import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from google.colab import files

# 1. 上传文件到 Colab

uploaded = files.upload() # 手动选择 train.xlsx 和 test.xlsx

# 2. 读取数据

train\_df = pd.read\_excel('train.xlsx') # 文件名需与上传的文件名一致

test\_df = pd.read\_excel('test.xlsx')

X\_train = train\_df['x'].values.reshape(-1, 1)

y\_train = train\_df['y'].values

X\_test = test\_df['x'].values.reshape(-1, 1)

y\_test = test\_df['y'].values

# 为特征矩阵添加一列1（用于截距项）

def add\_ones(X):

return np.hstack([np.ones((X.shape[0], 1)), X])

X\_train\_design = add\_ones(X\_train)

X\_test\_design = add\_ones(X\_test)

# --- 最小二乘法 ---

theta\_ols = np.linalg.inv(X\_train\_design.T @ X\_train\_design) @ X\_train\_design.T @ y\_train

# --- 梯度下降法 ---

def gradient\_descent(X, y, alpha=0.01, iterations=1000):

m = X.shape[0]

theta = np.zeros(X.shape[1])

for \_ in range(iterations):

y\_pred = X @ theta

gradient = (X.T @ (y\_pred - y)) / m

theta -= alpha \* gradient

return theta

theta\_gd = gradient\_descent(X\_train\_design, y\_train)

# --- 牛顿法 ---

def newton\_method(X, y):

m = X.shape[0]

theta = np.zeros(X.shape[1])

gradient = (X.T @ (X @ theta - y)) / m

H = (X.T @ X) / m

theta -= np.linalg.inv(H) @ gradient # 一次迭代即收敛

return theta

theta\_newton = newton\_method(X\_train\_design, y\_train)

# 计算均方误差

def compute\_mse(theta, X, y):

y\_pred = X @ theta

return np.mean((y\_pred - y) \*\* 2)

# 训练误差

train\_errors = {

'OLS': compute\_mse(theta\_ols, X\_train\_design, y\_train),

'GD': compute\_mse(theta\_gd, X\_train\_design, y\_train),

'Newton': compute\_mse(theta\_newton, X\_train\_design, y\_train)

}

# 测试误差

test\_errors = {

'OLS': compute\_mse(theta\_ols, X\_test\_design, y\_test),

'GD': compute\_mse(theta\_gd, X\_test\_design, y\_test),

'Newton': compute\_mse(theta\_newton, X\_test\_design, y\_test)

}

print("训练误差:")

for method, error in train\_errors.items():

print(f"{method}: {error:.4f}")

print("\n测试误差:")

for method, error in test\_errors.items():

print(f"{method}: {error:.4f}")

# 可视化测试结果

sorted\_indices = np.argsort(X\_test.flatten())

X\_test\_sorted = X\_test[sorted\_indices]

y\_test\_sorted = y\_test[sorted\_indices]

# 预测值

y\_pred\_ols = X\_test\_design @ theta\_ols

y\_pred\_gd = X\_test\_design @ theta\_gd

y\_pred\_newton = X\_test\_design @ theta\_newton

plt.figure(figsize=(12, 7))

# 绘制训练数据

plt.scatter(X\_train, y\_train, color='blue', alpha=0.7, label='Train Data')

# 绘制测试数据

plt.scatter(X\_test, y\_test, color='red', alpha=0.7, label='Test Data')

# 绘制预测曲线

plt.plot(X\_test\_sorted, y\_pred\_ols[sorted\_indices], 'r', label='OLS Predictions', linewidth=2)

plt.plot(X\_test\_sorted, y\_pred\_gd[sorted\_indices], 'g--', label='GD Predictions', linewidth=2)

plt.plot(X\_test\_sorted, y\_pred\_newton[sorted\_indices], 'b:', label='Newton Predictions', linewidth=2)

plt.xlabel('x')

plt.ylabel('y')

plt.title('Linear Regression: Train vs Test Data with Predictions')

plt.legend(loc='best')

plt.grid(True)

plt.show()