



Python程序设计（混合式）



杨尚东

南京邮电大学计算机学院，数据科学与工程系

shangdongyang.github.io

2023/11/4

目录

- 线性回归问题简介
- 单变量线性回归问题
- 基于Scikit-learn库求解单变量线性回归
- 自定义求解单变量线性回归
 - 基于最小二乘法
 - 基于梯度下降法
- 多变量线性回归问题

人工智能 vs 机器学习

□ 符号主义 (Symbolism) :

- ✓ 一种基于逻辑推理的智能模拟方法
- ✓ 又称逻辑主义、心理学派或计算机学派。
- ✓ 启发式算法→专家系统→知识工程，知识图谱

□ 连接主义 (Connectionism) :

- ✓ 又称仿生学派或生理学派
- ✓ 是一种基于神经网络及网络间的连接机制与学习算法的智能模拟方法，比如深度学习
- ✓ 从历史经验中去学习

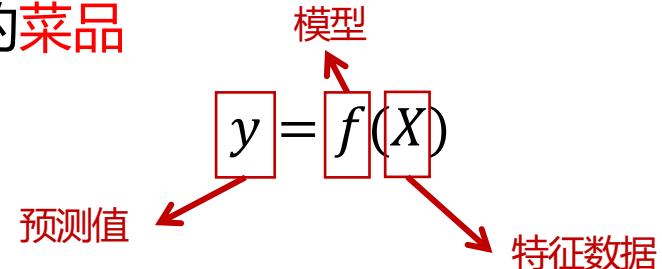
□ 行为主义 (Actionism) :

- ✓ 强化学习：“感知——行动”的行为智能模拟方法

机器学习

□ 机器学习 “预测” 场景

- ✓ 根据细风和晚霞预测明天的气温和天气
- ✓ 根据餐馆的座位情况和香味预测这家店的菜品
- ✓ 根据地铁、学区、居室情况估算房价



□ 机器学习

- ✓ 让计算机模仿人类，从过去经验中学习一个“模型”，通过学到的模型再对新情况给出一个预测
- ✓ “经验”通常是以“数据”的形式存在

□ 监督学习

- ✓ 分类 (classification)：预测的值是“离散”的
- ✓ 回归 (regression)：预测的值是“连续”的

问题定义

□ 数据集

□ 训练数据集(X_{train}, y_{train}) 特征 feature

样本 instance



标签 label



x_1	x_2	x_3	x_4	x_5	...	y
1.2	100	200	3	12		5.2
3.5	200	35	12	4.6		7.8
4.5	7.2	100	5	13.5		9.2
...

□ 测试数据集 X_{test}

x_1	x_2	x_3	x_4	x_5	...	y
1.8	200	20	4	13		?
13.5	300	15	11	4.8		?
14.5	52	2	3	12.5		?

□ 目标

- ✓ 从训练数据集中学习模型 f (**如何形式化 f**)
- ✓ 使得在测试数据集中 $f(X_{test})$ 和真实的 y_{test} 尽量接近 (**如何定义接近**)

线性回归问题与模型

□ 基本形式

- ✓ 给定有 d 个属性的样例 $x = (x_1; x_2; x_3; \dots; x_d)$, 其中 x_i 是 x 在第 i 个属性上的取值, 线性回归试图学习得到一个预测函数 $f(x)$, 该函数的值是各属性值的线性加权和, 公式如下

$$f(x) = w_1x_1 + w_2x_2 + \dots + w_dx_d + b$$

□ 向量形式为: $f(x) = w^T x + b$

- ✓ w 和 b 是**可学习和调整的参数**, 可根据经验手动设定, 或自动从数据中学习获得
- ✓ 预测某套商品房的总价

$$f(x) = 3 \times \text{面积大小} + 0.5 \times \text{楼层指数} + 0.2 \times \text{卧室数量指数}$$

□ 单变量线性回归与多变量线性回归

单变量线性回归问题

- 当样本仅1个属性时（即只有 x_1 ），只要求解两个参数（ w_1 和 b ），是
单变量线性回归模型

$$f(x) = w_1 x_1 + b$$

- 案例描述：设某小区通过某房产中介处已售出5套房，房屋总价与房屋面积之间有如下的数据关系。现有该小区的一位业主想要通过该房产中介出售房屋，在业主报出房屋面积后，根据训练数据，中介能否能估算出该房屋的合适挂售价格？

训练样本	房屋面积 (平方米)	房屋总价 (万元)
1	75	270
2	87	280
3	105	295
4	110	310
5	120	335

案例分析

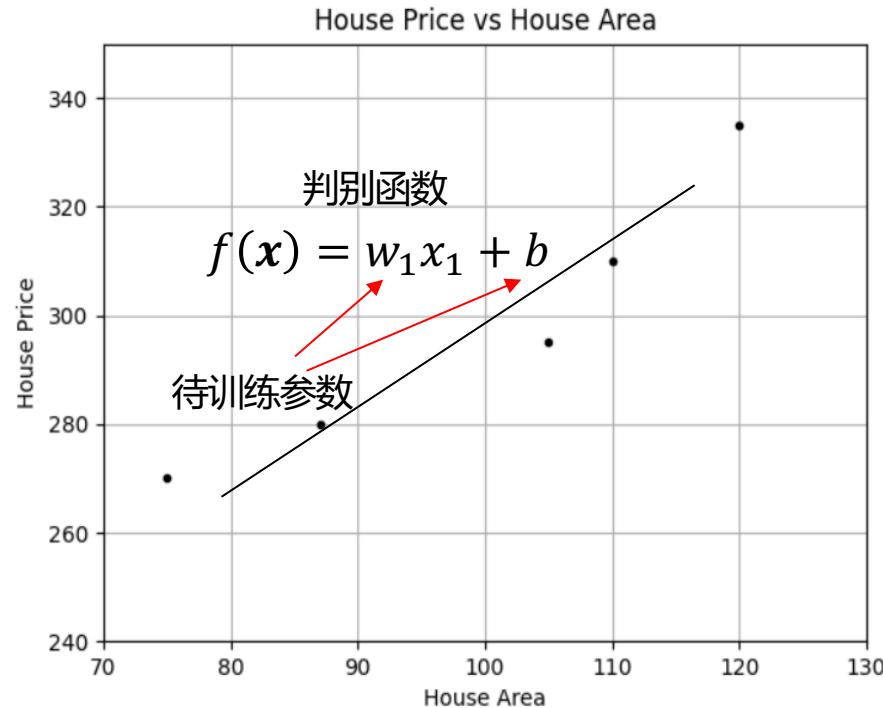
□ 把房屋面积看成自变量 x , 房屋总价看成因变量 y , 先通过绘图看出二者之间的关系

```
#代码3.1 查看小区已售房屋房价样本数据
```

```
import numpy as np  
import matplotlib.pyplot as plt
```

1. 房屋总价随着房屋面积的变化，大致呈现线性变化趋势；
2. 如果根据现有的训练样本数据能够拟合出一条直线，使之与各训练样本数据点都比较接近，那么根据该直线，就可以计算出任意房屋面积的房屋总价了。

```
return plt  
plt = initPlot()  
xTrain = np.array([75, 87, 105, 110, 120])  
yTrain = np.array([270, 280, 295, 310, 335])  
plt.plot(xTrain, yTrain, 'k.') #k表示绘制颜色为黑色，点表示绘制散点图  
plt.show()
```



思考

□ 通过以下两个特征是否可以用LinearRegression实现房价预测

- ✓ 房子长度
- ✓ 房子宽度

模型拟合

- 使得在测试数据集中 $f(X_{test})$ 和真实的 y_{test} 尽量接近
- 悲观的是，我们并不知道 y_{test}
- 退而求其次，希望在训练数据集中让 $f(X_{train})$ 和真实的 y_{train} 尽量接近
 - ✓ 在测试数据集中有泛化性 (generalization) 吗？
 - ✓ 如何定义接近？
- 对于回归问题，可以定义不同损失函数 (loss function)
 - ✓ 均方误差 (mean squared error)

$$mse = \sum_{i=1}^N (f(X_i) - y_i)^2$$

LinearRegression类

- Scikit-learn提供了sklearn.linear_model.LinearRegression线性回归类，可以解决大部分常见的线性回归操作
- 构造方法
- model = LinearRegression(**fit_intercept** = True, **normalize** = False, **copy_X** = True, **n_jobs** = 1)
 - ✓ **fit_intercept**: 是否计算模型的截距，默认值是True，为False时则进行数据中心化处理
 - ✓ **normalize**: 是否归一化，默认值是False
 - ✓ **copy_X**: 默认True，在X.copy()上进行操作，否则会在原始数据X上进行操作，覆盖原始数据
 - ✓ **n_jobs**: 表示使用CPU的个数，默认1，当-1时，代表使用全部的CPU

LinearRegression类

□ LinearRegression类的属性和方法

- ✓ `coef_`: 训练后的输入端模型系数, 如果label有两个, 即y值有两列, 是一个2D的数组
- ✓ `intercept_`: 截距, 即公式 $f(x) = w_1x_1 + w_0$ 中 w_0 的值
- ✓ `fit(x, y)`: 拟合函数, 通过训练数据x和训练数据的标签y来拟合模型
- ✓ `predict(x)`: 预测函数, 通过拟合好的模型, 对数据x预测y值
- ✓ `score`: 评价分数值, 用于评价模型好坏

下面介绍使用 `LinearRegression` 类解决关于房价预测的单变量线性回归问题, 先分析求解步骤, 再提供完整代码并绘图显示拟合结果

求解步骤

□ 第一步：准备训练数据

- ✓ `X_train = np.array([[75], [87], [105], [110], [120]])`
- ✓ `y_train = np.array([270, 280, 295, 310, 335])`

□ 第二步：创建模型对象

- ✓ `model = LinearRegression()`

□ 第三步：执行拟合

- ✓ `model.fit(X_train, y_train)`

□ 第四步：准备测试数据

- ✓ `X_test = np.array([[82], [104]])`
- ✓ `model.predict(X_test)`

代码3.2

```
#代码3.2 基于Scikit-learn实现房价预测线性规划代码

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
# 以矩阵形式表达(对于单变量, 矩阵就是列向量形式)
xTrain = np.array([[75], [87], [105], [110], [120]])
yTrain = np.array([ 270, 280, 295, 310, 335])
model = LinearRegression() # 创建模型对象
model.fit(xTrain, yTrain) # 根据训练数据拟合出直线(以得到假设函数)

print("截距b或w0=", model.intercept_) # 截距
print("斜率w1=", model.coef_) # 斜率
# 预测面积为70的房源的总价
print("预测面积为70的房源的总价: ", model.predict([[70]]))
# 也可以批量预测多个房源, 注意要以列向量形式表达
xTest = np.array([85, 90, 93, 109])[:, np.newaxis]
yTestPredicted = model.predict(xTest)
print("新房源数据的面积: ", xTest)
print("预测新房源数据的总价: ", yTestPredicted)
```

```
def initPlot():
    plt.figure()
    plt.title('House Price vs House Area')
    plt.xlabel('House Area')
    plt.ylabel('House Price')
    # 设置x轴和y轴的值域分别为70~130和240~350
    plt.axis([70, 130, 240, 350])
    plt.grid(True)
    return plt

plt = initPlot()
plt.plot(xTrain, 'k.') # 格式字符串'k.'，表示绘制黑色的散点
# 画出蓝色的拟合线
plt.plot([[70], [130]], model.predict([[70], [130]]), 'b-')
plt.show()
```

代码3.2

截距b或w0= 163.75113877922868

斜率w1= [1.35059217]

预测面积为70的房源的总价: [258.29259034]

新房源数据的面积: [[85]

[90]

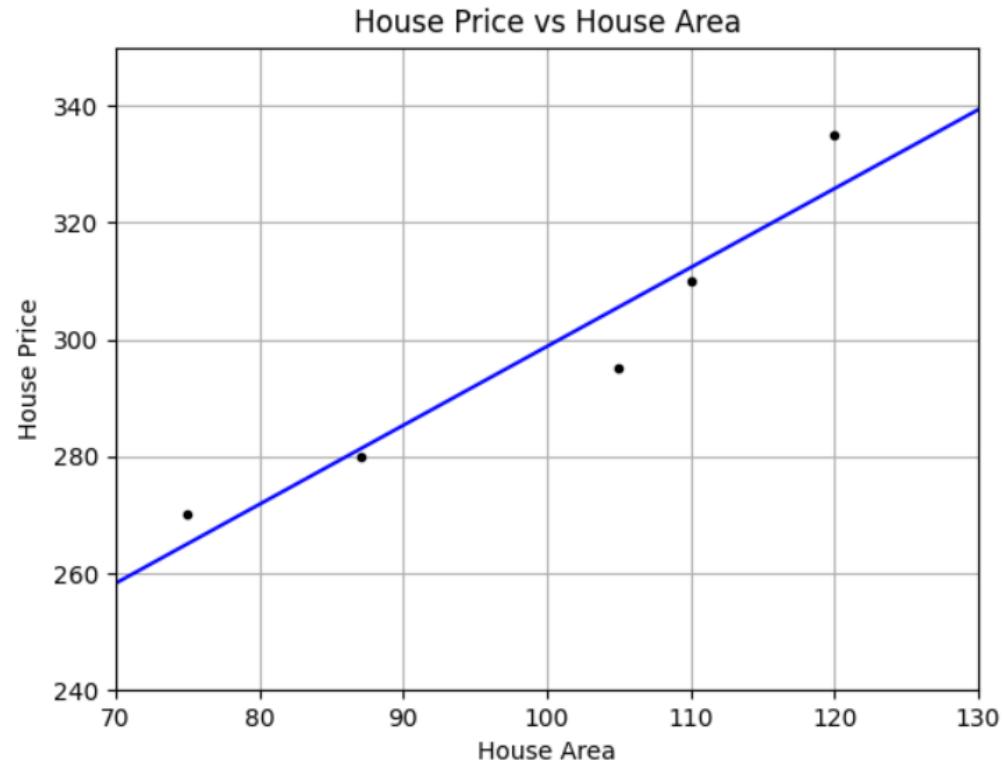
[93]

[109]]

预测新房源数据的总价:

[278.55147282 285.30443365 289.35621014

310.96568479]



求解步骤

□ 第一步：准备训练数据

- ✓ `X_train = np.array([[75], [87], [105], [110], [120]])`
- ✓ `y_train = np.array([270, 280, 295, 310, 335])`

□ 第二步：创建模型对象

- ✓ `model = LinearRegression ()`

□ 第三步：执行拟合

- ✓ `model.fit(X_train, y_train)`

问题2：如何优化模型？

□ 第四步：准备测试数据

- ✓ `X_test = np.array([[82], [104]])`
- ✓ `model.predict (X_test)`

问题1：模型结果是否准确？

模型评价

□ 如何评价该模型的好坏：

✓ 用什么**数据**对拟合模型进行评价？

…> 用训练数据计算的模型误差称之为训练误差

…> 用测试数据计算的模型误差称之为测试误差

…> 在训练过程中，只有训练数据是可见的

…> 训练误差的最小化，不一定能使得测试误差最小化

✓ 用什么**指标**对拟合模型进行评价？

…> 计算线性回归误差的指标主要包括**残差平方和与R方** (r^2)

残差

□ 残差 (Residual)

- ✓ 观测值 y 与通过模型判别函数计算的 y 值之间的差异
- ✓ 残差平方和是训练数据或者测试数据中所有样本残差的平方和
- ✓ 残差值越小则对应数据的拟合度越好，当残差为0时，预测 y 值与观测值 y 完全一致

$$SS_{res} = \sum_{i=1}^m (f(x_i) - y_i)^2$$



R方

□ R方 (R-Square) , 又称决定系数 (coefficient of determination)

- ✓ 表达因变量与自变量之间的总体关系
- ✓ 反映了因变量 y (标签) 的波动, 有多少百分比能被自变量 x (特征) 的波动所描述
- ✓ 与残差平方和在方差中所占的比率有关

$$SS_{res} = \sum_{i=1}^m (f(x_i) - y_i)^2 \quad SS_{total} = \sum_{i=1}^m (\bar{y} - y_i)^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{total}}$$

□ R方为负、0、1分别代表什么?

残差与R方的计算

口 手动计算：

- ✓ 训练数据残差平方和：

…> ssResTrain = sum((yTrainPredicted - yTrain) ** 2)

- ✓ 测试数据残差平方和：

…> ssResTest = sum((yTestPredicted - yTest) ** 2)

- ✓ 测试数据方差：

…> ssTotalTest = sum((np.mean(yTest) - yTest) ** 2)

- ✓ 测试数据R方：

…> rsquareTest = 1 - ssResTest / ssTotalTest

口 LinearRegression提供的自动计算方法：

- ✓ 训练数据残差平方和：model._residues，通过训练数据训练模型后自动获得
- ✓ 自动计算R方的函数：model.score(xTest, yTest)

代码3.3

```
#代码 3.3 加入评价的基于 Scikit-learn 实现房价预测线性回归代码
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

xTrain = np.array([[75], [87], [105], [110], [120]]) # 训练数据(面积)
yTrain = np.array([270,280,295,310,335]) # 训练数据(总价)
xTest = np.array([85, 90, 93, 109])[:,np.newaxis] # 测试数据(面积)
yTest = np.array([280, 282, 284, 305]) # 测试数据(总价)

model = LinearRegression()
model.fit(xTrain, yTrain)

# 针对训练数据进行预测以计算训练误差
yTrainPredicted = model.predict(xTrain)
# 针对测试数据进行预测
yTestPredicted = model.predict(xTest)
# 手动计算训练数据集残差
ssResTrain = sum((yTrainPredicted - yTrain)**2) print(ssResTrain)
# 自动计算的训练数据集残差
print(model._residues)
```

227.29
227.29
0.80
0.80

```
# 手动计算测试数据集残差
ssResTest = sum((yTestPredicted - yTest)**2)
# 手动计算测试数据集 y 值偏差平方和
ssTotalTest = sum((np.mean(yTest) - yTest)**2)
# 手动计算测试数据 R 方
rsquareTest = 1 - ssResTest / ssTotalTest
print(rsquareTest)
# 自动计算的测试数据集的 R 方
print(model.score(xTest, yTest))

plt.plot(xTrain, yTrain, 'r.') # 训练点数据(红色, 小点)
plt.plot(xTest, yTest, 'bo') # 测试点数据(蓝色, 大点)
plt.plot(xTrain, yTrainPredicted, 'g-') # 拟合的函数直线(绿色)
plt.show()
```



最小二乘法求解

- 根据残差的定义，单个训练数据点的残差是 $f(\mathbf{x}^{(i)}) - y^{(i)}$
- 训练目标是最小化训练数据残差绝对值之和

$$\sum_{i=1}^m |f(\mathbf{x}^{(i)}) - y^{(i)}|$$

- 绝对值不容易进行包括导数运算，因此采用数据的残差平方和

$$\sum_{i=1}^m |f(\mathbf{x}^{(i)}) - y^{(i)}|^2$$

替代残差绝对值之和作为优化目标，使得残差平方和最小化，这种方法被称为**最小二乘法**

- 对最小二乘法的优化目标进行求解，有两种具体的方法，分别是**导数法**和**矩阵法**

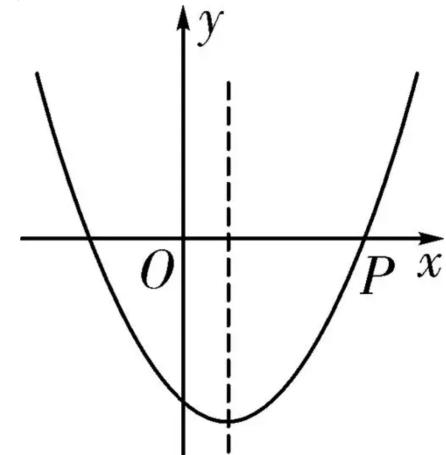
导数法求解最小二乘法

□ 训练目标: $L(w_0, w_1) = \frac{1}{m} \sum_{i=1}^m (f(\mathbf{x}^{(i)}) - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^m (w_1 x_1^{(i)} + w_0 - y^{(i)})^2$

□ 函数 L 也可被称为 “损失” (Loss) 函数

✓ 分别对 w_0 和 w_1 求一阶偏导, 并使之为0

$$\begin{cases} \frac{\partial L}{\partial w_0} = 2 \sum_{i=1}^m (w_1 x_1^{(i)} + w_0 - y^{(i)}) = 0 \\ \frac{\partial L}{\partial w_1} = 2 \sum_{i=1}^m x_1^{(i)} (w_1 x_1^{(i)} + w_0 - y^{(i)}) = 0 \end{cases}$$



✓ 求得 $w_0 = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - w_1 x_1^{(i)}) = \bar{y} - w_1 \bar{x}_1$, 其中 \bar{y} 和 \bar{x}_1 表示平均值

✓ 求得 $w_1 = \frac{\sum x_1^{(i)} y^{(i)} - m \bar{y} \bar{x}_1}{\sum (x_1^{(i)})^2 - m \bar{x}_1^2}$ (演示推导过程)

□ 最终公式: $\begin{cases} w_0 = \bar{y} - w_1 \bar{x}_1 \\ w_1 = \frac{\text{cov}(x_1, y)}{\text{var}(x_1)} \end{cases}$ (演示推导过程)

代码3.4

```
#代码3.4 使用导数求解最小二乘法  
import numpy as np  
X_train = np.array([75, 87, 105, 110, 120]) # 训练数据(面积)  
y_train = np.array([270,280,295,310,335]) # 训练数据(总价)  
  
w1 = np.cov(X_train , y_train, ddof = 1)[1, 0] / np.var(X_train, ddof = 1)  
w0 = np.mean(y_train) - w1 * np.mean(X_train)
```

w1= 1.350592165198907

w0= 163.75113877922863

使用矩阵运算求解

□ 下面在使用矩阵法对最小二乘法的优化目标进行求解

$$y = w_0 \cdot 1 + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_d \cdot x_d$$

□ 转为 $y = \mathbf{x}^T \mathbf{w}$ 和 $\mathbf{y} = \mathbf{X}^T \mathbf{w}$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & \cdots & x_2^{(m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_d^{(1)} & x_d^{(2)} & x_d^{(3)} & \cdots & x_d^{(m)} \end{bmatrix}$$

□ 求得结果 $\mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{Y}^T$ (演示推导过程)

代码3.5和3.6

```
#代码3.5 求参数w的矩阵方法代码: linreg_matrix.py  
import numpy as np  
  
def linreg_matrix(x, y):  
    X_X_T = np.matmul(x, x.T)  
    X_X_T_1 = np.linalg.inv(X_X_T)  
    X_X_T_1_X = np.matmul(X_X_T_1, x)  
    X_X_T_1_X_Y_T = np.matmul(X_X_T_1_X, y.T)  
  
return X_X_T_1_X_Y_T
```

```
x= [[ 1  1  1  1  1]  
     [ 75 87 105 110 120]]  
  
y= [[270 280 295 310 335]]  
  
w= [[163.75113878]  
     [ 1.35059217]]
```

```
#代码3.6 使用矩阵方法求解房价预测问题  
import numpy as np  
  
from linreg_matrix import linreg_matrix  
  
# 训练数据(面积), 每行表示一个数据点  
xTrain = np.array([[ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ]])  
# 训练数据(总价), 每行表示一个数据点  
yTrain = np.array([270, 280, 295, 310, 335])[:,np.newaxis]  
  
def make_ext(x) : #对x进行扩展, 加入一个全1的行  
    ones = np.ones(1)[:, np.newaxis] #生成全1的行向量  
    new_x = np.insert(x, 0, ones, axis = 0)  
    return new_x  
  
#为适应公式3.11的定义, 将xTrain和yTrain进项转换, 使得每一列  
表示一个数据点  
x = make_ext(xTrain.T)  
y = yTrain.T  
print("x=", x)  
print("y=", y)  
w = linreg_matrix(x, y)  
print("w=", w)
```

目录

- 线性回归问题简介
- 单变量线性回归问题
- 基于Scikit-learn库求解单变量线性回归
- 自定义求解单变量线性回归
 - 基于最小二乘法
 - 基于梯度下降法
- 多变量线性回归问题