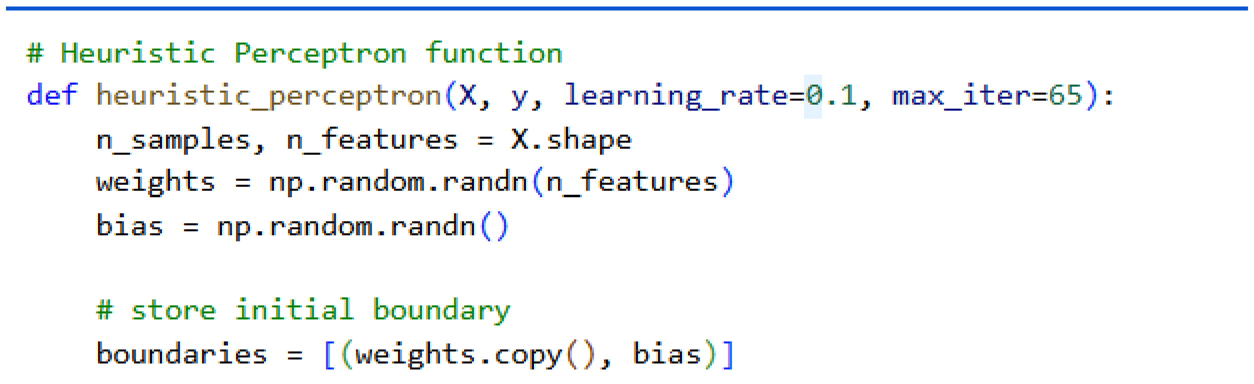
PERCEPTRON:

Introduction :

This assignment is about training a perceptron on a simple 2D binary classification dataset and studyinghow the learning rate affects training. In Part 1 I used the basic heuristic perceptron update, and in Part 2 I used a gradient-descent version with sigmoid and log-loss. For each method and learning rate, I plotted the decision boundary over time and analyzed how stable and effective the training was.

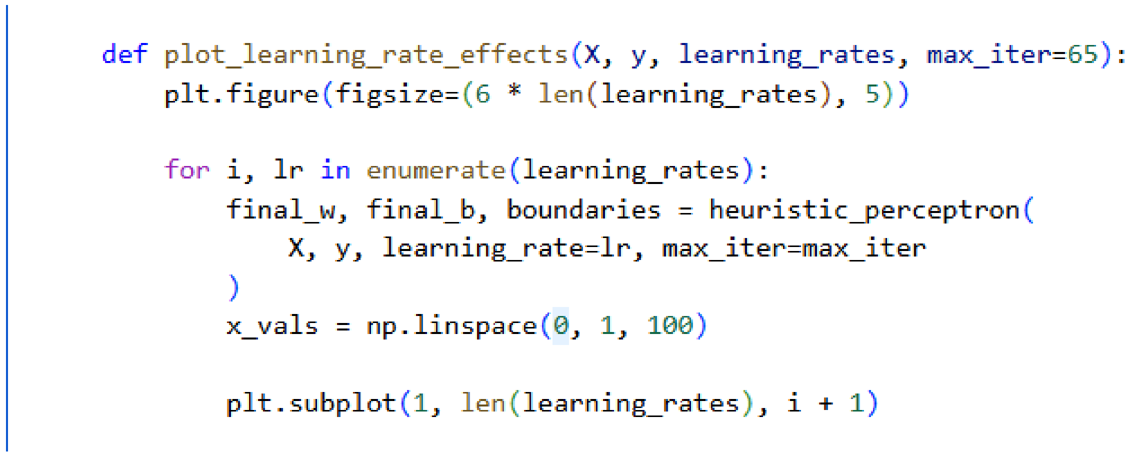
Heuristic function:



```
# Heuristic Perceptron function
def heuristic_perceptron(X, y, learning_rate=0.1, max_iter=65):
    n_samples, n_features = X.shape
    weights = np.random.randn(n_features)
    bias = np.random.randn()

    # store initial boundary
    boundaries = [(weights.copy(), bias)]
```

The plotting function:

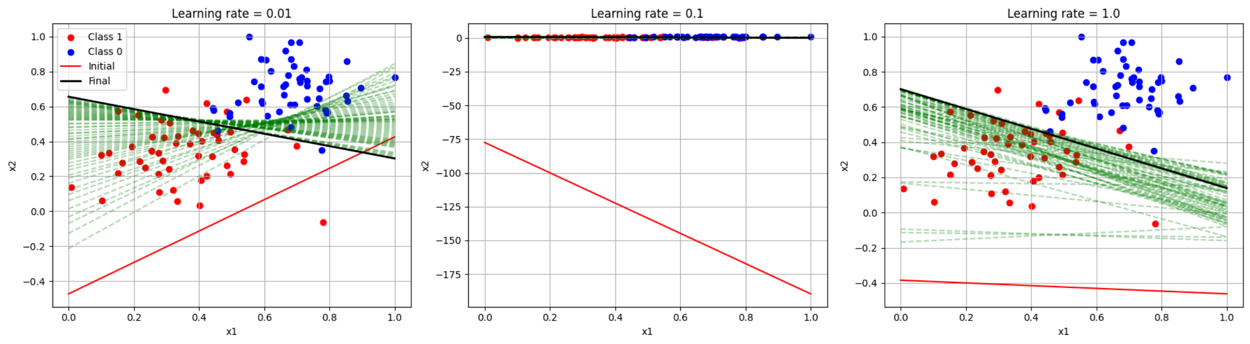


```
def plot_learning_rate_effects(X, y, learning_rates, max_iter=65):
    plt.figure(figsize=(6 * len(learning_rates), 5))

    for i, lr in enumerate(learning_rates):
        final_w, final_b, boundaries = heuristic_perceptron(
            X, y, learning_rate=lr, max_iter=max_iter
        )
        x_vals = np.linspace(0, 1, 100)

        plt.subplot(1, len(learning_rates), i + 1)
```

Heuristic plot:



Analysis :

For the heuristic perceptron, the learning rate changed the behavior a lot.

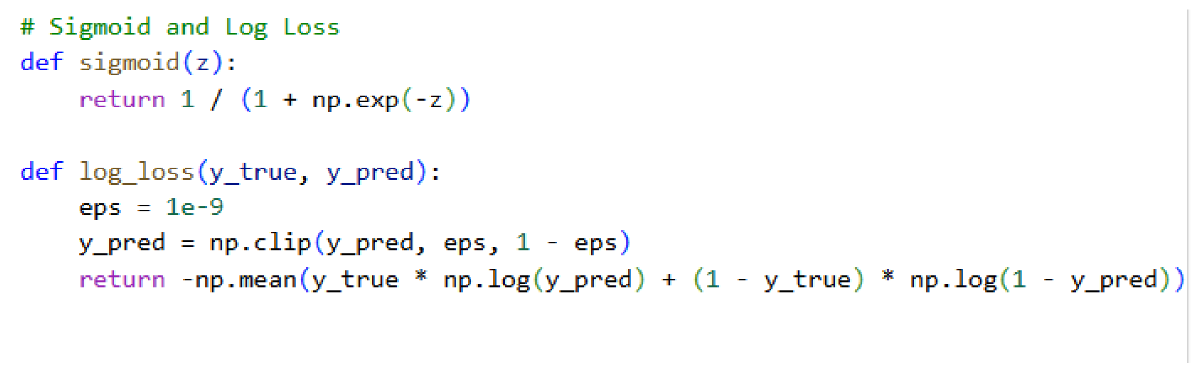
With learning rate 0.01, the line moved slowly in small steps. It took many passes, but the line stayed stable and ended up in a reasonable place between the red and blue points.

With learning rate 0.1, the steps were too big. The line shot far away from the data and even went off the visible plot. The green lines between epochs jumped around a lot. This means the weights were blowing up and training was not stable.

With learning rate 1.0, the jumps were even bigger. The line kept moving in large jumps from epoch to epoch. It finishes with some separating line, but the path is messy and depends a lot on the random start.

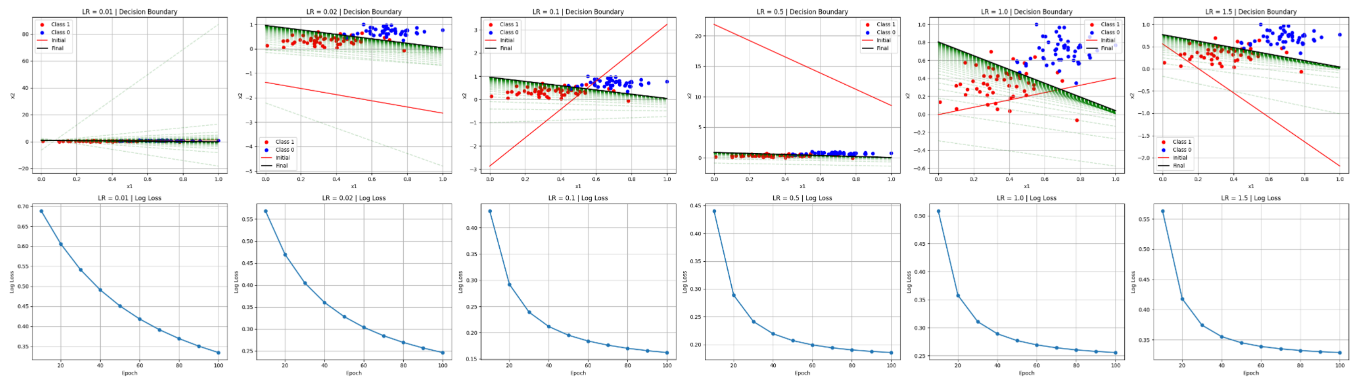
So, for this heuristic model, 0.01 is the only safe learning rate. The larger learning rates (0.1 and 1.0) make the training unstable and are not good choices.

Gradient descent function:



```
# Sigmoid and Log Loss
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def log_loss(y_true, y_pred):
    eps = 1e-9
    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))
```



Analysis:

In the gradient-descent perceptron, I used a sigmoid output and log-loss, and tried learning rates 0.01, 0.02, 0.1, 0.5, 1.0, 1.5. For each setting, I looked at how the decision boundary moved and how the log-loss changed over epochs.

- LR = 0.01
  The model learned very slowly. The line moved in small steps and needed many epochs to settle into a reasonable separator. The log-loss went down steadily, but the progress was slow.

- LR = 0.02
  This was faster than 0.01 but still stable. The boundary moved more each epoch and reached a good position earlier. The log-loss dropped quicker and then leveled off at a low value. This rate is safe and efficient.

- LR = 0.1
  This gave the best overall behavior. The decision boundary moved into a good spot in the first part of training and then only needed small adjustments. The log-loss dropped sharply at the beginning and then flattened out. Training was fast and stable, with no weird jumps.

- LR = 0.5, 1.0, 1.5
  These learning rates were much more aggressive. The decision boundary jumped more between epochs, and the log-loss curves were less smooth, but they still generally went down and reached similar final values as 0.1. There was no clear improvement over 0.1, just bigger and riskier updates.

Overall, the gradient-descent perceptron trained well for all these learning rates, but 0.1 gave the best mix of speed and stability.

Comparison

Heuristic vs Gradient-Descent Perceptron

- In the heuristic perceptron, only a small learning rate (0.01) worked well. With 0.1 and 1.0, the updates were too big, the decision boundary shot off the visible region, and the lines between epochs jumped around. Training became unstable.

- In the gradient-descent perceptron, all tested learning rates from 0.01 to 1.5 produced a sensible final decision boundary, and the log-loss kept going down over time. A learning rate of 0.1 trained the fastest while still staying stable.

At the same learning rate 0.1, the difference is clear:
the heuristic model blows up, while the gradient-descent model converges smoothly and its progress is easy to see in the loss curve. The heuristic perceptron is simpler, but very sensitive to the learning rate. The gradient-descent version is more robust and easier to tune.

Conclusion

These experiments show that the heuristic perceptron only behaves well with a very small learning rate on this dataset, and larger steps make it unstable. The gradient-descent perceptron with sigmoid and log-loss handles a much wider range of learning rates and trains smoothly, with a learning rate of 0.1 giving the best results. For this problem, the gradient-descent approach is clearly the more reliable and practical way to train a perceptron.