Ann5

import numpy as np

import matplotlib.pyplot as plt

np.random.seed(42)

X = np.array([[2, 3], [1, 1], [4, 2], [3, 6], [5, 4], [6, 5]])

y = np.array([1, -1, 1, -1, 1, 1]) # Labels (+1, -1)

class Perceptron:

def \_\_init\_\_(self, learning\_rate=0.1, epochs=10):

self.learning\_rate = learning\_rate

self.epochs = epochs

self.weights = None

self.bias = None

def fit(self, X, y):

num\_samples, num\_features = X.shape

self.weights = np.zeros(num\_features)

self.bias = 0

for \_ in range(self.epochs):

for i in range(num\_samples):

update = self.learning\_rate \* y[i]

if y[i] \* (np.dot(self.weights, X[i]) + self.bias) <= 0:

self.weights += update \* X[i]

self.bias += update

def predict(self, X):

return np.sign(np.dot(X, self.weights) + self.bias)

perceptron = Perceptron(learning\_rate=0.1, epochs=10)

perceptron.fit(X, y)

def plot\_decision\_boundary(X, y, model):

plt.figure(figsize=(6, 5))

# Plot data points

for i, label in enumerate(y):

color = 'blue' if label == 1 else 'red'

marker = 'o' if label == 1 else 's'

plt.scatter(X[i, 0], X[i, 1], color=color, marker=marker, s=100)

# Create decision boundary

x\_min, x\_max = 0, 7

y\_min, y\_max = -1, 7

xx = np.linspace(x\_min, x\_max, 100)

yy = -(model.weights[0] \* xx + model.bias) / model.weights[1]

plt.plot(xx, yy, 'black', linestyle='--', label='Decision Boundary')

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.title("Perceptron Decision Boundary")

plt.legend()

plt.grid()

plt.show()

plot\_decision\_boundary(X, y, perceptron)

test\_points = np.array([[3, 2], [4, 5]])

predictions = perceptron.predict(test\_points)

for point, pred in zip(test\_points, predictions):

print(f"Point {point} classified as: {'Class 1' if pred == 1 else 'Class -1'}")