## **Державний університет інформаційно-комунікаційних технологій**

**Практична робота №7**

**з дисципліни: Технології штучного інтелекту**

**Тема: Реалізація методу градієнтного спуску в задачах бінарної**

**класифікації**

Виконав:  
студент ДУІКТ  
Тертишний В.Ю.

група: ШІДМ-51

м.Київ

**Мета:** Ознайомитися з методом градієнтного спуску та його застосуванням для розв'язку задач оптимізації та бінарної класифікації. Навчитися застосовувати градієнтний спуск для знаходження мінімуму функції втрат та навчання моделей машинного навчання.

**Код:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import StandardScaler

# 1. Генерація синтетичних даних для прогнозування повернення товару

np.random.seed(42)

# Імітуємо два класи: "Поверне товар" та "Не поверне товар"

time\_steps = 10 # Часові ряди з характеристиками покупок

num\_features = 3 # Додаткові характеристики товару/поведінки

num\_samples = 200 # Кількість зразків

# Дані для класу "Не поверне товар"

no\_return\_data = np.random.normal(0.5, 0.1, (num\_samples // 2, time\_steps))

no\_return\_data = np.c\_[no\_return\_data, np.random.normal(1, 0.5, (num\_samples // 2, num\_features))]

# Дані для класу "Поверне товар"

return\_data = np.random.normal(1.5, 0.2, (num\_samples // 2, time\_steps))

return\_data = np.c\_[return\_data, np.random.normal(2, 0.5, (num\_samples // 2, num\_features))]

# Об'єднуємо дані та створюємо мітки

X = np.vstack([no\_return\_data, return\_data])

y = np.hstack([np.zeros(num\_samples // 2), np.ones(num\_samples // 2)]) # 0 - не поверне, 1 - поверне

# 2. Нормалізація даних

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Розділення на навчальний і тестовий набір

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 3. Логістична регресія (навчання)

weights = np.random.randn(X\_train.shape[1])

bias = 0.0

learning\_rate = 0.01

iterations = 1000

regularization\_param = 0.01

# Функції з оригінального коду

def sigmoid(z):

return 1 / (1 + np.exp(-z))

def compute\_loss(y\_true, y\_pred, weights, reg\_param):

log\_loss = -np.mean(y\_true \* np.log(y\_pred + 1e-15) + (1 - y\_true) \* np.log(1 - y\_pred + 1e-15))

l2\_regularization = reg\_param \* np.sum(weights \*\* 2)

return log\_loss + l2\_regularization

def gradient\_descent(X, y, weights, bias, learning\_rate, iterations, reg\_param):

losses = []

for i in range(iterations):

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

y\_pred = sigmoid(linear\_pred)

dw = np.dot(X.T, (y\_pred - y)) / len(y) + reg\_param \* weights

db = np.sum(y\_pred - y) / len(y)

weights -= learning\_rate \* dw

bias -= learning\_rate \* db

loss = compute\_loss(y, y\_pred, weights, reg\_param)

losses.append(loss)

if i % 100 == 0:

print(f"Iteration {i}, Loss: {loss}")

return weights, bias, losses

# Навчання моделі

weights, bias, losses = gradient\_descent(X\_train, y\_train, weights, bias, learning\_rate, iterations, regularization\_param)

# 4. Оцінка моделі

def predict(X, weights, bias):

return sigmoid(np.dot(X, weights) + bias) >= 0.5

y\_train\_pred = predict(X\_train, weights, bias)

y\_test\_pred = predict(X\_test, weights, bias)

print("\n--- Model Performance ---")

print(f"Train Accuracy: {accuracy\_score(y\_train, y\_train\_pred)}")

print(f"Test Accuracy: {accuracy\_score(y\_test, y\_test\_pred)}")

print("\nClassification Report on Test Set:")

print(classification\_report(y\_test, y\_test\_pred))

# 5. Графік втрат

plt.figure(figsize=(8, 4))

plt.plot(losses, label="Train Loss")

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.title("Training Loss Over Iterations")

plt.legend()

plt.grid()

plt.show()

# 6. Візуалізація класифікації (2D проєкція для демонстрації)

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

X\_train\_2d = pca.fit\_transform(X\_train)

X\_test\_2d = pca.transform(X\_test)

plt.figure(figsize=(8, 6))

plt.scatter(X\_train\_2d[y\_train == 0][:, 0], X\_train\_2d[y\_train == 0][:, 1], color='blue', label='No Return (Train)')

plt.scatter(X\_train\_2d[y\_train == 1][:, 0], X\_train\_2d[y\_train == 1][:, 1], color='red', label='Return (Train)')

plt.scatter(X\_test\_2d[y\_test == 0][:, 0], X\_test\_2d[y\_test == 0][:, 1], color='cyan', label='No Return (Test)', marker='x')

plt.scatter(X\_test\_2d[y\_test == 1][:, 0], X\_test\_2d[y\_test == 1][:, 1], color='orange', label='Return (Test)', marker='x')

plt.title("2D Projection of Data with PCA")

plt.legend()

plt.grid()

plt.show()

**Робота коду:**  
(**>Скріни сюди<**)

**Висновки**

Метод градієнтного спуску є одним із фундаментальних підходів для розв'язання задач оптимізації, зокрема у сфері машинного навчання. Він дозволяє ефективно знаходити мінімум функції втрат, поступово коригуючи параметри моделі на основі похідної функції втрат за кожним параметром. У контексті задачі бінарної класифікації метод градієнтного спуску застосовується для навчання моделі шляхом мінімізації логістичної функції втрат, яка враховує як точність прогнозів, так і можливу переобучуваність завдяки регуляризації. В процесі реалізації було продемонстровано, як цей метод працює для навчання моделі, що класифікує дані на дві категорії: "аварія можлива" та "аварія малоймовірна". Навчання включало обчислення похибки, оновлення вагових коефіцієнтів і побудову графіків, які ілюструють зменшення втрат з кожною ітерацією. Візуалізація результатів та аналіз точності на навчальних і тестових вибірках підтвердили, що градієнтний спуск є потужним і гнучким інструментом, придатним для розв'язання широкого спектра задач оптимізації в машинному навчанні.