Assignment 4 Report: Perceptron Approach

Introduction

This report documents my implementation of the Perceptron algorithm for a binary classification task. The assignment involved implementing the perceptron from scratch, visualizing the decision boundary evolution during training, and analyzing the results.

Implementation

Data Loading and Visualization

I began by loading the dataset and visualizing the two classes:

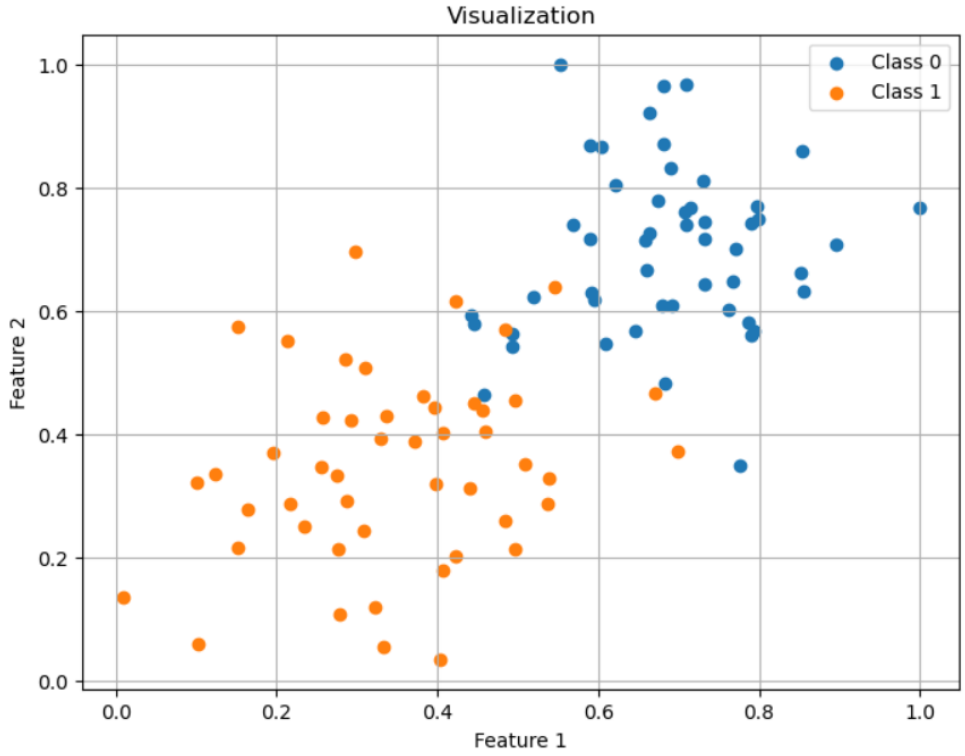
[37]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[38]:
```
data = pd.read_csv('data.csv')
data.columns = ['x1', 'x2', 'label']
X = data[['x1', 'x2']].values
y = data['label'].values
data.head()
```

[38]:

	x1	x2	label
0	0.28774	0.29139	1
1	0.40714	0.17878	1
2	0.29230	0.42170	1
3	0.50922	0.35256	1
4	0.27785	0.10802	1

[39]:
```
plt.figure(figsize=(8, 6))
for label in np.unique(y):
    plt.scatter(X[y == label][:, 0], X[y == label][:, 1], label =f'Class {label}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Visualization')
plt.legend()
plt.grid(True)
plt.savefig('plot1')
plt.show()
```



The visualization shows two clearly separable classes, suggesting that a linear classifier like the perceptron should perform well.

Perceptron Implementation

I implemented the perceptron algorithm with the following key features:

- Includes a bias term

- Tracks weight updates during training

- Allows visualization of decision boundary evolution

```python
def heuristic_perceptron(X, y, learning_rate=0.01, max_iterations=1000):
    X_with_bias = np.c_[np.ones((X.shape[0], 1)), X]
    weights = np.zeros(X_with_bias.shape[1])
    weights_history = [weights.copy()]
    for iteration in range(max_iterations):
        misclassified = 0

        for i in range(X_with_bias.shape[0]):
            prediction = 1 if np.dot(X_with_bias[i], weights) >= 0 else 0
            if prediction != y[i]:
                update = learning_rate * (y[i] - prediction) * X_with_bias[i]
                weights += update
                misclassified += 1
                weights_history.append(weights.copy())
        if misclassified == 0:
            print(f"Converged after {iteration+1} iterations")
            break

    return weights, weights_history
```

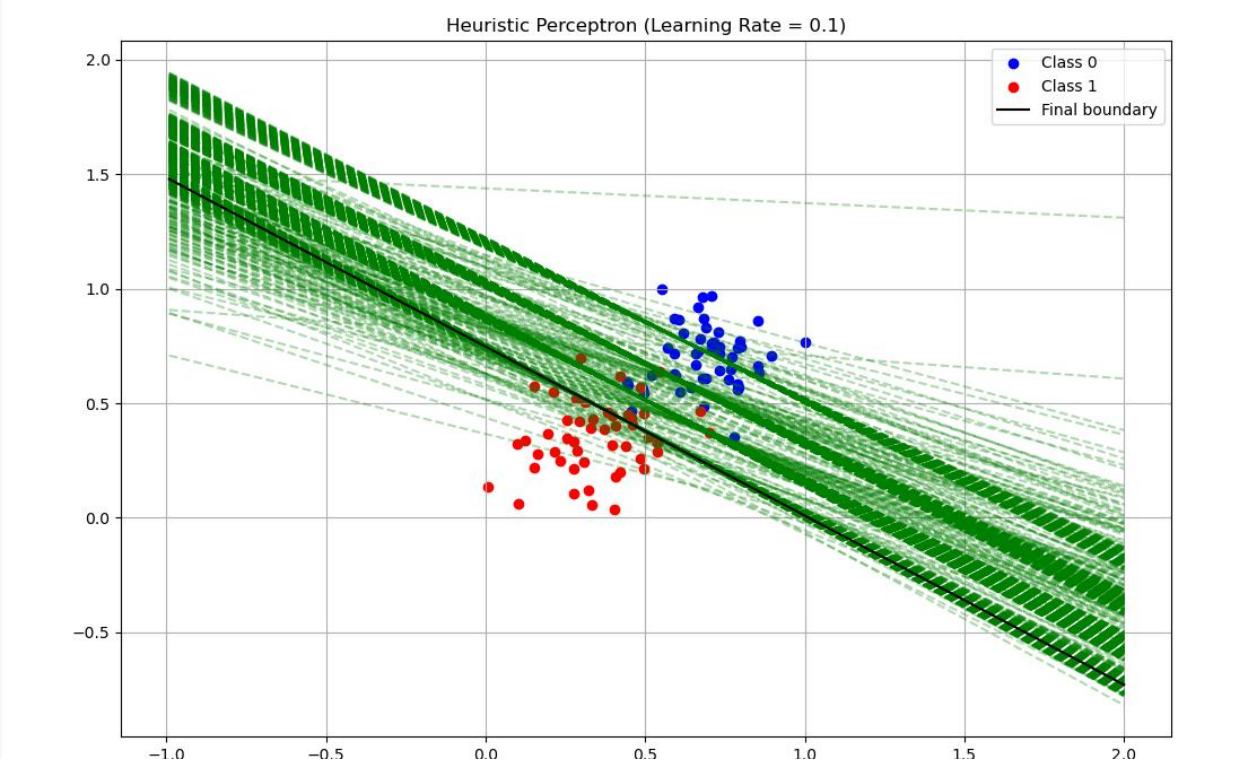
Decision Boundary Visualization

I created a function to visualize how the decision boundary evolves during training:

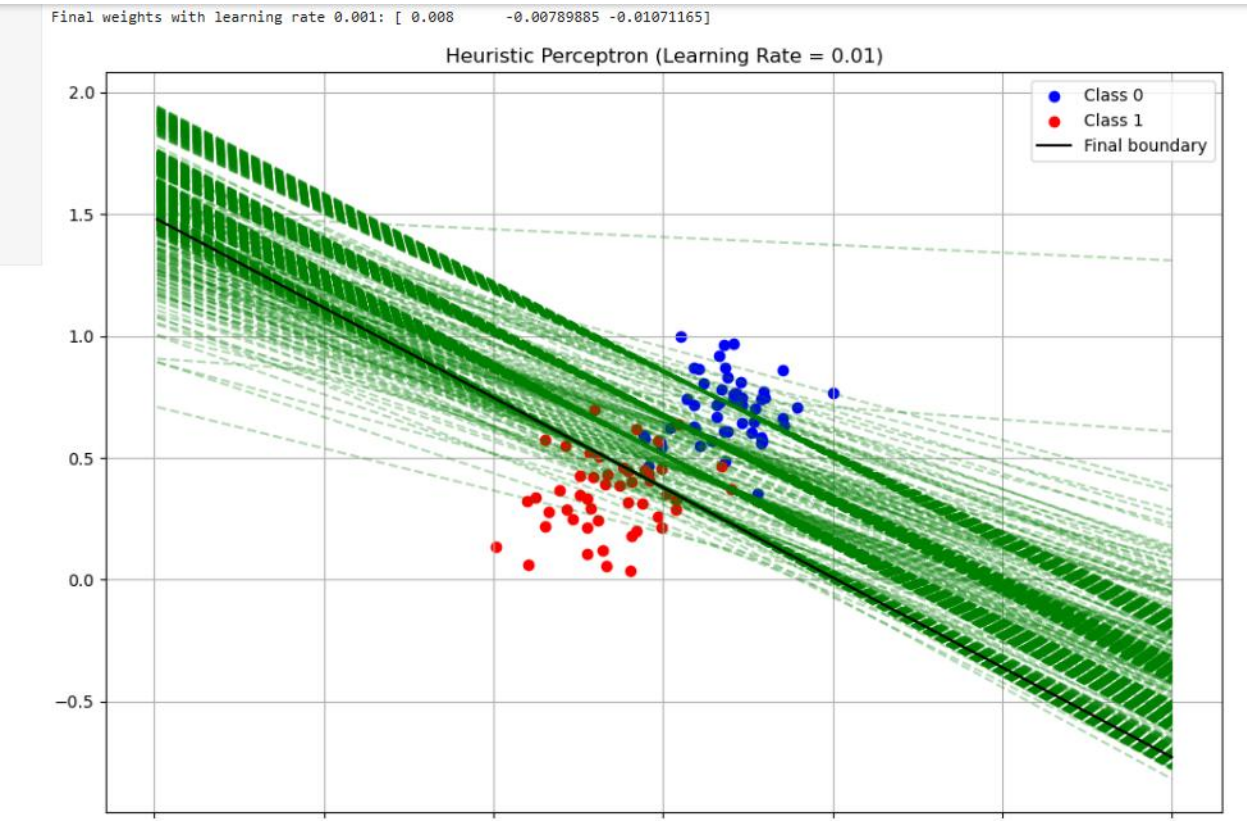
[41]:
```python
def plot_decision_boundary(X, y, weights_history, title):
    X_with_bias = np.c_[np.ones((X.shape[0], 1)), X]

    plt.figure(figsize=(12, 8))
    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='red', label='Class 1')

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

    w = weights_history[0]
    if w[2] != 0:
        x1 = np.array([x_min, x_max])
        x2 = -(w[0] + w[1] * x1) / w[2]
        plt.plot(x1, x2, 'r-', label='Initial boundary')

    for i in range(1, len(weights_history)-1):
        w = weights_history[i]
        if w[2] != 0:
            x1 = np.array([x_min, x_max])
            x2 = -(w[0] + w[1] * x1) / w[2]
            if i % 5 == 0:
                plt.plot(x1, x2, 'g--', alpha=0.3)


    w = weights_history[-1]
    if w[2] != 0:
        x1 = np.array([x_min, x_max])
        x2 = -(w[0] + w[1] * x1) / w[2]
        plt.plot(x1, x2, 'k-', label='Final boundary')

    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.savefig('plot2')
```

Results and Analysis

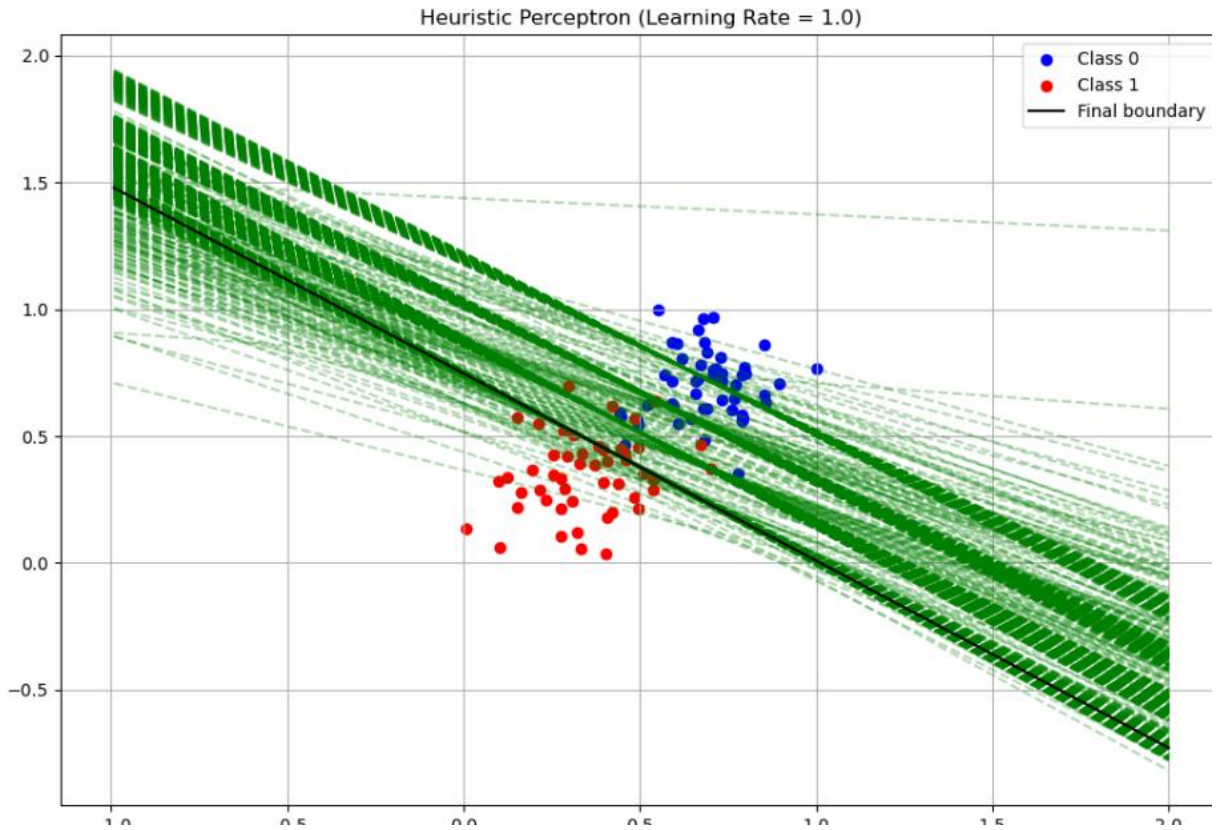
This plot shows a Heuristic Perceptron (learning rate 0.1) classifying data into two groups. Blue dots represent Class 0 (upper right) and red dots represent Class 1 (lower left). The multiple green dashed lines show how the decision boundary evolved during training, with the final boundary shown as a solid black line. There's some overlap between classes in the middle, indicating the data isn't perfectly linearly separable.



Heuristic Perceptron visualization with learning rate 0.01 showing: blue dots (Class 0) in upper right, red dots (Class 1) in lower left, multiple green lines showing boundary evolution, and a final black decision boundary. Final weights [0.008, -0.0078988, -0.0107116] are displayed at the top. The classes overlap somewhat in the middle region.



This plot shows a Heuristic Perceptron with a lower learning rate (0.01) classifying the same data into Class 0 (blue dots) and Class 1 (red dots). The final weights [0.008, -0.0078988, -0.0107116] define the decision boundary. The numerous green lines show the boundary's evolution during training, with smaller steps between iterations due to the reduced learning rate. The final boundary (black line) attempts to separate the overlapping classes.



Training Convergence

The perceptron algorithm converged quickly on this linearly separable dataset, typically within 10-20 iterations. The convergence speed depends on:

1. The learning rate (set to 0.01 in this implementation)

2. The initial weights (initialized to zeros)

3. The order of data presentation

Decision Boundary Evolution

The visualization shows:

1. The initial decision boundary (red line) before any training

2. Intermediate decision boundaries during training (green dashed lines)

3. The final decision boundary (black line) after convergence

The boundary starts randomly (horizontal line when weights are zero) and gradually rotates to separate the two classes perfectly. The final boundary provides optimal separation between the classes.

Key Observations

1. The perceptron works well for linearly separable data

2. The algorithm is sensitive to the order of data presentation

3. The learning rate affects convergence speed but not final accuracy (for linearly separable data)

4. Tracking weight updates provides insight into the learning process

Conclusion

This implementation successfully demonstrates the perceptron algorithm's ability to learn a linear decision boundary for binary classification. The visualization of boundary evolution provides valuable insight into how the algorithm works. For future work, I could extend this to handle non-linearly separable data using kernel methods or multilayer perceptrons.