实验一：线性模型

***Steven***

# 一、线性回归模型

本任务中你将使用一元线性回归来预测食厅的利润。假设你是一家特许餐厅的首席执行官，正在考虑在不同的城市开设一家新的分店。该连锁店已经在不同的城市有分店，你有这些城市的利润和人口数据。你希望使用这些数据来帮助您选择下一个要扩展到的城市。

文件 ex1data.csv 包含我们的线性回归问题的数据集。Population 代表一个城市的人口，profit 代表此个城市的餐厅利润。利润的负值表示亏损。

请将70%的数据用作训练集，30%的数据用作测试集，使用留出法对以上模型进行验证。

## **1. 题目分析**

本题是一个较为简单的一元线性回归问题，课本P54已推导出了该问题最优解参数的求解公式：

于是本题的求解过程便简单许多，只需要使用上述公式计算参数即可得到线性回归模型。

## **2. 解题步骤**

(1) 读取数据，绘制散点图。

(2) 使用留出法划分训练集和测试集。

(3) 使用上述公式计算参数值。

(4) 绘制散点图和线性回归方程。

(5) 评估模型，性能度量使用均方误差。

## **3. 结果分析**

图表, 散点图

描述已自动生成

(图1.1 原始数据散点图)

图表, 散点图

描述已自动生成

(图1.2 原始数据和线性回归模型图)

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**(图1.3 线性回归模型在训练集和测试集上的均方误差)**

# 二、逻辑回归模型

本任务中你将建立一个逻辑回归模型来预测一个学生是否被大学录取。假设你是一所大学系的管理员，你想根据两次考试的成绩来决定每个申请人的录取机会。你有以前申请者的历史数据，可以用作逻辑回归的训练集。对于每个培训示例，你都有申请人在两次考试中的分数和录取决定。你的任务是建立一个分类模型，根据这两次考试的分数来估计申请人的录取概率。

文件 ex1data2.csv 包含我们的逻辑回归问题的数据集,学生的两门成绩 Exam1，Exam2 和是否被录取 Accepted(1 为录取，0 为未录取)，请利用逻辑回归知识根据学生成绩判断学生是否会被录取，并将结果可视化。

请使用 5 折交叉验证法对以上模型进行验证。

## 1. 题目分析

本题是一个逻辑回归问题，属二分类问题。根据课本P59以及课本7.2的极大似然法思想，可知该类问题最优解等价于最小化以下问题：

其中，该问题可以通过梯度下降法进行求解。

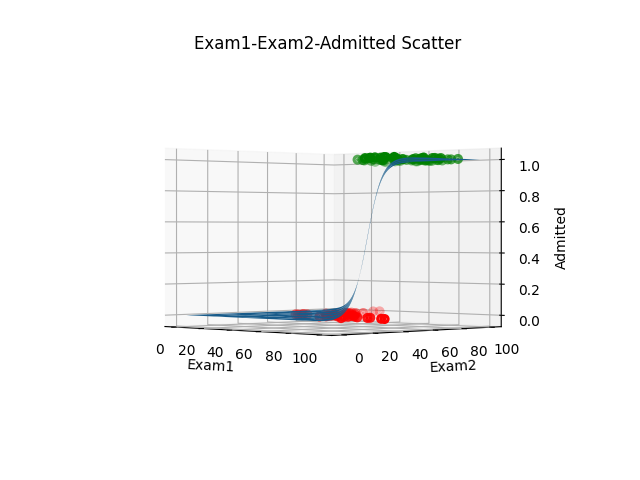
## 2. 解题步骤

(1) 读取数据。

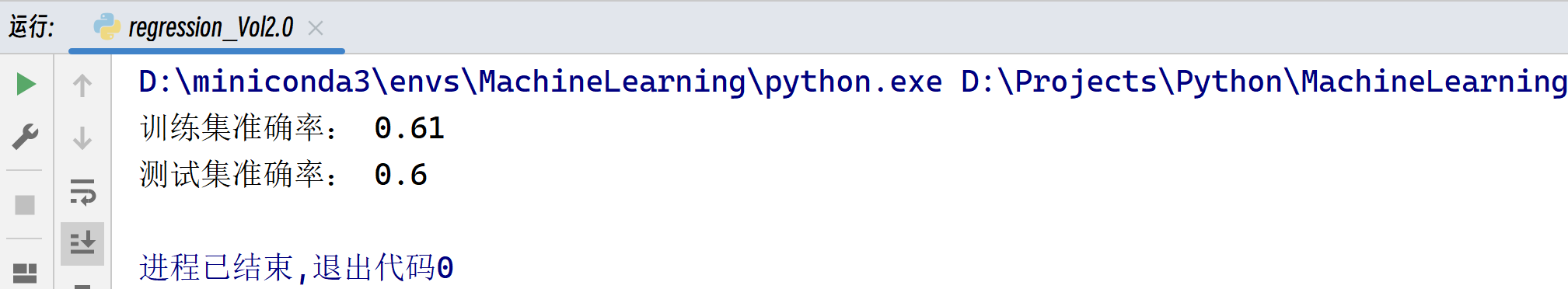
(2) 使用梯度下降法求解上述等价问题。

(3) 执行五折交叉验证，使用五次验证的均值作为结果。

## **3. 结果分析**



**(图2.1数据散点与逻辑回归模型图(三维))**



(图2.2 五折交叉验证的结果)

# 三、代码

|  |
| --- |
| **import** matplotlib **import** numpy **as** np **from** matplotlib **import** pyplot **as** plt **from** pandas **import** read\_csv **from** sklearn.model\_selection **import** train\_test\_split, KFold  matplotlib.use(**'TkAgg'**) *# 设定绘图后端* **def** calc\_liner\_MSE(x: np.ndarray, y: np.ndarray, omega: float, b: float) -> float:  *"""  计算线性回归的均方误差* **:param** *x: 数据集X* **:param** *y: 数据集Y* **:param** *omega: 线性回归函数 斜率* **:param** *b: 线性回归函数 截距* **:return***: 均方误差  """* **return** np.sum(np.power(y - (omega \* x + b), 2)) / len(x) *# 为忽略数据集大小造成的影响，这里额外除以len(xData)* **def** calc\_liner\_param(x: np.array, y: np.array) -> tuple[float, float]:  *"""  计算线性回归的参数* **:param** *x: 训练集X(numpy.ndarray)* **:param** *y: 训练集Y(numpy.ndarray)* **:return***: 参数omega和b(线性回归函数中的斜率和截距)  """  # 计算参数  # 根据omega和b的最优解的闭式解，直接计算参数，跳过梯度下降的步骤* x\_avg, y\_avg = np.average(x), np.average(y)  omega = np.sum((x - x\_avg) \* y) / (np.sum(np.power(x, 2)) - np.power(np.sum(x), 2) / len(x))  b = y\_avg - omega \* x\_avg  **return** omega, b   **def** liner\_regression():  *"""  第一题 线性回归* **:return***:None  """  # 初始化数据* df = read\_csv(**'./ex1data1.csv'**)  population, profit = df[**'population'**], df[**'profit'**]  *# 绘制原始数据的散点图* canvas1 = plt  canvas1.scatter(x=population, y=profit, marker=**'.'**, color=**'b'**, label=**'Population-Profit'**, linewidths=3)  canvas1.legend()  canvas1.xlabel(**'Population'**)  canvas1.ylabel(**'Profit'**)  canvas1.title(**'Population-Profit'**)  canvas1.show()  *# 计算线性回归参数* x\_train, x\_test, y\_train, y\_test = train\_test\_split(population, profit, test\_size=0.3)  omega, b = calc\_liner\_param(x\_train, y\_train) *# 计算线性回归参数  # 绘制散点和线性回归函数* canvas2 = plt  canvas2.scatter(x=population, y=profit, marker=**'.'**, color=**'b'**, label=**'Population-Profit'**, linewidths=3)  canvas2.plot(population, omega \* population + b, color=**'r'**, label=**r'$y=%.2fx%.2f$'** % (omega, b))  canvas2.legend()  canvas2.xlabel(**'Population'**)  canvas2.ylabel(**'Profit'**)  canvas2.title(**'Population-Profit(with linearRegression)'**)  canvas2.show()  print(**"训练集均方误差："**, calc\_liner\_MSE(x\_train, y\_train, omega, b))  print(**"测试集均方误差："**, calc\_liner\_MSE(x\_test, y\_test, omega, b))   **def** logistic(z):  **return** 1 / (1 + np.exp(-z))   **def** calc\_logistic\_param(x: np.ndarray, y: np.ndarray) -> np.array:  *"""  计算逻辑回归的参数* **:param** *x: 训练集X* **:param** *y: 训练集Y* **:return***:  """* **def** H(x\_hat, theta):  **return** logistic(np.dot(x\_hat, theta))   **def** J(x\_hat, y\_hat, theta):  **return** np.sum(-y\_hat \* np.log(H(x\_hat, theta)) - (1 - y\_hat) \* np.log(1 - H(x\_hat, theta))) / len(x\_hat)   **def** gradJ(x\_hat, y\_hat, theta):  **return** np.dot(x\_hat.T, H(x\_hat, theta) - y\_hat) / len(x\_hat)   *# 初始化参数* x\_hat = np.hstack((np.ones((x.shape[0], 1)), x))  theta = np.zeros((x\_hat.shape[1], 1))  alpha = 0.01  max\_iter = 1000  *# 梯度下降* **for** i **in** range(max\_iter):  theta -= alpha \* gradJ(x\_hat, y, theta)  **return** theta   **def** calc\_logistic\_accuracy(x\_train, y\_train, b, omega1, omega2):  *"""  计算逻辑回归的准确率* **:param** *x\_train: 训练集X* **:param** *y\_train: 训练集Y* **:param** *b: 截距* **:param** *omega1: 参数1* **:param** *omega2: 参数2* **:return***:  """* x\_hat = np.hstack((np.ones((x\_train.shape[0], 1)), x\_train))  theta = np.array([[b], [omega1], [omega2]])  y\_pred = logistic(np.dot(x\_hat, theta))  y\_pred = np.where(y\_pred >= 0.5, 1, 0)  **return** np.sum(y\_pred == y\_train) / len(y\_train)   **def** logistic\_regression():  *"""  第二题 逻辑回归* **:return***: None  """   # 初始化数据* df = read\_csv(**'./ex1data2.csv'**)  exam1, exam2, accepted = df[**'Exam1'**], df[**'Exam2'**], df[**'Accepted'**]  accepted = np.matrix(accepted).T  x = np.stack((exam1, exam2), axis=1) *# 将二维的exam数据组合起来  # 使用梯度下降法计算参数* param = calc\_logistic\_param(x, accepted)  b, omega1, omega2 = param[0][0], param[1][0], param[2][0]  *# 绘制散点和Logistic函数* x\_range, y\_range = np.meshgrid(np.linspace(0, 100, 100), np.linspace(0, 100, 100))  accepted\_pred = logistic(omega1 \* x\_range + omega2 \* y\_range + b)  canvas = plt  axis3 = canvas.axes(projection=**'3d'**)  axis3.scatter([exam1[i] **for** i **in** range(len(accepted)) **if** accepted[i] == 1],  [exam2[i] **for** i **in** range(len(accepted)) **if** accepted[i] == 1],  [1 **for** i **in** range(len(accepted)) **if** accepted[i] == 1],  marker=**'.'**, label=**'Admitted'**, color=**'green'**, linewidth=3) *# 画出被录取的散点* axis3.scatter([exam1[i] **for** i **in** range(len(accepted)) **if** accepted[i] == 0],  [exam2[i] **for** i **in** range(len(accepted)) **if** accepted[i] == 0],  [0 **for** i **in** range(len(accepted)) **if** accepted[i] == 0],  marker=**'.'**, label=**'Not Admitted'**, color=**'red'**, linewidth=3) *# 画出未被录取的散点* axis3.plot\_surface(x\_range, y\_range, accepted\_pred, label=**'Logistic Regression'**, color=**'blue'**, alpha=0.5)  axis3.set\_xlabel(**'Exam1'**)  axis3.set\_ylabel(**'Exam2'**)  axis3.set\_zlabel(**'Admitted'**)  canvas.title(**'Exam1-Exam2-Admitted'**)  canvas.show()  *# 五折交叉验证* kf = KFold(n\_splits=5, shuffle=**True**)  accuracy\_train, accuracy\_test = [], []  **for** train\_index, test\_index **in** kf.split(x):  x\_train, x\_test = x[train\_index], x[test\_index]  y\_train, y\_test = accepted[train\_index], accepted[test\_index]  param = calc\_logistic\_param(x\_train, y\_train)  b, omega1, omega2 = param[0][0], param[1][0], param[2][0]  accuracy\_train.append(calc\_logistic\_accuracy(x\_train, y\_train, b, omega1, omega2))  accuracy\_test.append(calc\_logistic\_accuracy(x\_test, y\_test, b, omega1, omega2))  print(**"训练集准确率："**, np.mean(accuracy\_train))  print(**"测试集准确率："**, np.mean(accuracy\_test))   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  liner\_regression()  logistic\_regression() |