Rajshahi University of Engineering & Technology

Heaven's Light is Our Guide

Course Code: CSE 4204

Course Title: Sessional based on CSE 4203

Experiment no.: 02

Submitted by:

Name: Ayan Sarkar
Roll: 1903162
Section: C
Department: CSE

Submitted to:

Md. Mazharul Islam
Lecturer,
Department of CSE,
RUET

Problem: Implement a single-layer perceptron to classify data points based on given features.

Experiment Description: Three different implementations of the perceptron learning algorithm are tested:

- Perception 1: Standard perceptron without a learning rate, using direct weight updates based on misclassified points.
- Perception 2: Perceptron with a learning rate factor to control the magnitude of weight updates.
- Perception 3: Perceptron with weight updates proportional to the classification error rather than a step function.

Each algorithm is run for a fixed number of iterations (800), and accuracy is measured on the training dataset. If the dataset contains only two features, the decision boundary is visualized.

Dataset Characteristic: The dataset used for training and testing is loaded from a CSV file (perceptron_dataset.csv). It consists of two feature columns and a final column representing the class label. Features are numerical, and labels are binary (0 or 1). The dataset is divided into input features (X) and output labels (Y), with the following structure:

- X: Feature columns (numerical values)
- Y: Binary class labels (0 or 1)

Code:

Perceptron 1:

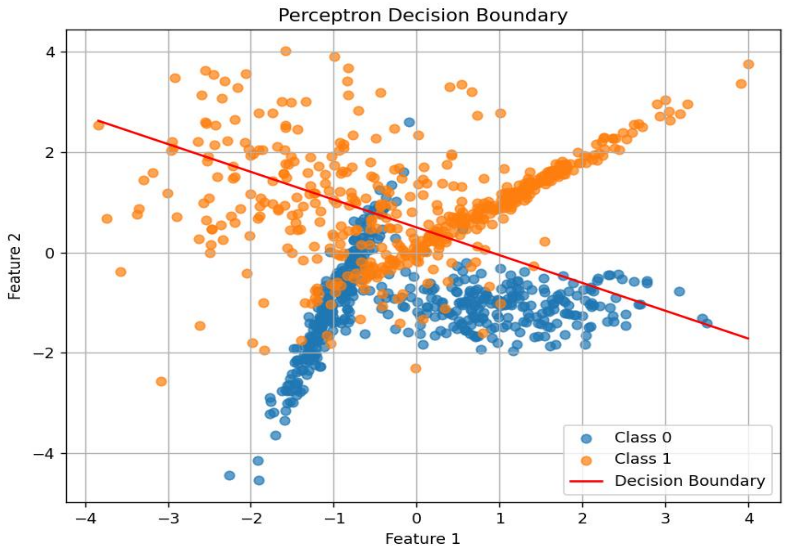
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("Class_2/perceptron_dataset.csv")

X = data.iloc[:, :-1].values
Y = data.iloc[:, -1].values

epoch = 800
weights = np.random.rand(X.shape[1])
thresold = 1
weights = np.insert(weights, 0, -thresold)

```

```python
# Perceptron learning algorithm 1
for iter in range(epoch):
    for i in range(len(X)):
        x = X[i]
        x = np.insert(x, 0, 1)
        weighted_sum = np.dot(x, weights)

        output = 1 if weighted_sum >= 0 else 0

        delta = Y[i] - output

        if Y[i]==0 and output==1:
            weights -= x
        elif Y[i]==1 and output==0:
            weights += x
```

```python
bias = weights[0]
weights = weights[1:]

print("Bias:", end="\t")
print(bias)
print("Weights: ", end=" ")
print(weights)
```

```python
correct_predictions = 0
for i in range(len(X)):
    weighted_sum = np.dot(X[i], weights) + bias
    output = 1 if weighted_sum >= 0 else 0
    if output == Y[i]:
        correct_predictions += 1

accuracy = (correct_predictions / len(X)) * 100
print(f"Accuracy: {accuracy:.2f}%")

```

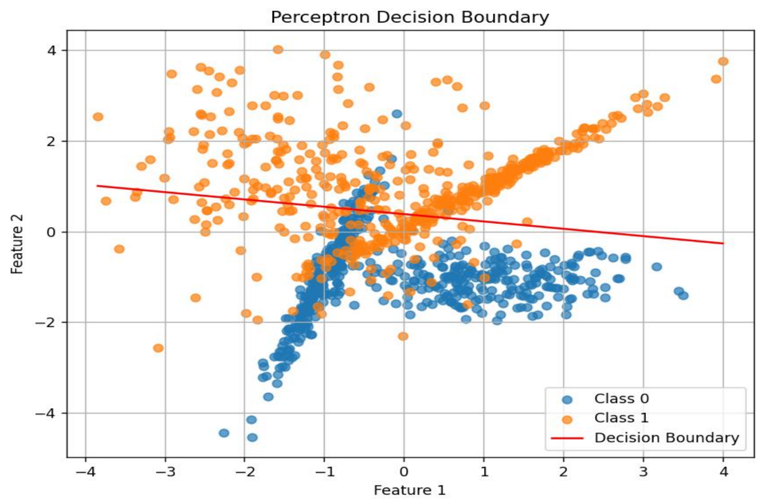
```python
if X.shape[1] == 2:
    plt.figure(figsize=(8, 6))

    for label in np.unique(Y):
        plt.scatter(X[Y == label, 0], X[Y == label, 1], label=f"Class {label}", alpha=0.7)

    x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    if weights[1] != 0:
        y_values = (-weights[0] / weights[1]) * x_values - (bias / weights[1])
        plt.plot(x_values, y_values, color='red', label='Decision Boundary')
    else:
        print("Warning: Cannot plot decision boundary (weights[1] = 0).")
```



Visualization 1:

Perceptron 2:

```python
# Perceptron learning algorithm 2
for iter in range(iteration):
    for i in range(len(X)):
        x = X[i]
        x = np.insert(x, 0, 1);
        weighted_sum = np.dot(x, weights)

        output = 1 if weighted_sum >= 0 else 0

        delta = Y[i] - output

        if Y[i]==0 and output==1:
            weights -= learning_rate*x
        elif Y[i]==1 and output==0:
            weights += learning_rate*x
```

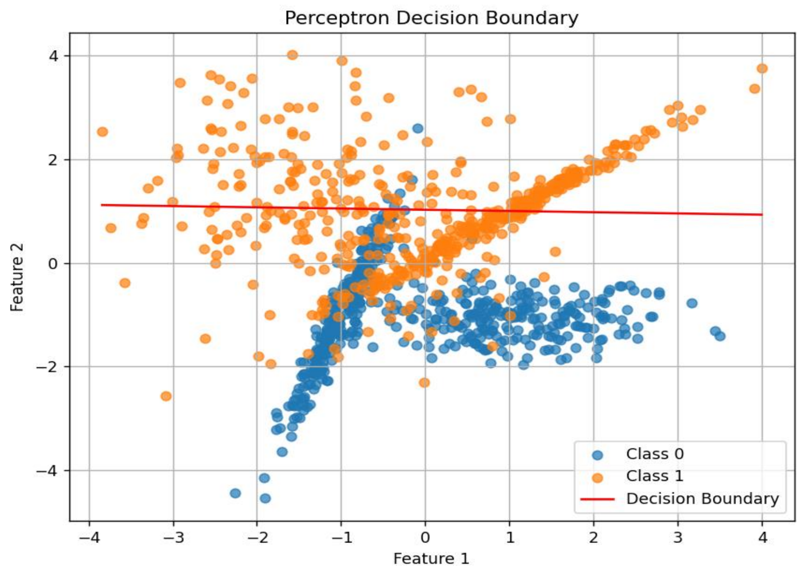


Visualization 2:

Perceptron 3:

```
# Perceptron learning algorithm 3
for iter in range(iteration):
    for i in range(len(X)):
        x = X[i]
        x = np.insert(x, 0, 1);
        weighted_sum = np.dot(x, weights)

        output = 1 if weighted_sum >= 0 else 0

        delta = Y[i] - output

        weights += learning_rate*delta*x
```

Visualization 3:



Result: Each perceptron variation achieves different levels of accuracy:

- **Perceptron 1**: Uses a fixed weight update rule and achieves 78.20% accuracy based on misclassification corrections.

```
Bias:   -1.0
Weights:  [1.08660178 2.05834055]
Accuracy: 78.20%
```

- **Perceptron 2:** Achieves 77.50% accuracy using the learning rate 0.01.

```
Bias:   -0.009999999999999254
Weights:  [0.01411316 0.02373065]
Accuracy: 77.50%
```

- **Perceptron 3**: Adjusts weights based on the classification error magnitude rather than a step function, leading to different learning dynamics achieving 79.90% accuracy.

```
Bias:   -0.10000000000000014
Weights:  [0.09817932 0.2492727 ]
Accuracy: 79.90%
```

Conclusion: The perceptron learning algorithm is effective in solving linearly separable classification problems. Adding a learning rate helps stabilize learning, and using error-proportional weight updates may further refine convergence. The experiment highlights the strengths and limitations of the single-layer perceptron, particularly its inability to classify non-linearly separable data.