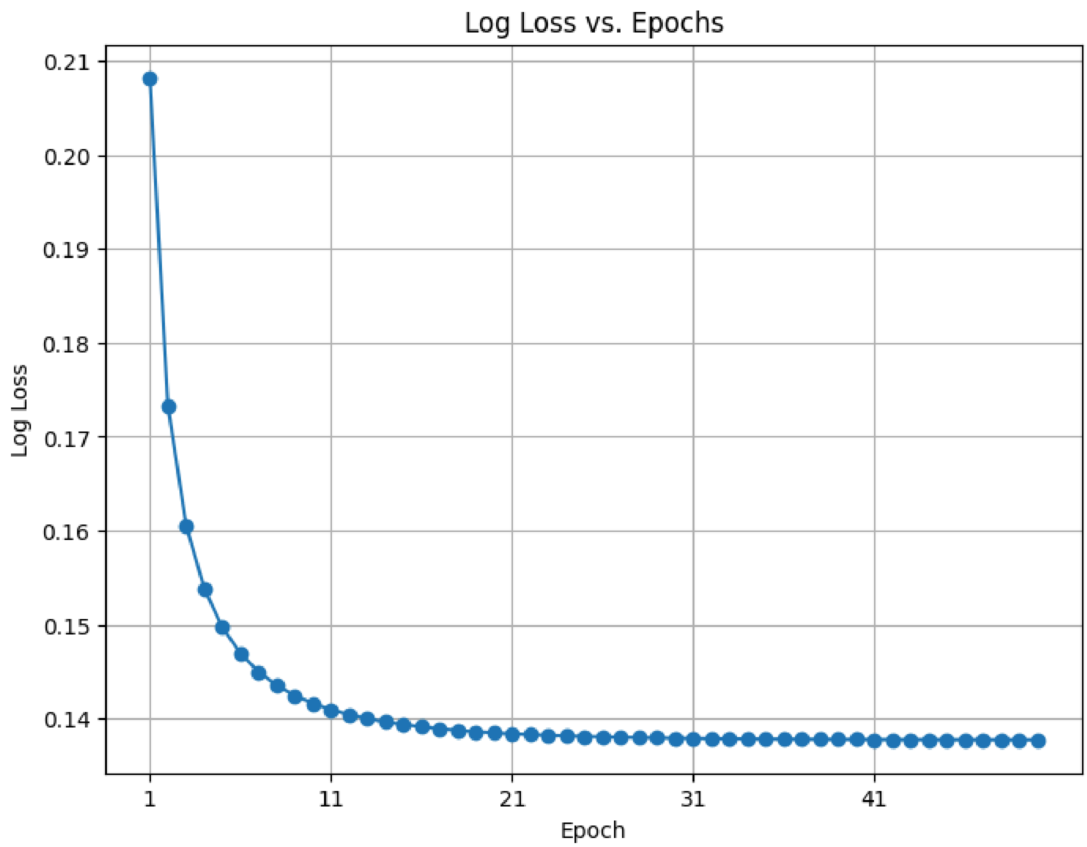
7. Final Visualization:
Throughout training, the code prints out the log loss every 10 epochs for monitoring.

8. Analysis:

Using gradient descent, the perceptron employs a sigmoid activation for continuous output, enabling smoother adjustments of the decision boundary. Initially, the boundary is set based on random weights and bias and displayed in red. As training proceeds through each epoch, weight updates gradually minimize the log loss, while intermediate dashed green boundaries illustrate incremental improvements. Every ten epochs, the error is recorded and plotted to visualize convergence trends. The final decision boundary, shown in black, effectively divides the data into two classes. Overall, this method yields smooth convergence and improved stability, although it requires careful tuning of the learning rate.

1st trial:
Learning rate: **0.1**
Epochs: **50**
Epoch **10**, Log Loss: **0.1416**
Epoch **20**, Log Loss: **0.1385**
Epoch **30**, Log Loss: **0.1379**
Epoch **40**, Log Loss: **0.1378**
Epoch **50**, Log Loss: **0.1377**





2st trial:

Learning rate: **0.01**

Epochs: **100**

Epoch **10**, Log Loss: **0.2165**

Epoch **20**, Log Loss: **0.1744**

Epoch **30**, Log Loss: **0.1601**

Epoch **40**, Log Loss: **0.1530**

Epoch **50**, Log Loss: **0.1487**

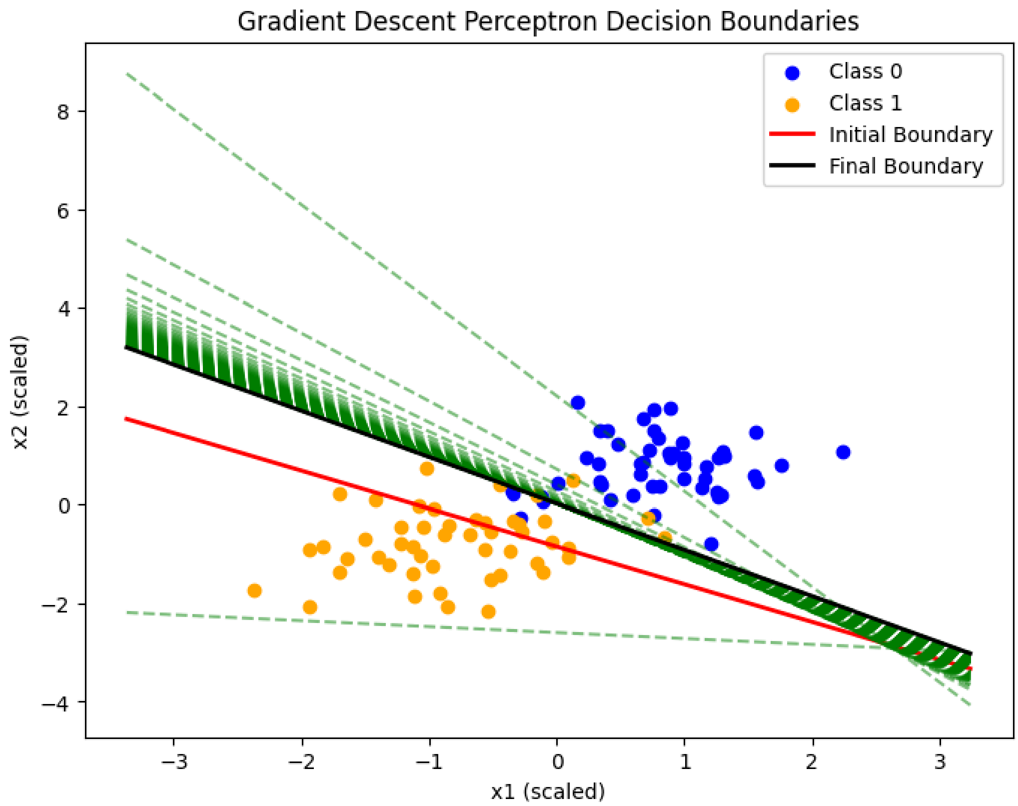
Epoch **60**, Log Loss: **0.1459**

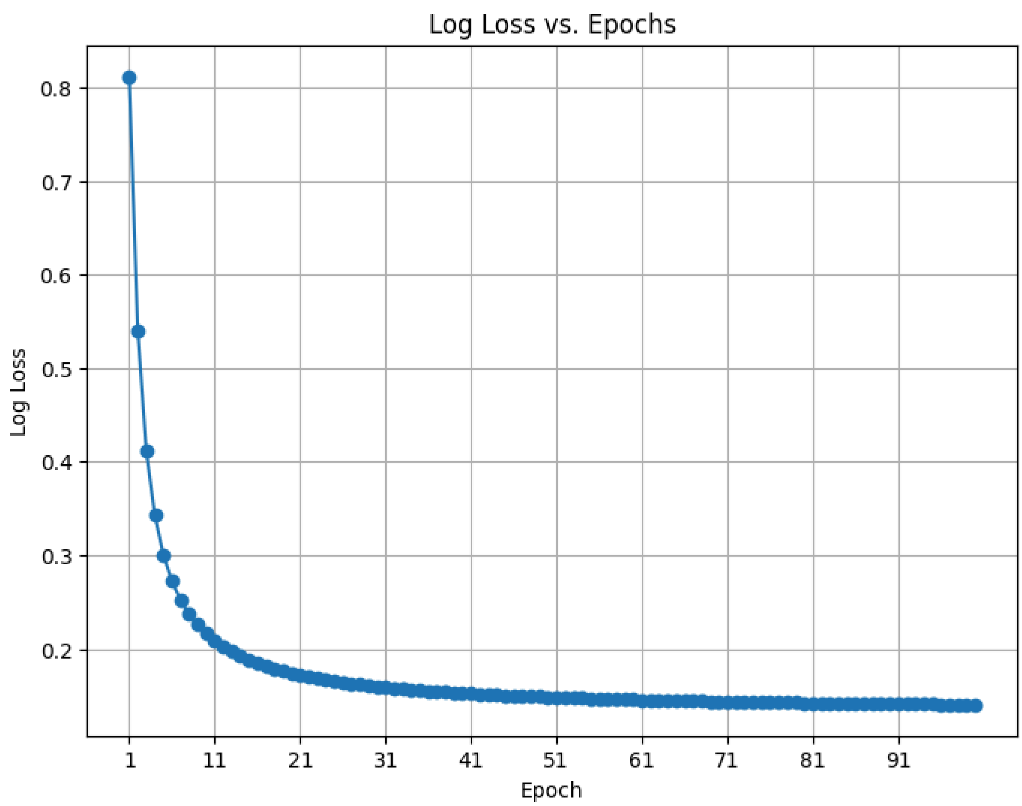
Epoch **70**, Log Loss: **0.1440**

Epoch **80**, Log Loss: **0.1426**

Epoch **90**, Log Loss: **0.1415**

Epoch **100**, Log Loss: **0.1407**





3st trial:

Learning rate: **0.05**

Epochs: **60**

Epoch **10**, Log Loss: **0.1488**
Epoch **20**, Log Loss: **0.1408**
Epoch **30**, Log Loss: **0.1386**
Epoch **40**, Log Loss: **0.1378**
Epoch **50**, Log Loss: **0.1374**
Epoch **60**, Log Loss: **0.1373**