|  |  |
| --- | --- |
|  | import numpy as np |
|  | import pandas as pd |
|  | import matplotlib.pyplot as plt |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from sklearn.preprocessing import StandardScaler |
|  | from sklearn.metrics import mean\_squared\_error |
|  |  |
|  | # 读取数据 |
|  | path = 'user/杨文文/机器学习作业/regress\_data1.csv' |
|  | data = pd.read\_csv(path) |
|  | X = data.iloc[:, :-1].values |
|  | y = data.iloc[:, -1].values |
|  |  |
|  | # 数据分割为训练集和测试集 |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |
|  |  |
|  | # 数据归一化 |
|  | scaler = StandardScaler() |
|  | X\_train\_scaled = scaler.fit\_transform(X\_train) |
|  | X\_test\_scaled = scaler.transform(X\_test) |
|  |  |
|  | # 添加偏置项 |
|  | X\_train\_scaled = np.concatenate([np.ones((X\_train\_scaled.shape[0], 1)), X\_train\_scaled], axis=1) |
|  | X\_test\_scaled = np.concatenate([np.ones((X\_test\_scaled.shape[0], 1)), X\_test\_scaled], axis=1) |
|  |  |
|  | # 设置参数 |
|  | alpha = 0.01 # 学习率 |
|  | iters = 1000 # 迭代次数 |
|  | lambda\_ = 0.01 # 正则化参数 |
|  |  |
|  | # 梯度下降法实现线性回归 |
|  | def gradient\_descent(X, y, W, alpha, lambda\_, iters): |
|  | m = len(y) |
|  | cost\_history = np.zeros(iters) |
|  | for it in range(iters): |
|  | y\_pred = np.dot(X, W) |
|  | error = y\_pred - y |
|  | gradient = (1/m) \* np.dot(X.T, error) + (lambda\_/m) \* W |
|  | W = W - alpha \* gradient |
|  | cost = compute\_cost(X, y, W, lambda\_) |
|  | cost\_history[it] = cost |
|  | return W, cost\_history |
|  |  |
|  | # 计算损失函数 |
|  | def compute\_cost(X, y, W, lambda\_): |
|  | m = len(y) |
|  | y\_pred = np.dot(X, W) |
|  | cost = (1/(2\*m)) \* np.sum(np.square(y\_pred - y)) + (lambda\_/(2\*m)) \* np.sum(np.square(W[1:])) |
|  | return cost |
|  |  |
|  | # 初始化权重 |
|  | W = np.zeros(X\_train\_scaled.shape[1]) |
|  |  |
|  | # 使用梯度下降法训练模型 |
|  | W\_gd, cost\_history\_gd = gradient\_descent(X\_train\_scaled, y\_train, W, alpha, lambda\_, iters) |
|  |  |
|  | # 使用最小二乘法求解线性回归 |
|  | def normal\_equation(X, y, lambda\_): |
|  | theta = np.linalg.inv(np.dot(X.T, X) + lambda\_ \* np.identity(X.shape[1])) \ |
|  | .dot(X.T).dot(y) |
|  | return theta |
|  |  |
|  | theta\_ne = normal\_equation(X\_train\_scaled, y\_train, lambda\_) |
|  |  |
|  | # 绘制训练和测试损失曲线 |
|  | def plot\_loss\_curve(train\_cost\_history, test\_cost\_history): |
|  | plt.figure(figsize=(10, 5)) |
|  | plt.plot(range(iters), train\_cost\_history, label='Train Loss') |
|  | plt.plot(range(iters), test\_cost\_history, label='Test Loss') |
|  | plt.xlabel('Iteration') |
|  | plt.ylabel('Loss') |
|  | plt.title('Loss Curve') |
|  | plt.legend() |
|  | plt.show() |
|  |  |
|  | # 计算测试集上的损失 |
|  | y\_pred\_gd\_train = np.dot(X\_train\_scaled, W\_gd) |
|  | y\_pred\_gd\_test = np.dot(X\_test\_scaled, W\_gd) |
|  | train\_mse\_gd = mean\_squared\_error(y\_train, y\_pred\_gd\_train) |
|  | test |