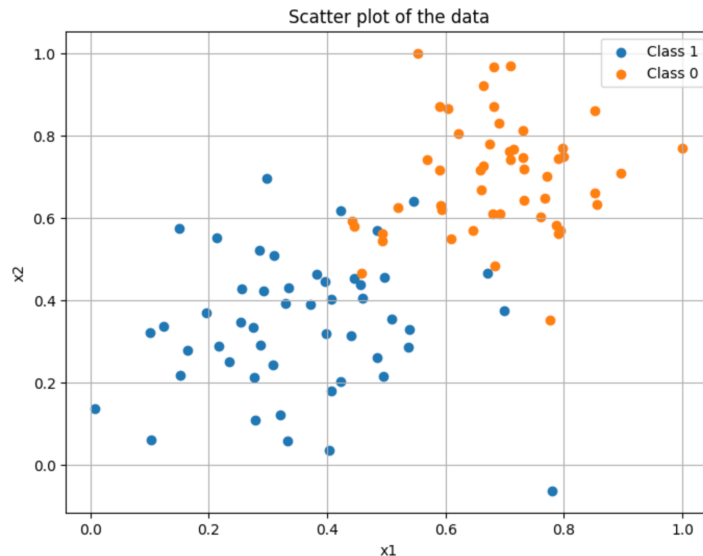


# Assignment - 5 Report

Github link - <https://github.com/Baddala-Govardhan/CSCI580/tree/main>

## Part - 1

### 1. Load Data and Plot



Loaded the dataset from data.csv which contains two input features ( $x_1$ ,  $x_2$ ) and a binary label (label). The dataset was then visualized using a scatter plot.

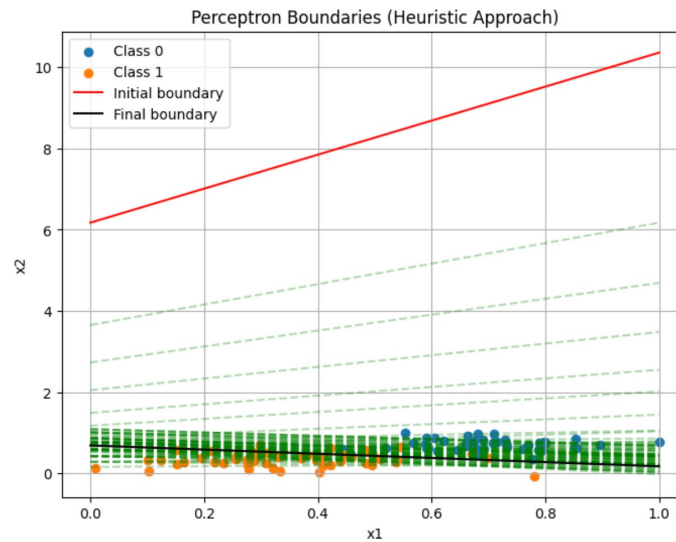
- Blue dots represent data points belonging to Class 1
- Orange dots represent data points from Class 0
- The data is well separated which visually suggests it may be linearly separable

### 2. Implement Perceptron (Heuristic Rule)

Implemented the perceptron using a simple heuristic method. The model starts with random weights and updates them whenever it misclassifies a point. After 94 updates the weights improved, helping the perceptron learn a better decision boundary.

- Number of updates: 94
- Initial weights: [ 1.13870973 0.77271514 -0.18449577]
- Final weights: [ 0.53870973 -0.40369186 -0.78443577]

### 3. Plot the initial separation



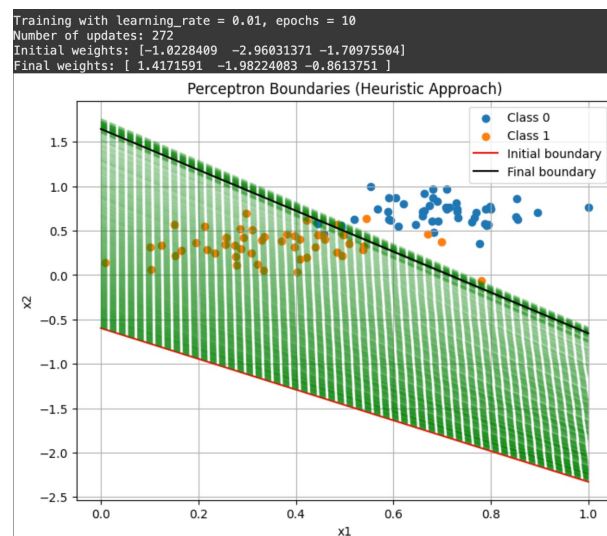
In this step, I plotted how the perceptron updated its decision boundary over time.

- The red line shows the initial boundary before training.
- The green dashed lines represent the boundary after each weight update.
- The black line is the final decision boundary after training completes.

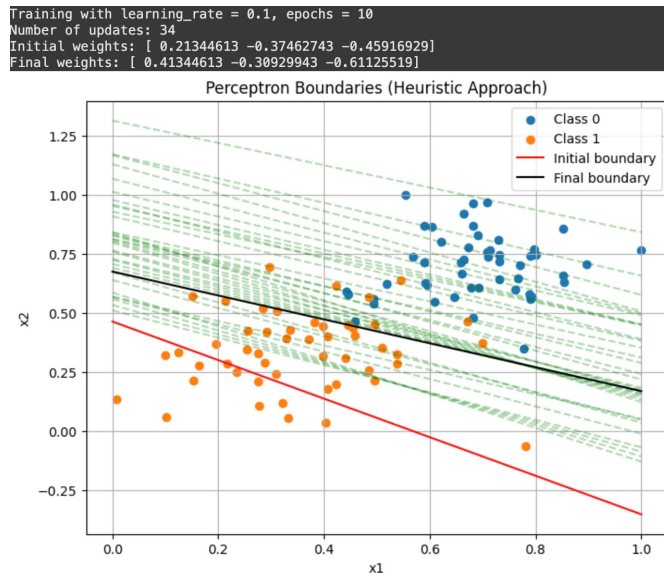
This visualization helps us understand how the model improves its classification by adjusting the boundary gradually based on misclassified points

### 4. Learning Rates

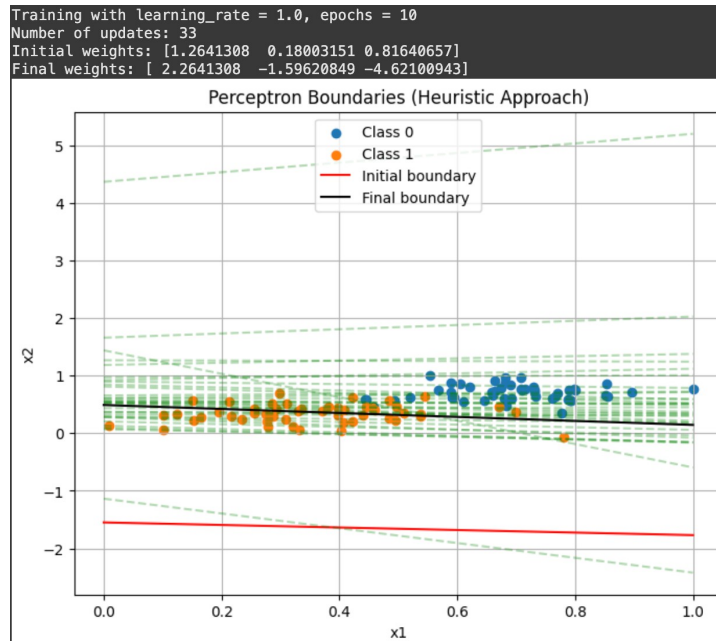
Learning rate - 0.01



Learning rate - 0.1



Learning rate - 1.0



Experimented with different learning rates: 0.01, 0.1, and 1.0 to observe how they affect the decision boundary updates and convergence.

- Learning rate = 0.01

The model took more updates (272) to converge. The decision boundary changed slowly, and the final boundary appeared very flat, even stretching unusually. This shows that a small learning rate leads to slower learning and can require more updates to converge.

- Learning rate = 0.1

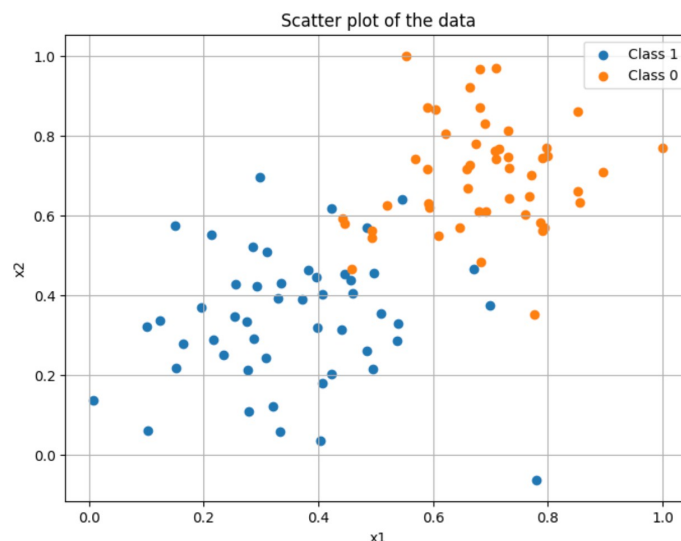
This value provided a balanced result. It needed only (34) updates and showed smoother boundary shifts. The model converged reasonably fast and accurately, making it a reliable default choice.

- Learning rate = 1.0

With a high learning rate, the model updated quickly and used just (33) updates. However, the decision boundary changed sharply between updates, which might lead to instability in some cases. In this experiment, though, it still produced a good final separating line.

## Part -2

### 1. Load Data and Plot



Loaded the dataset from data.csv which contains two input features ( $x_1$ ,  $x_2$ ) and a binary label (label). The dataset was then visualized using a scatter plot.

- Blue dots represent data points belonging to Class 1
- Orange dots represent data points from Class 0
- The data is well separated which visually suggests it may be linearly separable

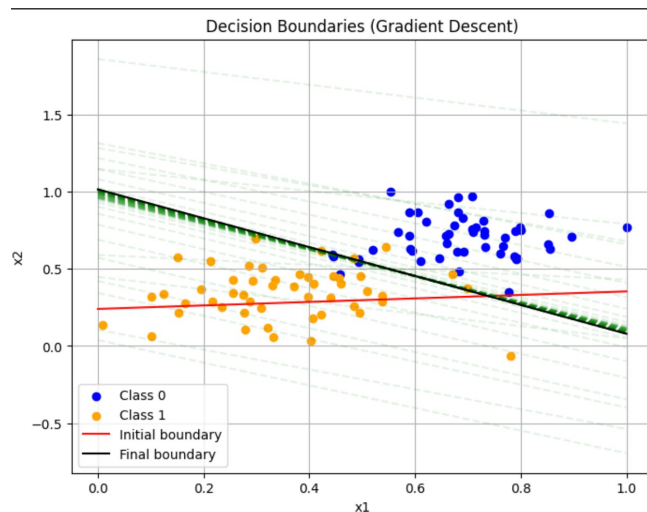
## 2. Implement perceptron using the Gradient Descent

started with random weights and updated them gradually using the error between predicted output ( $\hat{y}$ ) and true labels ( $y$ ). The weights were adjusted in small steps for every epoch to reduce the loss.

After training for 10 epochs:

- Final Weights: [10.13, -9.34, -9.98]
- First 5 Log Loss values: [0.612, 0.547, 0.514, 0.583, 0.988]

## 3. Plot the initial separation

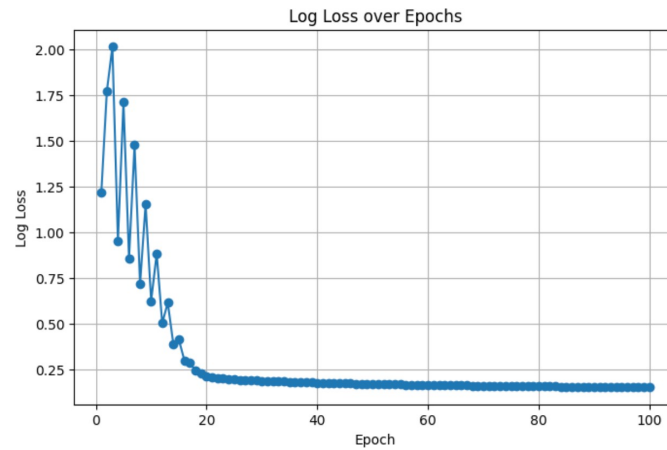


Visualized how the decision boundary evolved during training using Gradient Descent.

- The initial boundary is shown in red, starting far below the data points.
- Each intermediate boundary during the 100 training steps is shown as a dashed green line, representing the gradual updates.
- The final boundary, after training, is shown in black, which closely separates the two classes.

This helps us clearly observe how the model gradually adjusted to separate the two classes better

#### 4. Compute log loss (error)

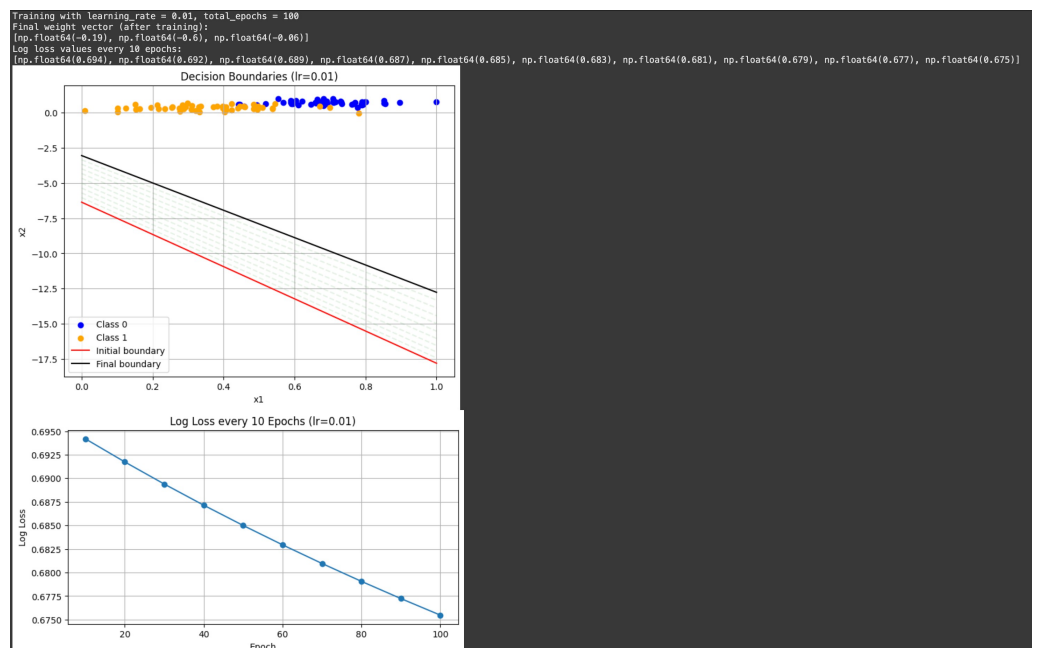


Calculated the log loss (error) during each training epoch to measure how well the model was learning.

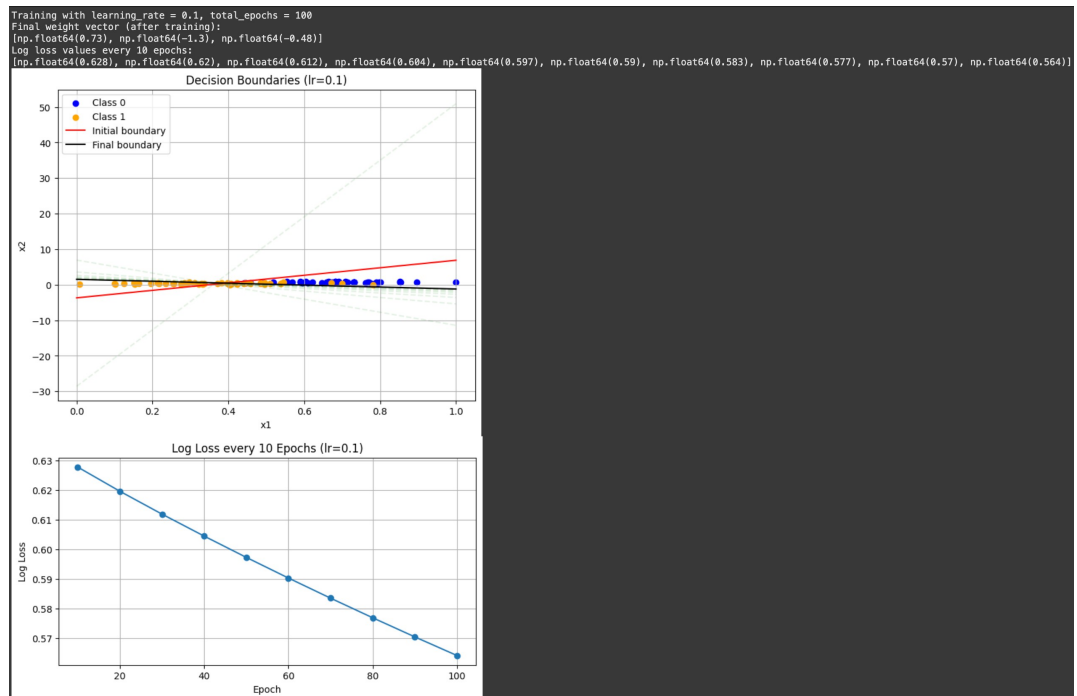
- The loss started off high and fluctuated for the first few epochs as the model adjusted.
- After around 20 epochs, the loss steadily decreased and stabilized, indicating the model had converged.
- This plot provides a clear view of the model's learning curve, confirming that it improved consistently over time.

#### 5. Learning Rate

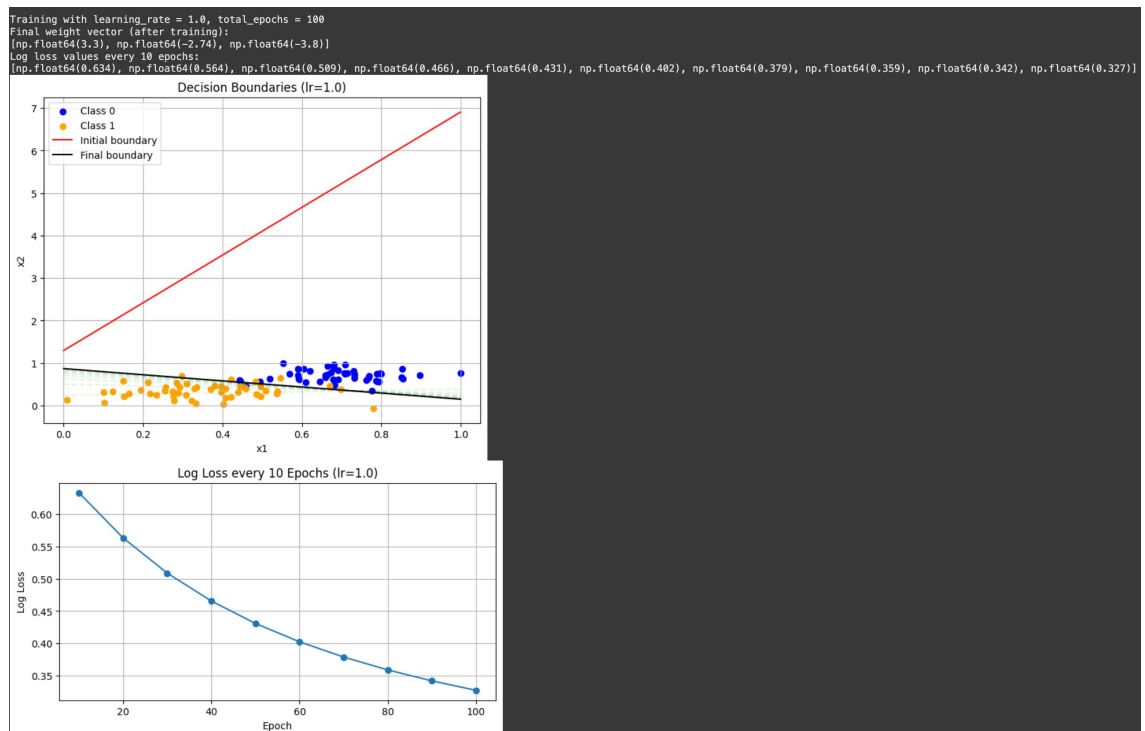
Learning rate - 0.01



## Learning rate - 0.1



## Learning rate - 1.0



Learning Rate = 0.01

Decision Boundary:

The boundary line shifted gradually and smoothly during training. There were no sudden jumps, showing that the model learned steadily over time. It took more updates, but the final decision line correctly separated the classes.

Log Loss Behavior:

The log loss decreased consistently every 10 epochs. The graph showed a gentle and steady downward slope, indicating stable learning. This proves that a small learning rate helps the model improve slowly but reliably

Learning Rate = 0.1

Decision Boundary:

The updates to the boundary were more noticeable and quicker compared to 0.01. The boundary still moved in the right direction, but there was slightly more movement between steps. The model learned faster, but the training was a bit less smooth.

Log Loss Behavior:

The log loss also decreased over time but less smoothly than with 0.01. The downward trend was still clear, meaning the model was learning, just with more fluctuation. This learning rate is a good balance between speed and stability.

Learning Rate = 1.0

Decision Boundary:

With this high learning rate, the boundary moved drastically in fewer steps. The model made large updates, which sometimes caused the line to overshoot. It still found a reasonable separating line, but the learning process was more unstable.

Log Loss Behavior:

The log loss dropped sharply every 10 epochs. The decrease was more dramatic than with lower learning rates, but the larger changes risk missing optimal values. While faster, this learning rate can lead to overshooting or less precise results.

## Conclusion

Across both approaches observed that learning rate has a strong impact on how fast and how well the perceptron learns:

- A low learning rate (0.01) results in very stable but slow learning. The model converges gradually and safely, but requires more time and updates.
- A moderate learning rate (0.1) gives the best balance between speed and accuracy. It allows the model to learn efficiently without causing instability.
- A high learning rate (1.0) leads to unstable training, with erratic decision boundary movements and fluctuating log loss, often resulting in poor convergence.

For both the heuristic and gradient descent methods, 0.1 was the most effective learning rate fast enough to train efficiently but stable enough to converge properly