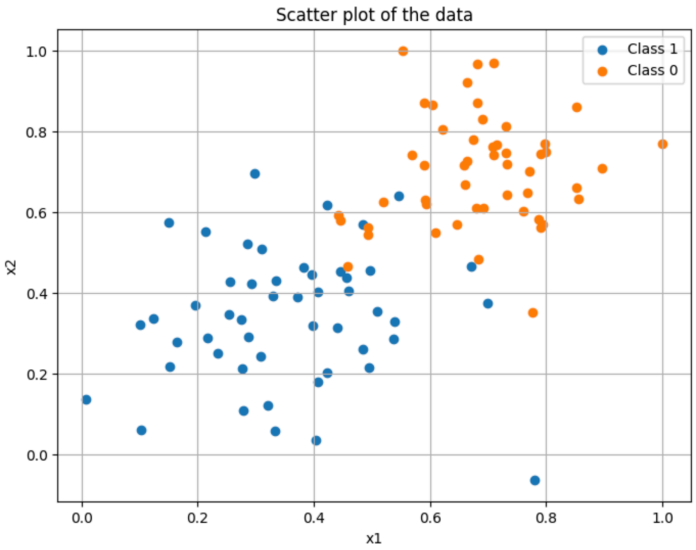
Assignment - 4 Report

Github link -

Part - 1

1. Load Data and Plot



Loaded the dataset from data.csv which contains two input features (x1, x2) and a binary label (label). The dataset was then visualized using a scatter plot.

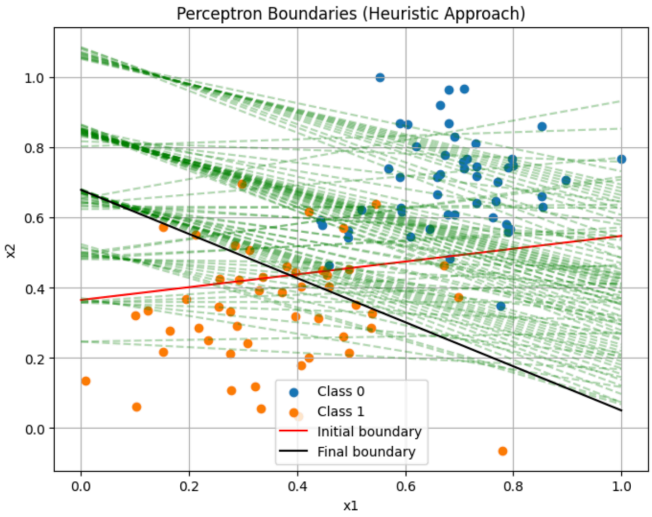
- Blue dots represent data points belonging to Class 1
- Orange dots represent data points from Class 0
- The data is well separated which visually suggests it may be linearly separable

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 92 updates the weights improved, helping the perceptron learn a better decision boundary.

- Number of updates: 92
- Initial weights: [0.34533785 0.17249454 -0.94499391]
- Final weights: [0.54533785 -0.50466946 -0.80284291]

3. Plot the initial separation



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

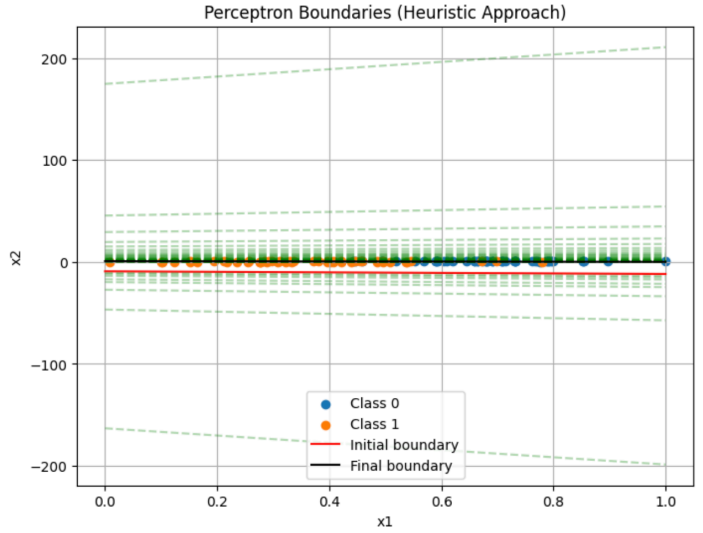
- The red line shows the initial boundary before training.
- The green dashed lines represent the boundary after each weight update.
- The black line is the final decision boundary after training completes.

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

4. Learning Rates

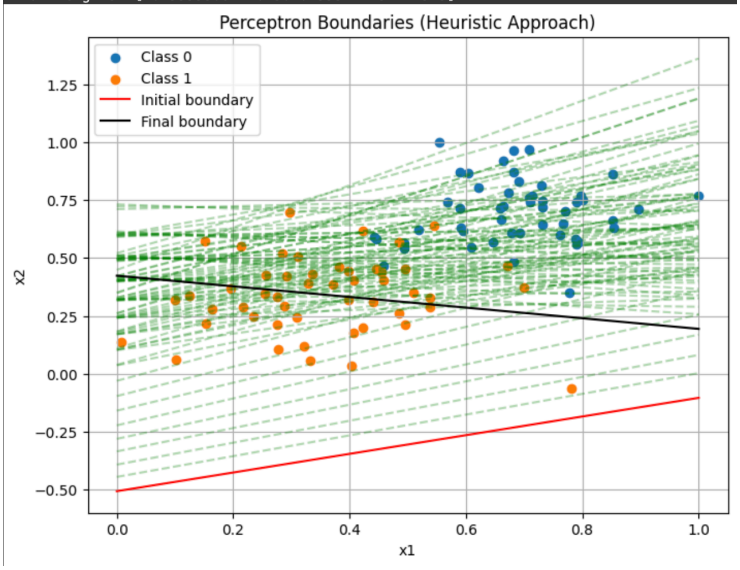
Learning rate - 0.01

```
Training with learning_rate = 0.01, epochs = 10
Number of updates: 94
Initial weights: [0.57594961 0.16343281 0.06264678]
Final weights: [ 0.15594961 -0.13543449 -0.19786252]
```



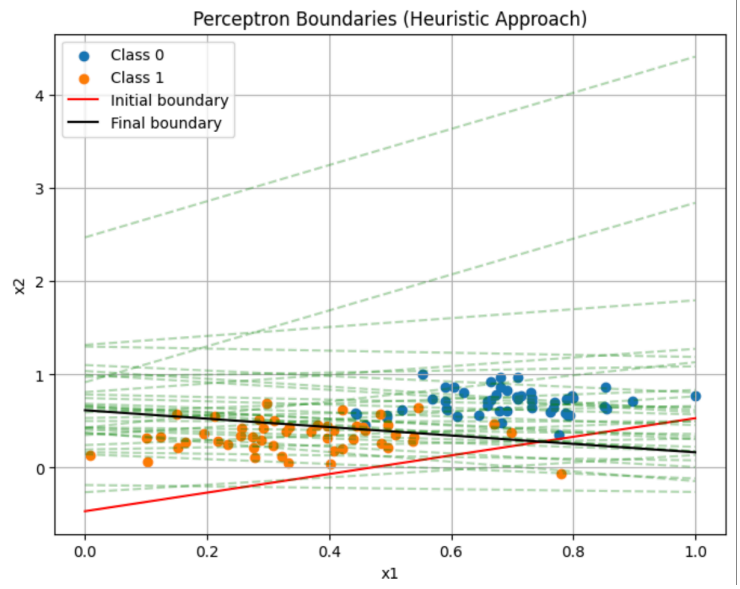
Learning rate - 0.1

```
Training with learning_rate = 0.1, epochs = 10
Number of updates: 88
Initial weights: [-0.84446919  0.67180555 -1.66734735]
Final weights: [ 0.55553081 -0.30104553 -1.31172525]
```



Learning rate - 1.0

```
Training with learning_rate = 1.0, epochs = 10
Number of updates: 33
Initial weights: [-0.42809382  0.9138427  -0.91510692]
Final weights: [ 2.57190618 -1.8838673  -4.17458592]
```

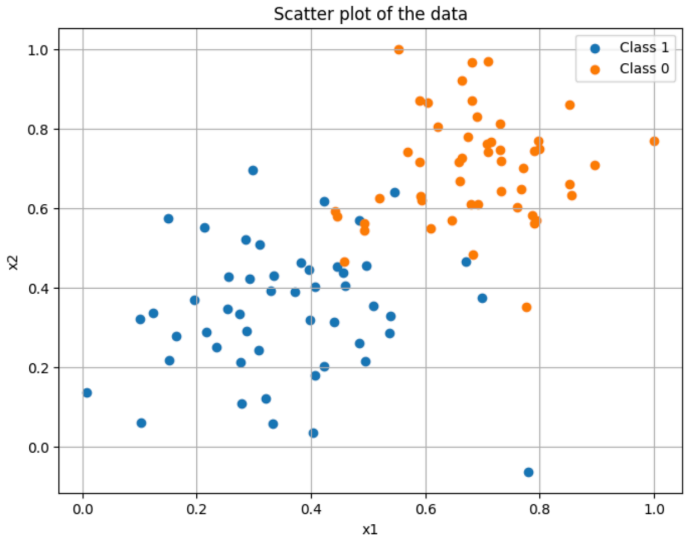


Experimented with different learning rates: 0.01, 0.1, and 1.0 to observe how they affect the decision boundary updates and convergence.

- Learning rate = 0.01
  The model took more updates (94) 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 (88) 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 (x1, x2) 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

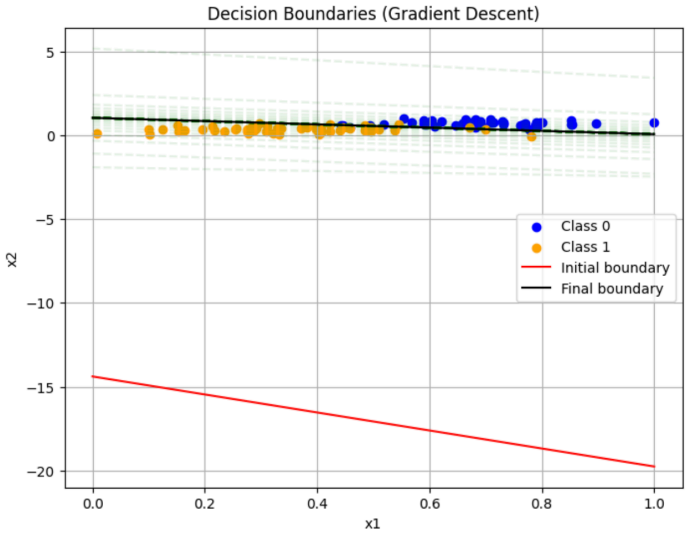
2. Implement perceptron using the Gradient Descent

started with random weights and updated them gradually using the error between predicted output (ŷ) 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: [10.16, -9.58, -9.78]
- First 5 Log Loss values: [0.558, 0.526, 0.618, 0.988, 1.619]

3. Plot the initial separation

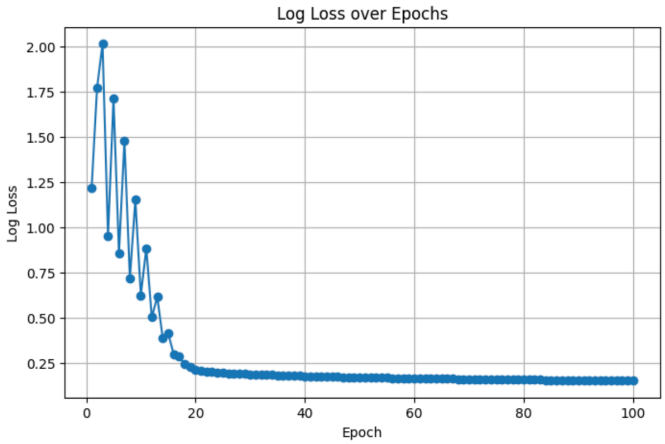


Visualized how the decision boundary evolved during training using Gradient Descent.

- 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)

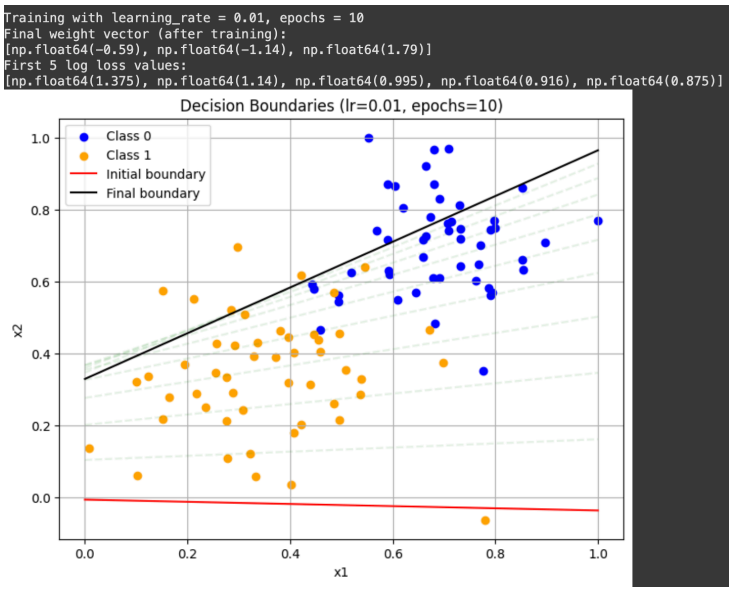


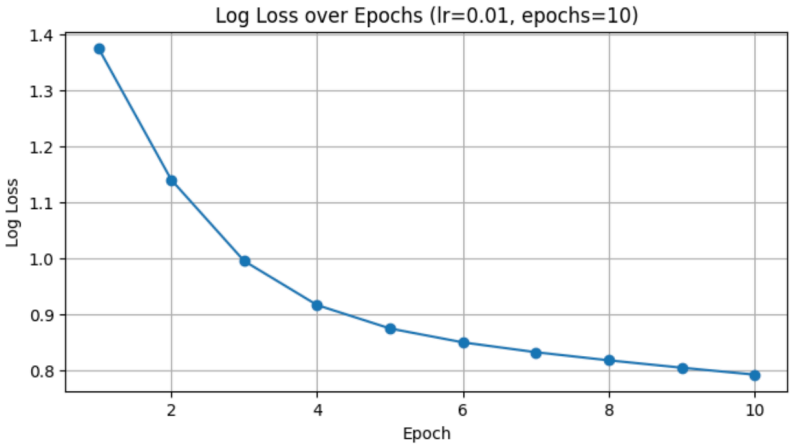
Calculated the log loss (error) during each training epoch to measure how well the model was learning.

- 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.

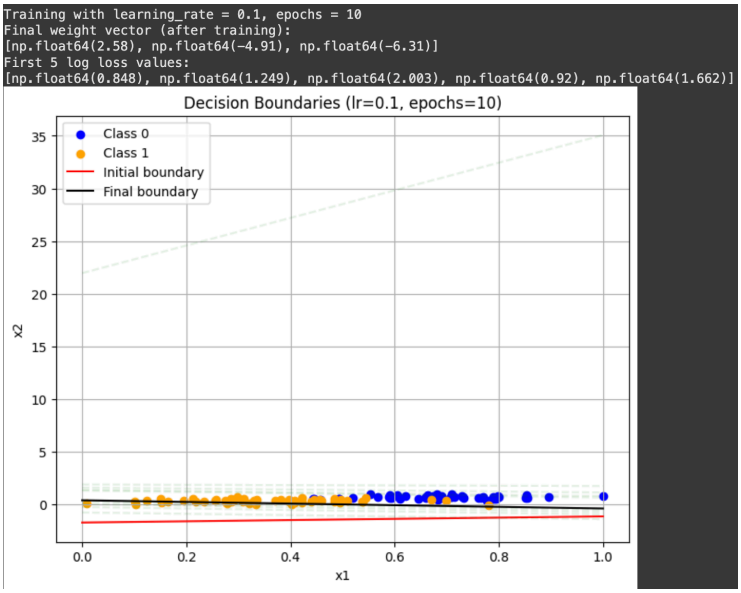
5. Learning Rate

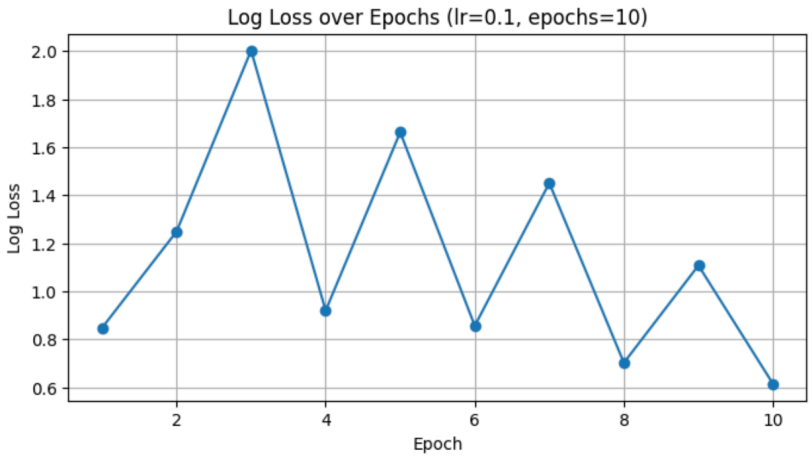
Learning rate - 0.01





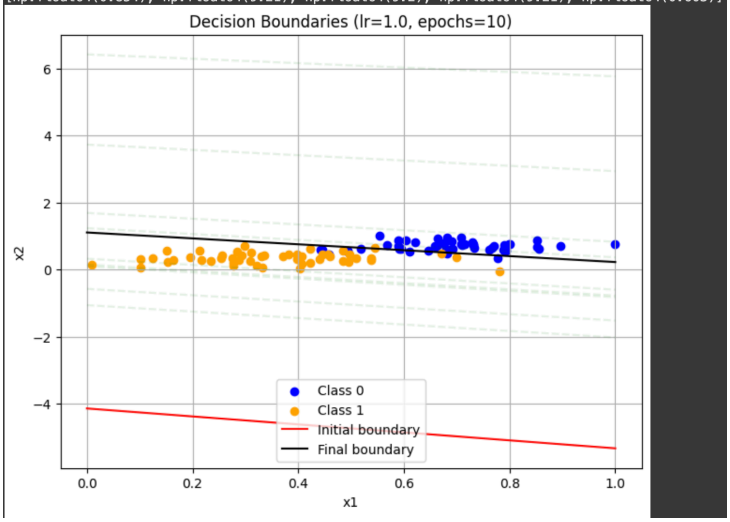
Learning rate - 0.1

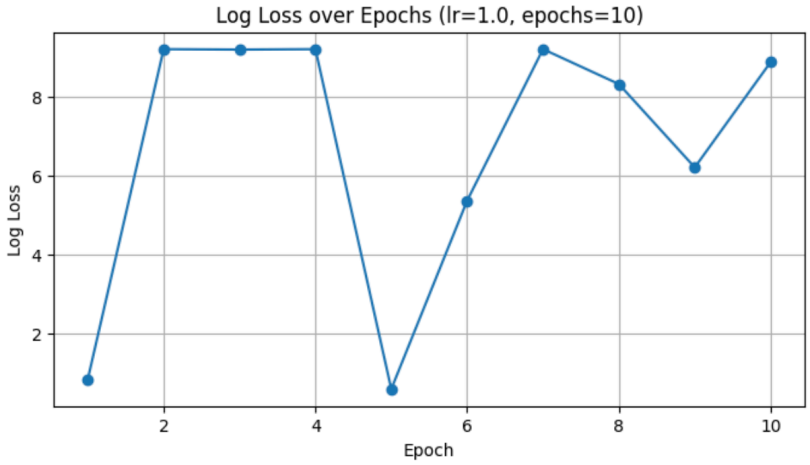




Learning rate - 1.0

```
Training with learning_rate = 1.0, epochs = 10
Final weight vector (after training):
[np.float64(56.64), np.float64(-45.2), np.float64(-51.22)]
First 5 log loss values:
[np.float64(0.854), np.float64(9.21), np.float64(9.2), np.float64(9.21), np.float64(0.603)]
```





Learning Rate = 0.01

Decision Boundary:
The boundary line changed slowly and smoothly with each update. It didn't jump too much, which made the learning process very stable. Although it took more updates to learn, the final line did a good job separating the two classes.

Log Loss Behavior:
The error (log loss) kept going down steadily over the 10 training rounds. The curve was smooth, showing that the model was learning correctly with each step. This means a low learning rate helps the model learn safely, just more slowly.

Learning Rate = 0.1

Decision Boundary:
The boundary moved faster than with 0.01. You could see more noticeable changes in the line between updates. It was learning quicker, but the movements were sometimes a bit shaky. This shows faster learning, but slightly less control.

Log Loss Behavior:
The error went up and down a few times instead of dropping smoothly. The model was still improving overall, but the jumps suggest it was learning too fast at times and might have missed the best adjustments.

Learning Rate = 1.0

Decision Boundary:
The boundary changed a lot in just a few steps. The model made big jumps when updating the line. Even though it learned faster, the line didn't always go in the best direction. This made the learning feel unstable.

Log Loss Behavior:
The error was very jumpy going up and down sharply. It didn't settle down well. This means the model had trouble learning and might even start to perform worse if trained longer. A learning rate this high is often too aggressive.

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.
- A moderate learning rate (0.1) gives the best balance between speed and accuracy.
- A high learning rate (1.0) leads to unstable training, with erratic behavior and 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