CSCI 580
ASSIGNMENT #5

Yash Sharma

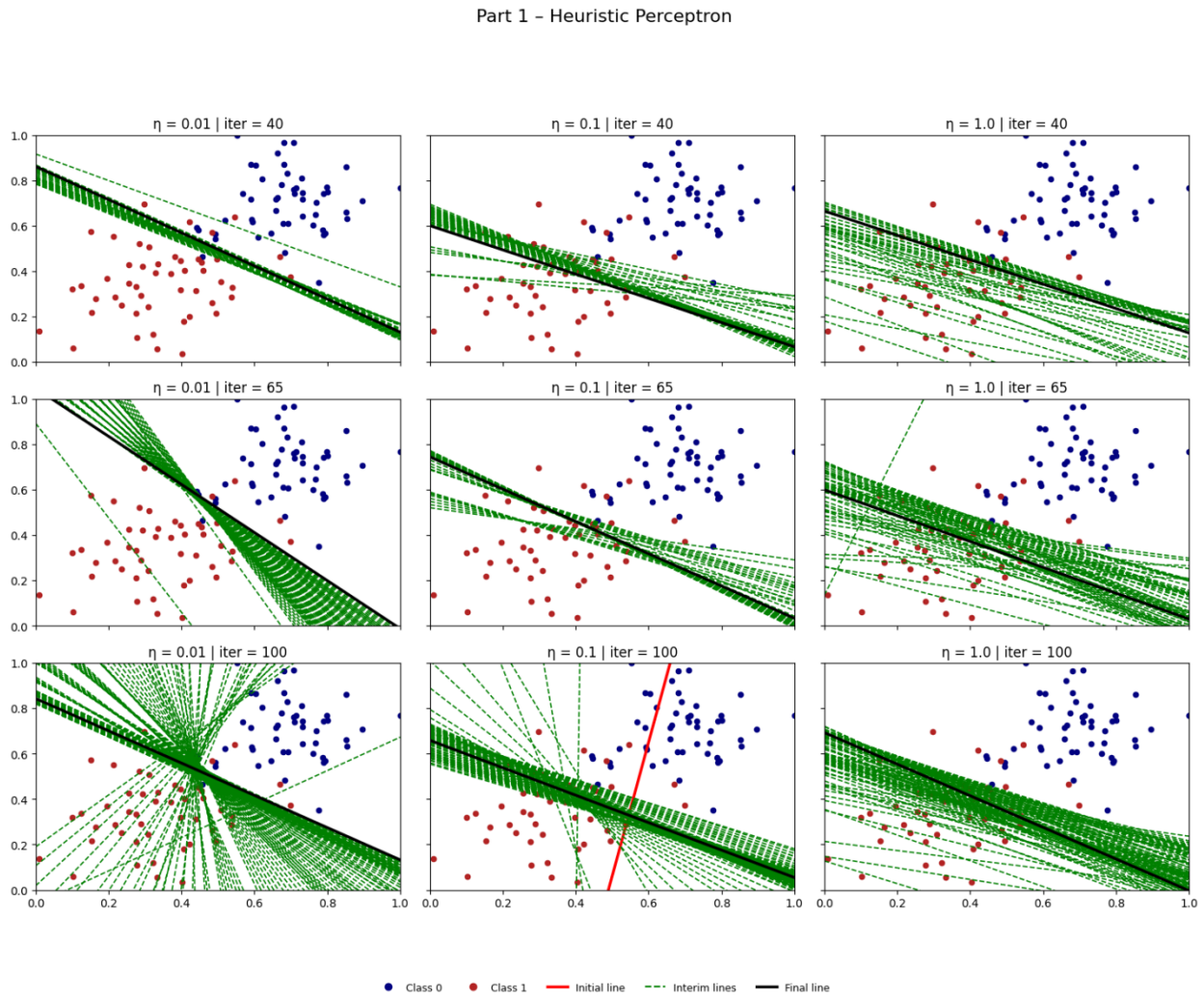
011940046

1.

Code Snippet:

```
def perceptron_heuristic(X, y, lr=0.1, max_iters=65):
    rng = np.random.default_rng()
    w   = rng.standard_normal(2)
    b   = rng.standard_normal()
    history = [(w.copy(), b)]                       # store initial line
    for _ in range(max_iters):
        for xi, yi in zip(X, y):
            y_hat = 1 if (np.dot(w, xi) + b) >= 0 else 0
            if y_hat != yi:                         # mis-classified
                if y_hat == 0:                      # move toward +class
                    b += lr;  w += lr * xi
                else:                               # move toward -class
                    b -= lr;  w -= lr * xi
        history.append((w.copy(), b))
    return history

lr_list   = [0.01, 0.1, 1.0]
iter_list = [40, 65, 100]


legend_handles = [
    mlines.Line2D([], [], color='navy',      marker='o', linestyle='None',
                  markersize=6, label='Class 0'),
    mlines.Line2D([], [], color='firebrick', marker='o', linestyle='None',
                  markersize=6, label='Class 1'),
    mlines.Line2D([], [], color='red',   linewidth=2.5, label='Initial line'),
    mlines.Line2D([], [], color='green', linewidth=1.5, linestyle='dashed',
                  label='Interim lines'),
    mlines.Line2D([], [], color='black', linewidth=2.5, label='Final line')
]

fig, axes = plt.subplots(len(iter_list), len(lr_list),
                         figsize=(5*len(lr_list), 4*len(iter_list)),
                         sharex=True, sharey=True)

for row, max_iter in enumerate(iter_list):
    for col, lr in enumerate(lr_list):
        ax   = axes[row, col] if axes.ndim == 2 else axes[col]
        hist = perceptron_heuristic(X, y, lr=lr, max_iters=max_iter)

        # data points
        ax.scatter(X[y == 0, 0], X[y == 0, 1], c='navy',      s=20)
        ax.scatter(X[y == 1, 0], X[y == 1, 1], c='firebrick', s=20)

        # decision boundaries
        draw_line(ax, *hist[0],  color='red',   style='solid',  width=2.5)   # initial
        for w, b in hist[1:-1]:
            draw_line(ax, w, b,  color='green', style='dashed', width=1.2)   # interim
        draw_line(ax, *hist[-1], color='black', style='solid',  width=2.5)   # final

        ax.set_xlim(0, 1);  ax.set_ylim(0, 1)
        ax.set_title(f'η = {lr} | iter = {max_iter}')

fig.legend(handles=legend_handles, loc='lower center',
           ncol=len(legend_handles), frameon=False, fontsize=9)

fig.suptitle('Part 1 - Heuristic Perceptron', fontsize=16, y=1.02)
fig.tight_layout(rect=[0, 0.08, 1, 0.95])
plt.show()
```

Graphs:



Analysis:

The graphs illustrate how the plain perceptron's decision boundary evolves under three learning-rates and three fixed iteration budgets. With the smallest step size (η = 0.01) the boundary inches forward at 40 updates and finally reaches the class gap by 65, but the tight burst of green lines at 100 iterations reveals that once it gets there it begins to oscillate, wasting extra updates without further improvement. At the mid-range rate (η = 0.1) almost all progress happens in the first 40 iterations; by 65 the separator is already locked in and the additional passes add only a handful of nearly identical traces evidence that this setting still delivers the best balance of speed and stability. With the largest step (η = 1.0) the boundary leaps into a separating position almost immediately, yet each oversized update overshoots the margin, so the dashed-green fan remains wide even after 100 iterations; the model separates quickly but never truly settles. Taken together, the new

grid reinforces the earlier conclusion: a very small learning-rate is reliable but inefficient, an excessively large rate converges fastest but keeps jittering, and the middle value (0.1) remains the most efficient, well-behaved choice for the heuristic perceptron on this dataset.

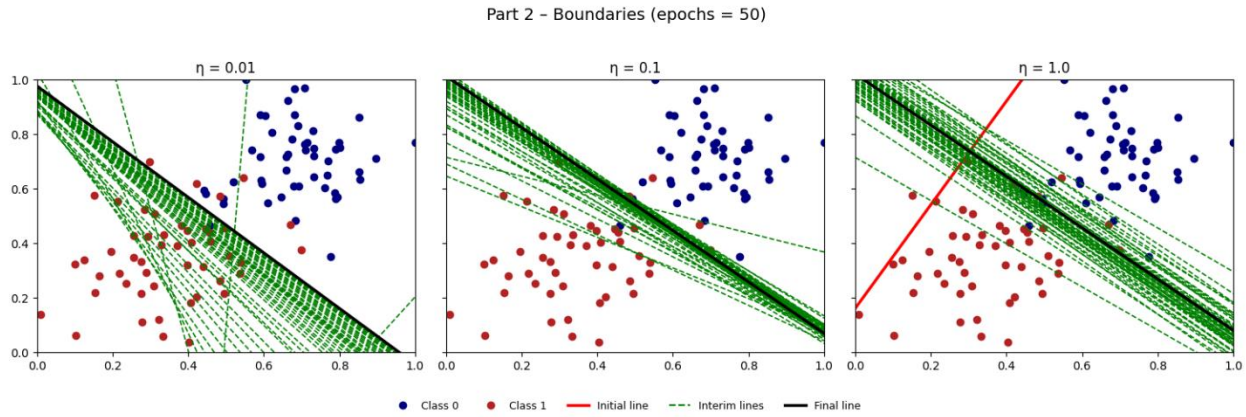
2.

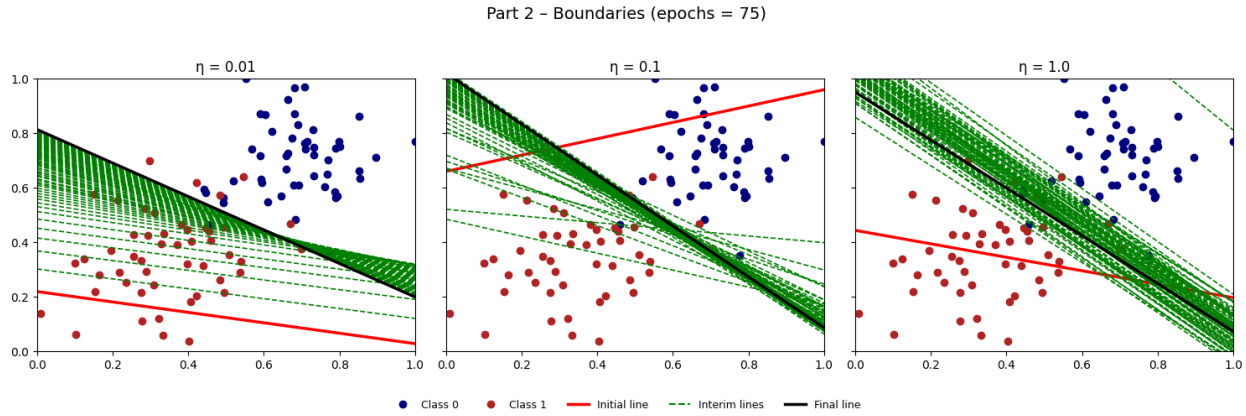
Code Snippets:

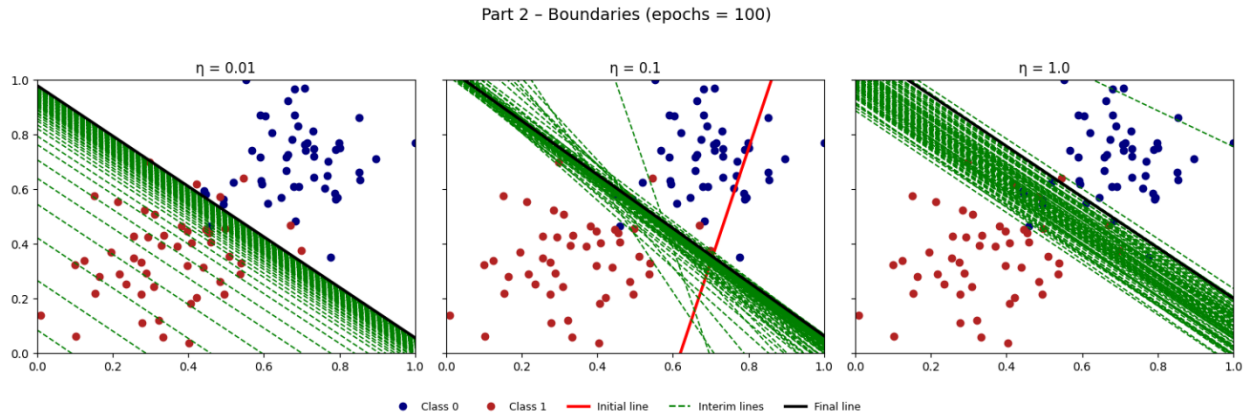
```
def perceptron_gd(X, y, lr=0.05, epochs=100):
    rng = np.random.default_rng()
    w   = rng.standard_normal(2)
    b   = rng.standard_normal()
    history, losses = [(w.copy(), b)], []
    for ep in range(epochs):
        for idx in rng.permutation(len(X)):
            xi, yi = X[idx], y[idx]
            err    = yi - sigmoid(np.dot(w,xi) + b)
            b     += lr * err
            w     += lr * err * xi
        history.append((w.copy(), b))
        y_hat  = sigmoid(X @ w + b)
        ll     = -np.mean(y*np.log(y_hat+1e-9) + (1-y)*np.log(1-y_hat+1e-9))
        losses.append(ll)
    return history, losses

lr_list_p2 = [0.01, 0.1, 1.0]
epoch_list = [50, 75, 100]


legend_handles = [
    mlines.Line2D([], [], color='navy',      marker='o', linestyle='None',
                  markersize=6, label='Class 0'),
    mlines.Line2D([], [], color='firebrick', marker='o', linestyle='None',
                  markersize=6, label='Class 1'),
    mlines.Line2D([], [], color='red',   linewidth=2.5, label='Initial line'),
    mlines.Line2D([], [], color='green', linewidth=1.5, linestyle='dashed',
                  label='Interim lines'),
    mlines.Line2D([], [], color='black', linewidth=2.5, label='Final line')
]

for epochs in epoch_list:
    fig, axes = plt.subplots(1, len(lr_list_p2),
                             figsize=(15, 5), sharex=True, sharey=True)

    for ax, lr in zip(axes, lr_list_p2):
        hist, loss = perceptron_gd(X, y, lr=lr, epochs=epochs)

        ax.scatter(X[y == 0, 0], X[y == 0, 1], c='navy')
        ax.scatter(X[y == 1, 0], X[y == 1, 1], c='firebrick')

        draw_line(ax, *hist[0],  color='red',   style='solid',  width=2.5)   # initial
        for w, b in hist[1:-1]:
            draw_line(ax, w, b, color='green', style='dashed', width=1.2)    # interim
        draw_line(ax, *hist[-1], color='black', style='solid', width=2.5)    # final

        ax.set_xlim(0, 1);  ax.set_ylim(0, 1)
        ax.set_title(f'η = {lr}')

    fig.legend(handles=legend_handles, loc='lower center',
               ncol=len(legend_handles), frameon=False, fontsize=9)

    fig.suptitle(f'Part 2 – Boundaries (epochs = {epochs})', fontsize=14)
    fig.tight_layout(rect=[0, 0.08, 1, 0.95])   # leave space for legend
    plt.show()
```

```python
# Log-loss curve grid
lr_list = [0.01, 0.1, 1.0]
epoch_list = [50, 75, 100]

fig, axes = plt.subplots(len(epoch_list), len(lr_list), figsize=(15, 10), sharex=False, sharey=False)

for row, epochs in enumerate(epoch_list):
    for col, lr in enumerate(lr_list):
        _, loss = perceptron_gd(X, y, lr=lr, epochs=epochs)
        ax = axes[row, col]
        ep_axis = np.arange(10, epochs + 1, 10)
        ax.plot(ep_axis, np.array(loss)[9::10], marker="o", color="navy")
        ax.set_title(f"η = {lr}, epochs = {epochs}")
        ax.set_xlabel("Epoch")
        ax.set_ylabel("Log-loss")
        ax.grid(alpha=0.3)

fig.suptitle("Log-Loss Curves – Gradient Descent Perceptron", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

Graphs:



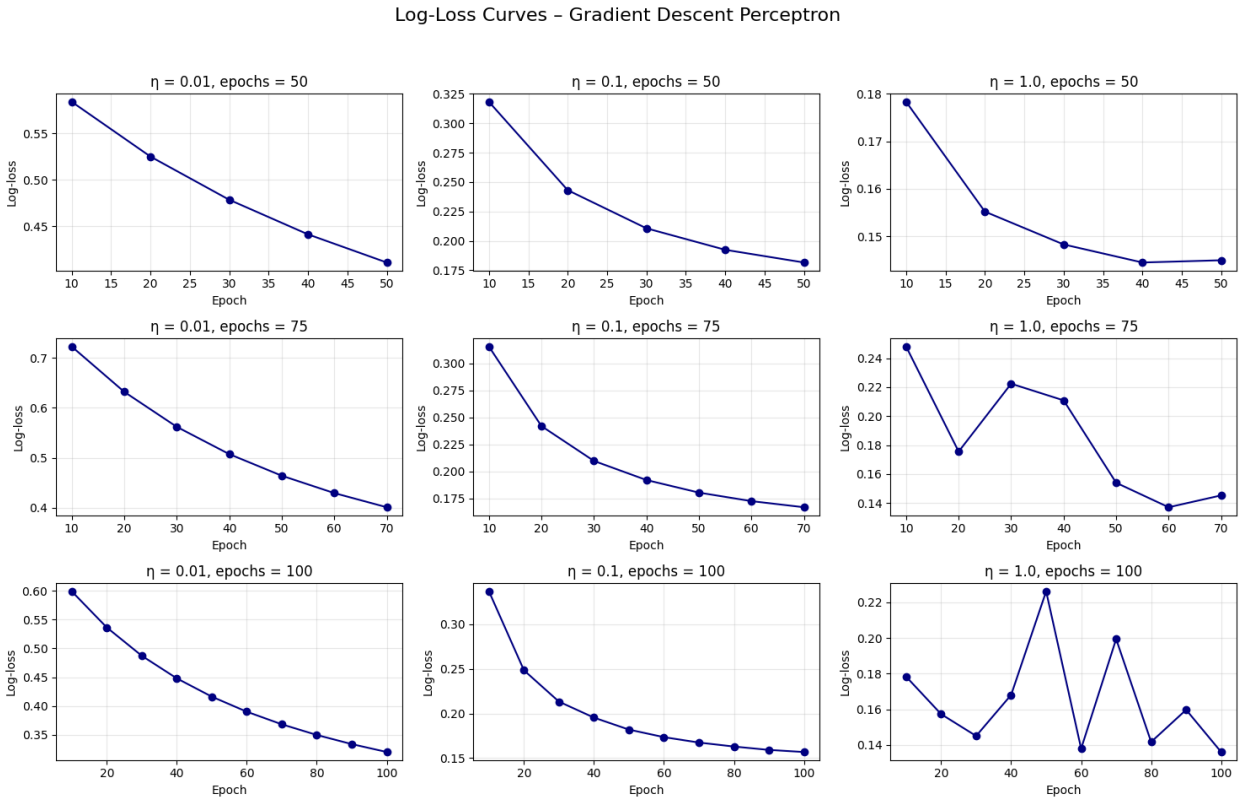




Analysis:

The plots show how the perceptron's decision boundary evolves when trained using gradient descent over different learning-rates and epoch counts. With the smallest learning-rate (η = 0.01), the model converges slowly but steadily. At 50 epochs the separator is close, but not fully aligned; by 75 it finds a clean margin, and at 100 the boundary appears well-settled, with a dense trail of intermediate lines showing a consistent and gradual path toward convergence. At the mid-range (η = 0.1), the boundary converges faster. Already by 50 epochs the black line is near-optimal, though the green paths still show some angular noise. By 75 and 100 epochs, the final line stabilizes, but the early jumpy updates (as seen in the red initial line placement) make the path a bit more erratic compared to η = 0.01. With the largest step (η = 1.0), the model makes very aggressive updates early on. Although it reaches the correct boundary quickly, it continues to jitter around the optimal spot across all three epoch settings. The separator forms fast, but the wide spread of green dashed lines shows ongoing overshooting. Overall, a small η is slower but smoother, a large η is fast but chaotic, and η = 0.1 again provides a good balance—getting close to optimal early while stabilizing with a bit more time. These patterns highlight how learning-rate and epoch choices impact both convergence speed and path stability in the gradient-based perceptron.

Error Plot:



Analysis:

The log-loss curves reveal how different learning rates and epoch budgets affect model convergence. With **η = 0.01**, the loss decreases gradually and consistently as epochs increase. The decline is smooth across all three plots (50, 75, and 100 epochs), confirming that this small step size leads to stable, predictable convergence—but at the cost of slower improvement per epoch.

In contrast, **η = 0.1** shows a faster loss reduction early on. At 50 and 75 epochs, the curve drops sharply in the first 10–20 epochs and then flattens out, suggesting that the model is reaching a good solution faster. By 100 epochs, the loss reaches a low point and plateaus, showing that the model has effectively converged. This learning rate continues to show the most efficient balance between speed and stability.

With **η = 1.0**, however, the plots become noticeably erratic. While the initial loss does drop quickly, the curves across all epoch settings show sharp fluctuations, especially between epochs 10 and 60. Even by 100 epochs, the log-loss still oscillates rather than steadily decreasing. This suggests that the learning rate is too high, causing the model to overshoot optimal weights during updates.

In summary, the log-loss graphs reinforce the pattern seen in the boundary plots:

- A small learning rate (0.01) is stable but slow to optimize,
- A large learning rate (1.0) causes instability,
- And a mid-range value (0.1) offers the fastest and most stable convergence overall. This makes **η = 0.1 the most balanced and effective choice** for training the gradient-descent perceptron on this dataset.

GitHub Link:

https://github.com/Yash92100/CSCI580_Assignment-5