import numpy as np

import matplotlib.pyplot as plt

class Perceptron:

def \_\_init\_\_(self, learning\_rate=0.01, n\_iterations=1000):

self.learning\_rate = learning\_rate

self.n\_iterations = n\_iterations

self.weights = None

self.bias = None

def fit(self, X, y):

n\_samples, n\_features = X.shape

# Initialize weights and bias

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

self.bias = 0

# Convert labels to {0, 1} format if necessary

y\_ = np.where(y <= 0, -1, 1)

for \_ in range(self.n\_iterations):

for idx, x\_i in enumerate(X):

linear\_output = np.dot(x\_i, self.weights) + self.bias

y\_pred = np.sign(linear\_output)

print(idx,end='')

# Update rule

if y\_[idx] \* y\_pred <= 0:

update = self.learning\_rate \* y\_[idx]

self.weights += update \* x\_i

self.bias += update

def predict(self, X):

linear\_output = np.dot(X, self.weights) + self.bias

return np.where(linear\_output >= 0, 1, 0)

# Manually create linearly separable data

X = np.array([

[2, 3], [1, 1], [2, 1], [3, 3], [2, 2], # Class 0

[6, 5], [7, 7], [8, 6], [7, 5], [9, 7] # Class 1

])

y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])

# Train perceptron

perceptron = Perceptron(learning\_rate=0.1, n\_iterations=10)

perceptron.fit(X, y)

# Plotting decision regions

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

# Set axis boundaries

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

# Create meshgrid

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01),

np.arange(y\_min, y\_max, 0.01))

# Predict on grid points

grid\_points = np.c\_[xx.ravel(), yy.ravel()]

Z = model.predict(grid\_points)

Z = Z.reshape(xx.shape)

# Plot the decision boundary

plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.hot)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolor='k')

plt.title("Perceptron Decision Regions")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

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