

Python编程及人工智能应用

第三章 线性回归及Python编程

<https://bolei-zhang.github.io/course/python-ai.html>

- 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

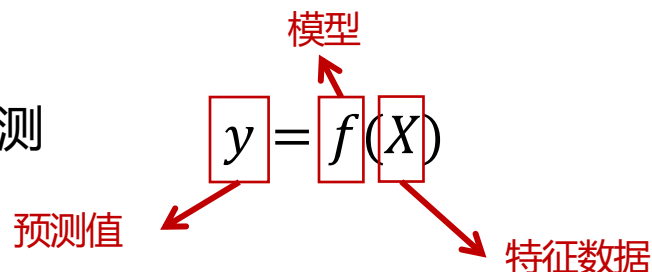
- 机器学习

- 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

- 机器学习“预测”场景

- 根据房屋位置、面积、装修等预测房价
 - 根据图像预测图像的分类，对图像进行分割、检测
 - 机器翻译、自动问答、文本理解



- 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

标签 label

样本 instance

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： $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$

- w 和 b 是可学习和调整的参数，可自动从数据中学习获得
- 预测某套商品房的总价

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

- 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。

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。

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) #显示网格
```

```
return plt
```

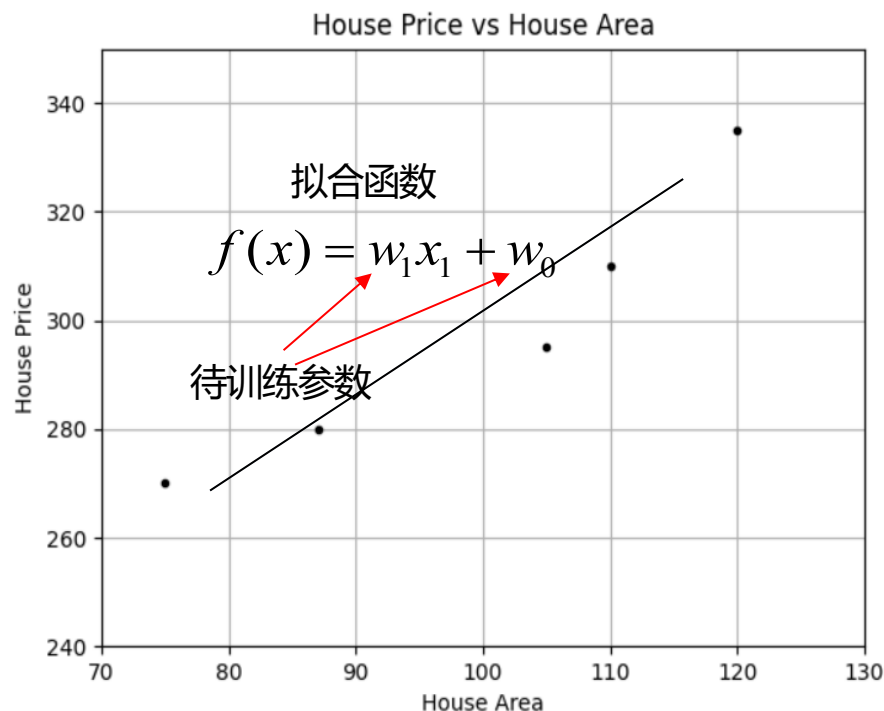
```
plt = initPlot()
```

```
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()
```



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)

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

- 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，否则X会被改写；
- 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，即公式中的 w_0 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；

- 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

Python编程及人工智能应用

第三章 线性回归及Python编程

<https://bolei-zhang.github.io/course/python-ai.html>

内容提要

- 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
- 多变量线性回归问题
- 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

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1：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2：模型结果是否准确？

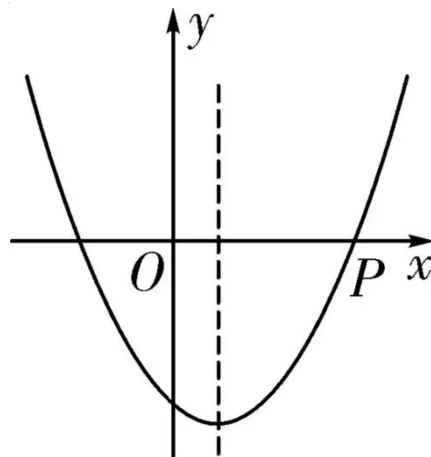
最小二乘法求解

- **训练目标**是最小化训练数据残差平方之和
 - 为什么选择这个目标
 - 误差符合高斯分布
 - 其他损失函数：huber loss, quantile loss
- 采用数据的残差平方和和作为优化目标，求解最优的参数（ w 和 b ），使得残差平方和最小化，这种方法被称为最小二乘法（Ordinary least squares method）

- 优化目标

$$\min \sum_{i=1}^n ((w_1 x_{i1} + w_0) - y_i)^2$$

- 导数为0的位置取到最小值



导数法求解最小二乘法

$$L = \sum_{i=1}^n ((w_1 x_{i1} + w_0) - y_i)^2$$

• 分别对 w_0 和 w_1 求一阶偏导，并使之等于0

• 求得
$$\begin{cases} \frac{\partial L}{\partial w_0} = 2 \sum_{i=1}^n ((w_1 x_{i1} + w_0) - y_i) = 0 \\ \frac{\partial L}{\partial w_1} = 2 \sum_{i=1}^n x_{i1} ((w_1 x_{i1} + w_0) - y_i) = 0 \end{cases}$$

• 最终公式：

$$w_1 = \frac{\sum_{i=1}^n y_i (x_i - \bar{x})}{\sum_{i=1}^n x_i^2 - \frac{1}{n} (\sum_{i=1}^n x_{i1})^2} \quad w_0 = \frac{1}{n} \sum_{i=1}^n (y_i - w_1 x_{i1})$$

线性回归问题具有解析解 (closed-form solution)

#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

- 使用矩阵法对最小二乘法的优化目标进行求解 $y_i = \mathbf{w}^T \mathbf{x}_i + b$

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} & 1 \\ x_{21} & x_{22} & \dots & x_{2d} & 1 \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nd} & 1 \end{bmatrix}$$

- 目标损失函数:

$$L = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

- 对 \mathbf{w} 求导^[1] :

$$\frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

- 当 $\mathbf{X}\mathbf{X}^T$ 为可逆时求得结果

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

#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_train = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ],  
[ 120 ] ] )
```

```
y_train = 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
```

```
x = make_ext ( X_train.T )
```

```
y = y_train.T
```

```
print ( "x=", x )
```

```
print ( "y=", y )
```

```
w = linreg_matrix ( x, y )
```

```
print ( "w=", w )
```

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

开始训练模型



```
# 以矩阵形式表达(对于单变量，矩阵就是列向量形式
X_train = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ],
[ 120 ] ] ) )
y_train = np.array ( [ 270, 280, 295, 310, 335 ] )
```

模型训练

```
model = LinearRegression ( ) # 创建模型对象
model.fit ( X_train, y_train )    # 根据训练数据拟合
出直线(以得到假设函数)
```

#查看模型参数

```
print ( "截距b或w0=", model.intercept_ )    # 截距
print ( "斜率w1=", model.coef_ )           # 斜率
```

模型预测

```
X_test = np.array ( [ [85], [90], [93], [109] ] )
y_pred= model.predict ( X_test )
print ( "预测新房源数据的总价：", y_pred )
```

```
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 ( X_train, y_train, c='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运行结果

•输出

截距b或w0= 163.75113877922868

斜率w1= [1.35059217]

新房源数据的面积： [[85], [90], [93], [109]]

预测新房源数据的总价：

[278.55, 285.30, 289.35, 310.96]



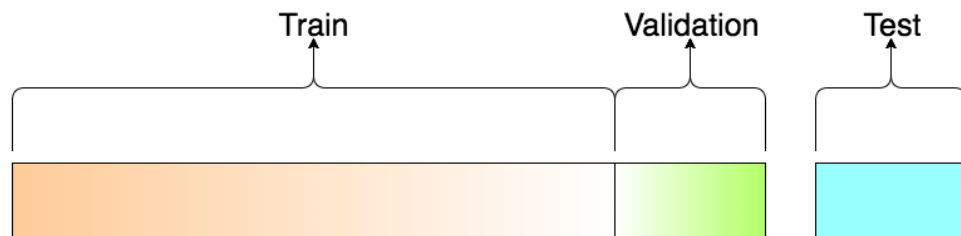
问题2：预测的模型是否准确？

模型结果评价

•数据集

- 1. 在测试数据集X_test中，并不知道测试数据集中的数据所对应的真实的结果，因此无法用测试数据集来评估模型
- 2. 在训练数据集 (X_train, y_train) 中，虽然可以进行比较，但该结果是否在测试数据集中也有效？
- 3. 验证数据集：

- 从训练数据集中切分出一部分数据
- 该部分数据并不参与训练



•用什么指标对拟合模型进行评价？

- 计算回归 (regression) 误差的指标主要包括

- MSE (均方误差)

- r2 (r-squared , 决定系数 , coefficient of determination)

- mape (平均百分比误差) $MAPE = \frac{100\%}{n} \sum_{i=1}^n \left| \frac{\hat{y}_i - y_i}{y_i} \right|$

- 具体采用哪一种指标更合理是和实际问题高度相关的

重新回顾这个问题

- 第一步：准备训练数据和验证数据

- `X_train = np.array([[75], [87], [105], [110], [120]])`
- `y_train = np.array ([270, 280, 295, 310, 335])`
- `X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2)`

- 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：模型验证

- `y_pred = model.predict(X_val)`
- `r2 = r2_score(y_val, y_pred)`
- 如果结果不好，则回到第一步，考虑优化模型

- 第四步：准备测试数据并预测

- `X_test=np.array([[82], [104]])`
- `mode.predict (np.array ([[70]]))`

- 实际 y 值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 值与实际 y 值完全一致。

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



- 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

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

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

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

- r^2 为0、1分别代表什么 ?

残差与R方的计算

- 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：
 - `metrics.r2_score(X_test, y_test)`
 - `model.score(X_test, y_test)`

```
#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
X_train = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ] ] ) # 训练数据(面积)
Y_train = np.array ( [ 270,280,295,310,335 ] ) # 验证数据(总价)
X_val = np.array ( [ 85, 90, 93, 109 ] ) [ :, np.newaxis ] #验证数据(面积)
y_val = np.array ( [ 280, 282, 284, 305 ] ) # 测试数据(总价)
model = LinearRegression ( )
model.fit ( X_train, y_train )

y_pred = model.predict ( X_val ) # 针对验证数据进行预测
ss_res = sum ( ( y_pred - y_val ) ** 2 ) # 计算验证数据集残差
ss_total = sum ( ( np.mean ( y_val ) - y_val ) ** 2 ) # 手动计算验证数据集y值偏差平方和
```



227.29
227.29
0.80
0.80

```
r2 = 1 - ss_res / ss_total # 手动计算验证数据R方
print ( r2 )
print ( model.score ( X_val, y_val ) ) # 自动计算的验证数据集的R方
def initPlot ( ) :
    plt.figure ( )
    plt.title ( 'House Price vs House Area' )
    plt.xlabel ( 'House Area' )
    plt.ylabel ( 'House Price' )
    plt.axis ( [ 70, 130, 250, 350 ] ) # 设置x轴和y轴的值域分别为70~130和240~350
    plt.grid ( True )
    return plt
plt = initPlot ( )
plt.plot ( X_train, y_train, 'r.' ) # 训练点数据(红色, 小点)
plt.plot ( X_val, y_val, 'bo' ) # 验证数据(蓝色, 大点)
plt.plot ( X_val, y_pred, 'g-' ) # 拟合的函数直线(绿色)
plt.show ( )
```

思考



- 对于样本、特征非常多的数据集，例如100万个样本，1000维特征，用最小二乘法求解是否合适？
- 如果数据集的特征多余样本，例如1000个样本，10000维特征，是否可以用最小二乘法？

•

Python编程及人工智能应用

第三章 线性回归及Python编程

<https://bolei-zhang.github.io/course/python-ai.html>

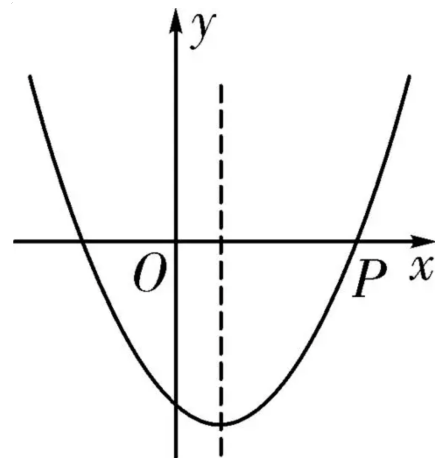
- 线性回归问题简介
- 单变量线性回归问题
- 基于Scikit-learn库求解单变量线性回归
- 自定义求解单变量线性回归
 - 基于最小二乘法
 - 基于梯度下降法
- 多变量线性回归问题

模型优化

- 在线性回归问题中，当优化目标是均方误差时，可以有闭式解

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- 但是在实际情况中
 - 闭式解的计算开销往往很高
 - 假设数据包含 $n=100$ 万个样本，每个样本有 $d=1000$ 个特征
 - 矩阵相乘： $n*d*n$
 - 矩阵求逆： $n*n*n$
 - $\mathbf{X}\mathbf{X}^T$ 为往往不是满秩矩阵，需要有正则项
 - [sklearn.linear_model.SGDRegressor](#)



- **梯度下降法**：在一个随机的位置沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度
- **牛顿法**：在极小点附近通过二阶泰勒展开，找到极小点的下一个估计值

二次函数梯度下降法求解

- 梯度下降法是一种求函数极值的数值方法，其基本思想是对目标函数选取一个初始点，从该点出发，根据梯度（导数）的方向步步运算，能够到达一个极值，该极值可能是局部最优解

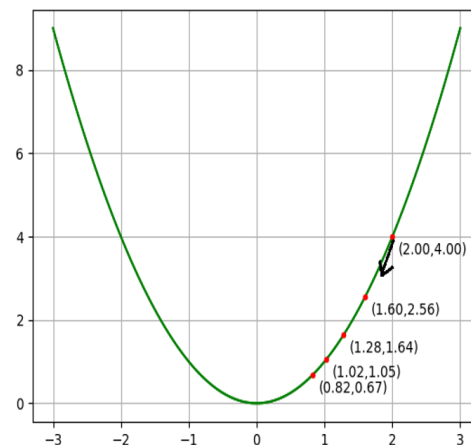
- 对于目标损失函数 $f(w) = w^2$ 的梯度下降法求极值

- 第一步：选定一个初始值，比如选定 $w_{(0)} = 2$ ， $f(w_{(0)}) = 4$
- 第二步：计算该点的导数 $f'(w_{(0)}) = 2w_{(0)}$
- 第三步：按公式计算 w 的新值 $w_{(1)}$

$$w_{(1)} = w_{(0)} - l_r f'(w_{(0)})$$

- $l_r = 0.1$ 是学习速度

- $f'(w)$ 用于控制每步前进的方向，朝着导数相反方向前进



- 第四步：计算 $f(w_{(1)})$ ，并判断是否达到精度要求（如 0.00001 ）
 - 第五步：判断是否达到精度要求，如果未达到，则跳转到第二步继续执行
- 可能输出不精确的极值；
- 如果有多个极值，则有可能找到的是局部最优点，而非全局最优点

代码3.7



```
def obj_fun(x):  
    return x * x #需要求极值的目标函数  
def dir_fun(x):  
    return 2 * x #目标函数的导数  
  
x_list = []  
y_list = [] # 用于保存经过的点  
def minimize ( init_x, lr = 0.1, diff = 1e-9, max_iter = 1000 ) :  
    # init_x: 初始点, lr: 学习速率, dif: 相邻两步差异临界值, m  
    ax_iter: 最大迭代次数  
    x0 = init_x  
    y0 = obj_fun ( x0 )  
    x_list.append ( x0 )  
    y_list.append ( y0 )  
    for i in range ( max_iter ) :  
        x1 = x0 - lr * dir_fun ( x0 )  
        y1 = obj_fun ( x1 )  
        x_list.append ( x1 )  
        y_list.append ( y1 )
```

if abs (y1 - y0) <= diff : #达到收敛条件

```
    print ( "是否收敛 : True", "极小点 : (%e, %e)"%(x  
1, y1), "循环次数 : %d" % I )
```

```
    return
```

```
    x0 = x1
```

```
    y0 = y1
```

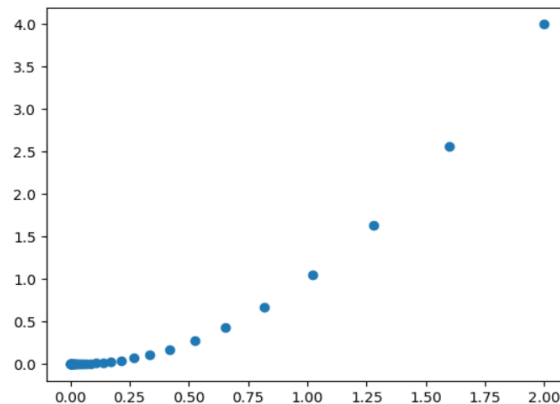
```
    print ( "是否收敛 : False", "极小点 : NaN", "循环次数 :  
%d" % max_iter )
```

```
minimize ( 2.0 )
```

```
plt.plot ( x_list, y_list, "
```

```
plt.show ( )
```

是否收敛 : True 极小点 :
(3.794275e-09, 1.4396
52e-17) 循环次数 : 89



批量梯度下降法

- 对于最小二乘法，其目标函数是：

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

- 优化的参数有两个，分别是 w_0 和 w_1
- 需要一次性批量地使用**全部**训练数据，被称为批量梯度下降法（Batch Gradient Decent）

• 算法步骤：

• 第一步：随机选定 w_0 和 w_1 的初始值，并计算 $L(w_0, w_1)$

• 第二步： w_0 和 w_1 的偏导函数分别为 $\frac{\partial L}{\partial w_0}$ 和 $\frac{\partial L}{\partial w_1}$ ，偏导方向 w 变化最快

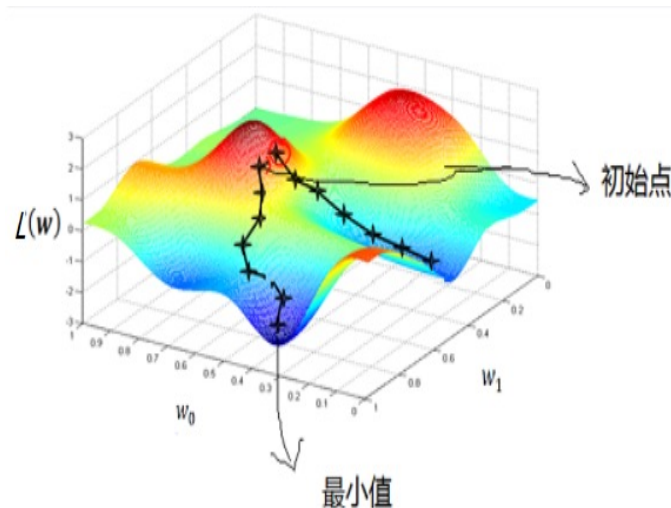
$$w_0 = w_0 - lr * \frac{\partial L}{\partial w_0} \quad w_1 = w_1 - lr * \frac{\partial L}{\partial w_1}$$

- 利用更新后的 w_0 和 w_1 值再次计算 $L(w_0, w_1)$ 和其导数
- 第三步：循环执行第二步，直到参数变化小于某个临界值或循环迭代次数达到设定的最大次数

• 演示推导过程

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \frac{1}{m} \sum_{i=1}^m (w_1 x_1^{(i)} + w_0 - y^{(i)})$$

$$\frac{\partial L(w_0, w_1)}{\partial w_1} = \frac{1}{m} \sum_{i=1}^m [(w_1 x_1^{(i)} + w_0 - y^{(i)}) \cdot x_1^{(i)}]$$



代码3.8 批量梯度下降实现

#代码3.8 批量梯度下降法 : bgd_optimizer.py

```
def bgd_optimizer ( target_fn, grad_fn, init_w, X, Y, lr=0.0001,tolerance=1e-12, max_iter=1000000  
00 ):
```

```
    w = init_w
```

```
    target_value = target_fn ( w, X, Y )          # 计算当前w值下的L(w)值
```

```
    for i in range ( max_iter ):
```

```
        grad = grad_fn ( w, X, Y )              # 计算梯度
```

```
        next_w = w - grad * lr                  # 向量计算，调整了w
```

```
        next_target_value = target_fn ( next_w, X, Y )    # 计算新值
```

```
        # 如果两次计算之间的误差小于tolerance，则表明已经收敛
```

```
        if abs ( next_target_value - target_value ) < tolerance:
```

```
            return i, next_w #返回迭代次数和参数w的值
```

```
        else: w, target_value = next_w, next_target_value    # 继续进行下一轮计算
```

```
    return i, None #返回迭代次数，由于未收敛，w没有优化的值
```

代码3.9



#代码3.9 基于梯度下降法求解房价预测问题

```
import numpy as np
```

```
from bgd_optimizer import bgd_optimizer
```

```
import matplotlib.pyplot as plt
```

```
def target_function ( w, X, Y): # 定义目标函数
```

```
    w0, w1 = w # w:[w0, w1]
```

```
    # 应使用np.sum, 而不要使用sum。
```

```
    #np.sum支持向量/矩阵运算
```

```
    return np.sum ( ( w0 + X * w1 - Y ) ** 2 ) / ( 2 * len  
( X ) )
```

```
# 根据目标函数定义梯度, 对x0和x1求导数,
```

```
# 累计各点导数平均值
```

```
def grad_function ( w, X, Y):
```

```
    w0, w1 = w
```

```
    # 对应w0的导数
```

```
    w0_grad = np.sum ( w0 + X * w1 - Y ) / len ( X )
```

```
    # 对应w1的导数。注意采用向量运算
```

```
    w1_grad = X.dot ( w0 + X * w1 - Y ) / len ( X )
```

```
    return np.array ( [ w0_grad, w1_grad ] )
```

```
# 训练数据(面积)
```

```
x = np.array ( [ 75, 87, 105, 110, 120 ], dtype = np.float64 )
```

```
# 训练数据(总价)
```

```
y = np.array ( [ 270, 280, 295, 310, 335 ], dtype = np.float64 )
```

```
np.random.seed ( 0 )
```

```
init_W = np.array ( [ np.random.random ( ), np.random.random ( ) ] ) # 随机初始化W值
```

```
i, W= bgd_optimizer ( target_function, grad_function, init_W, x, y )
```

```
if W is not None :
```

```
    w0, w1 = w
```

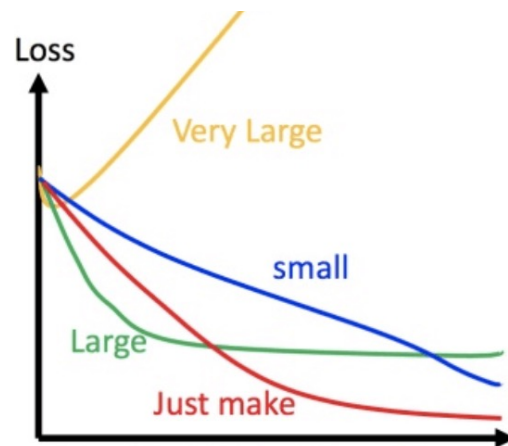
```
    print ( '迭代次数: 4051854 , 最优的w0和w1:(163.746743, 1.350635)' )
```

思考: 如果将lr设为较大值 (如设置为0.01或0.1) 会出现什么状况?

如果将lr设为较小值 (如设置为1e-6) 呢?

参数的选择

- 学习率 lr 的选择



- w 初始值的选择

- 不要距离目标值太远

随机梯度下降法

- 批量梯度下降法的缺陷：当样本数量非常庞大时，计算复杂、费时且极有可能超出硬件能力（内存容量、CPU计算能力等）限制。
- 随机梯度下降法（Stochastic Gradient Decent，简称SGD），其原理是每次随机选取一部分样本对目标函数进行优化。
- 随机梯度下降法能够达到不弱于批量梯度下降法的优化效果，在实际场景中应用更加广泛。

代码3.10



#代码3.10 随机梯度下降法 : sgd_optimizer.py

```
import numpy as np
```

```
def sgd_optimizer ( target_fn,grad_fn, init_w, X, Y, lr=0.0001, tolerance=1e-12, max_iter=1000000000 ) :
```

```
    w, rate = init_w, lr
```

```
    min_w, min_target_value = None, float ( "inf" )
```

```
    no_improvement = 0
```

```
    target_value = target_fn ( W, X, Y )
```

```
    for i in range ( max_iter ) :
```

```
        index = np.random.randint ( 0, len ( X ) )           # 获得一组随机数据的索引值
```

```
        gradient = grad_fn ( w, X [ index ], Y [ index ] )   # 计算该数据点处的导数
```

```
        w = w - lr * gradient
```

```
        new_target_value = target_fn ( w, X, Y )
```

```
        if abs ( new_target_value - target_value ) < tolerance :
```

```
            return i, w
```

```
    return i, None
```

批量梯度下降vs随机梯度下降

- 运算速度

- 批量：可以通过矩阵计算梯度，更易于并行，速度更快
- 随机：不易于并行

- 运算空间

- 批量：需要所有样本，在样本很大时，运算空间要求很高
- 随机：每一轮只需要一个样本，单步更新快，需要的空间少

- 算法稳定性

- 批量：通过所有样本计算梯度，更稳定
- 随机：需要通过每个样本计算梯度，不稳定

多变量线性回归问题

• 现实生活中，房价不仅与面积有关，也与户型、楼层等相关。增加了房屋户型作为因变量，变化后的数据如表所示

• 户型类型：1（一居室）、2（二居室）、3（三居室）、4（四居室）

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

验证样本	房屋面积（平方米）	房屋户型	房屋总价（万元）
1	85	2	280
2	90	3	282
3	93	3	284
4	109	4	305

基于Scikit-learn库求解



#代码3.12 使用Scikit-learn库实现多变量房价预测问题求解

```
import numpy as np
```

```
from sklearn.linear_model import LinearRegression
```

```
xTrain = np.array ( [ [ 75, 1 ], [ 87, 3 ], [ 105, 3 ], [ 110, 3 ], [ 120, 4 ] ] ) # 训练数据(面积 , 户型)
```

```
yTrain = np.array ( [ 270,280,295,310,335 ] ) # 训练数据(总价)
```

```
xTest = np.array ( [ [ 85, 2 ], [ 90, 3 ], [ 93, 3 ], [ 109, 4 ] ] ) # 验证数据(面积 , 户型)
```

```
yTest = np.array ( [ 280, 282, 284, 305 ] ) #验证数据(总价)
```

```
model = LinearRegression ( )
```

```
model.fit ( xTrain, yTrain )
```

```
print ( model._residues ) #验证数据集残差 : 226.8291
```

```
print ( model.score ( xTest, yTest ) ) #验证数据集的R方 : 0.8374
```

残差 : 226.82910636037715

R方 : 0.8373813078020356

基于最小二乘法求解

#代码3.13 使用矩阵方法求解多变量房价预测问题

```
import numpy as np
from linreg_matrix import linreg_matrix
xTrain = np.array ( [ [ 75, 1 ], [ 87, 3 ], [ 105, 3 ], [ 110, 3 ], [ 120,
4 ] ] ) # 训练数据(面积, 户型)
# 训练数据(总价)
yTrain = np.array ( [ 270, 280, 295, 310, 335 ] )
# 验证数据(面积, 户型)
xTest = np.array ( [ [ 85, 2 ], [ 90, 3 ], [ 93, 3 ], [ 109, 4 ] ] )
# 验证数据(总价)
yTest = np.array ( [ 280, 282, 284, 305 ] )

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3.9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
```

```
w = linreg_matrix ( x, y )
```

```
print ( "w=", w )
```

```
yTrainPredicted = w.dot ( x )
```

```
yTestPredicted = w.dot ( make_ext ( xTest.T ) )
```

```
ssResTrain = sum ( ( yTrain - yTrainPredicted ) ** 2 )
```

```
print ( "训练数据残差平方和 : ", ssResTrain )
```

```
ssResTest = sum ( ( yTest - yTestPredicted ) ** 2 )
```

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

```
rsquareTest = 1 - ssResTest / ssTotalTest
```

```
print ( "验证数据R方 : ", rsquareTest )
```

```
w= [162.19293587  1.38427832  -0.639357
32]
```

```
训练数据残差平方和 : 226.8291063603765
```

```
验证数据R方 : 0.837381307801918
```

基于梯度下降法求解

- 梯度下降方法同样可以用于多变量房价预测求解，需要对目标函数和梯度函数做一些扩展。

$$L(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^m \left(\sum_{k=1}^d w_k x_k^{(i)} + w_0 - y^{(i)} \right)^2$$

$$\frac{\partial L(\mathbf{w})}{\partial w_0} = \frac{1}{m} \sum_{i=1}^m \left(\sum_{k=1}^d w_k x_k^{(i)} + w_0 - y^{(i)} \right)$$

- 对于 $j > 0$ ，有

$$\frac{\partial L}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m \left[\left(\sum_{k=1}^d w_k x_k^{(i)} + w_0 - y^{(i)} \right) x_j^{(i)} \right]$$

代码3.14



#代码3.14 基于批量梯度下降法求解多变量房价预测问题

```
def target_function ( w, X, Y ): # 定义目标函数
```

```
    return np.sum ( ( w [ 0 ] + X.dot ( w [ 1: ] ) - Y ) ** 2 ) / ( 2 * len ( X ) )
```

```
def grad_function ( w, X, Y ): # 根据目标函数定义梯度 , 对w0, w1, w2求导数平均值
```

```
    w0_grad = np.sum ( w [ 0 ] + X.dot ( w [ 1: ] ) - Y ) / len ( X ) #对应w0的导数
```

```
    w1_grad = X [ :, 0 ].dot ( np.array ( w [ 0 ] ) + X.dot ( w [ 1: ] ) - Y ) / len ( X ) #对应w1的导数
```

```
    w2_grad = X [ :, 1 ].dot ( np.array ( w [ 0 ] ) + X.dot ( w [ 1: ] ) - Y ) / len ( X ) #对应w2的导数
```

```
    return np.array ( [ w0_grad,w1_grad,w2_grad ] )
```

```
x = np.array ( [ [ 75, 1 ], [ 87, 3 ], [ 105, 3 ], [ 110, 3 ], [ 120, 4 ] ], dtype = np.float64 )
```

```
y = np.array ( [ 270, 280, 295, 310, 335 ], dtype = np.float64 )
```

```
np.random.seed ( 0 )
```

```
init_W = np.array ( [ np.random.random(), np.random.random ( ), np.random.random ( ) ] ) #随机初始化W
```

```
i, w = bgd_optimizer ( target_function, grad_function, init_w, x, y )
```

```
if W is not None :
```

```
    w0, w1, w2 = w
```

```
    print ( "迭代次数 : %d , 最优的w0, w1, w2:(%f, %f, %f)" % ( i, w0, w1, w2 ) )
```

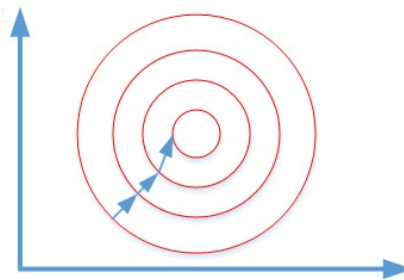
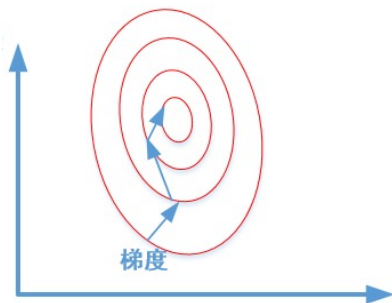
```
else: print ( "达到最大迭代次数 , 未收敛" )
```

迭代次数 : 6598762 ,
最优的w0, w1, w2:(162.
185449, 1.384389, -0.6
40646)

数据归一化问题

- 在多变量情况下，各变量的值域有很大差别，比如房屋面积的范围是几十到几百的浮点数，房屋户型的值域是1到5之间的整数
- 值域差异过大，容易造成计算过程中出现溢出或无法收敛，导致各个变量作用权重受到影响
- 可以对数据进行归一化（Normalization）处理来解决这一问题
- 归一化方法：

$$x_norm_i = \frac{x_i - \bar{x_i}}{std(x_i)}$$



代码3.15



#代码3.15 先进行归一化处理，再使用Scikit-learn库求解

```
import numpy as np
```

```
from sklearn.linear_model import LinearRegression
```

```
def normalize ( X ) :
```

```
    X_mean = np.mean ( X, 0 ) #计算均值
```

```
    X_std = np.var ( X, 0 ) #计算标准差
```

```
    return ( X - X_mean ) / X_std
```

```
xTrain = np.array ( [ [ 75, 1 ], [ 87, 3 ], [ 105, 3 ], [ 110, 3 ], [ 120, 4 ] ] ) # 训练数据(面积 , 户型)
```

```
yTrain = np.array ( [ 270, 280, 295, 310, 335 ] ) # 训练数据(总价)
```

```
xTrain = ( normalize ( xTrain ) )
```

```
model = LinearRegression ( )
```

```
model.fit ( xTrain, yTrain )
```

```
print( model._residues ) # 训练数据集残差
```

输出结果为：226.82910636037712

数据归一化

- 两种常见的归一化方法

- Min-max

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

- Z-score

$$\frac{x - x_{mean}}{x_{std}}$$

- 在进行预测时，是否仍然需要归一化

- 单变量房价预测问题中只有一个自变量“房屋面积”，因此拟合出来的是一条直线；多变量房价预测问题中有两个自变量“房屋面积”和“房屋户型”，因此拟合出来的是一个直平面
- 采用“曲线”或“曲面”模型来拟合能够对训练数据产生更逼近真实值的效果，这就是高阶拟合，有可能是非线性的。

- 对于一个自变量的场景，可以采用多阶函数：

$$f_n(x) = \sum_{k=0}^n w_k x^k$$

- 二阶单变量函数为： $f_2(x) = w_0 + w_1 x + w_2 x^2$

- 三阶单变量函数为： $f_3(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

- 四阶单变量函数为： $f_4(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

- 可以通过如下方式进行变换：

- 加入新变量 x_1 ，设 $x_1 = x$ ；
- 加入新变量 x_2 ，设 $x_2 = x^2$ ；
- 加入新变量 x_3 ，设 $x_3 = x^3$ ；
- 加入新变量 x_4 ，设 $x_4 = x^4$ ；

代码3.16



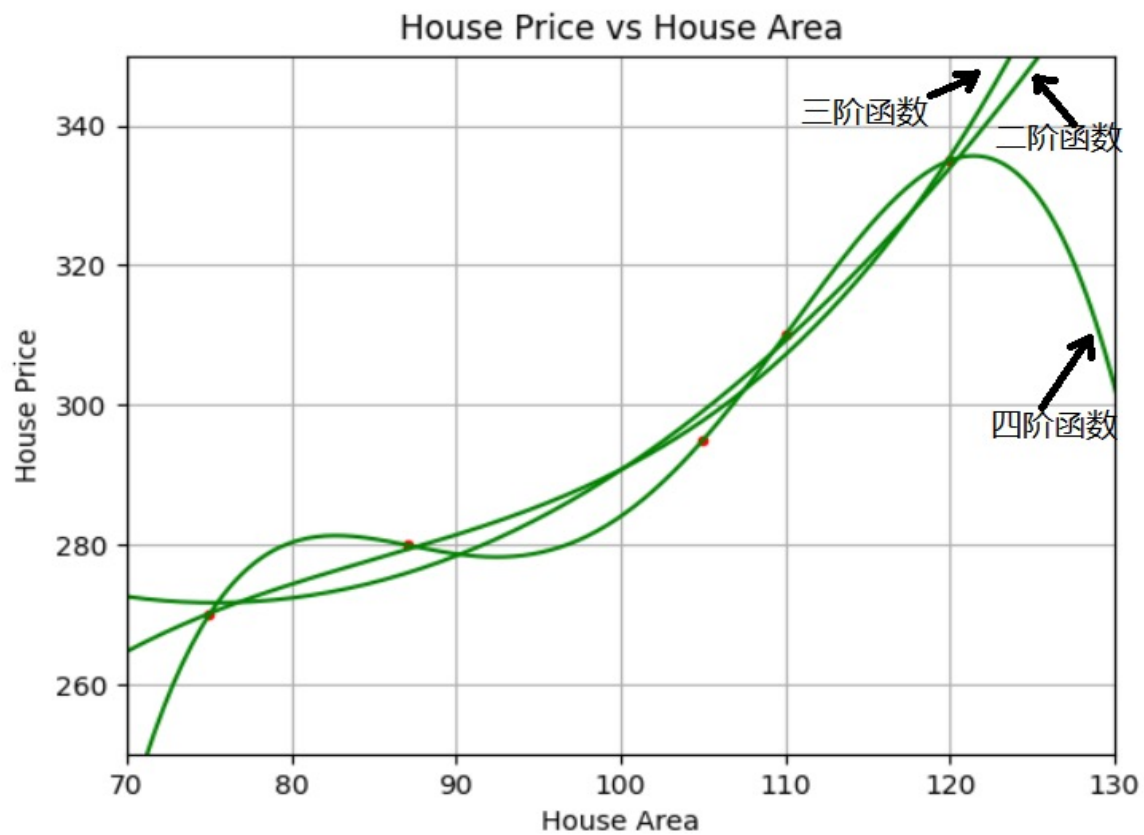
```
#代码3.16 使用Scikit-learn库实现高阶拟合单变量房价预测
xTrain = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ] ] )
x1 = xTrain
x2 = xTrain ** 2
x3 = xTrain ** 3
x4 = xTrain ** 4
new_xTrain2 = np.concatenate ( [ x1, x2 ], axis = 1 )
new_xTrain3 = np.concatenate ( [ x1, x2, x3 ], axis = 1 )
new_xTrain4 = np.concatenate ( [ x1, x2, x3, x4 ], axis =
1 )

yTrain = np.array ( [ 270, 280, 295, 310, 335 ] )
xTest = np.array ( [ [ 85 ], [ 90 ], [ 93 ], [ 109 ] ] )
x1 = xTest
x2 = xTest ** 2
x3 = xTest ** 3
x4 = xTest ** 4
new_xTest2 = np.concatenate ( [ x1, x2 ], axis = 1 )
new_xTest3 = np.concatenate ( [ x1, x2, x3 ], axis = 1 )
new_xTest4 = np.concatenate ( [ x1, x2, x3, x4 ], axis = 1 )
```

```
new_xTest4 = np.concatenate ( [ x1, x2, x3, x4 ], axis = 1 )
yTest = np.array ( [ 280, 282, 284, 305 ] )      # 测试数据(总价)
model1 = LinearRegression ( )
model1.fit ( xTrain, yTrain )
print ( "一阶训练数据集残差：", model1._residues )
print ( "一阶测试数据集R方：", model1.score ( xTest, yTest ) )
model2 = LinearRegression ( )
model2.fit ( new_xTrain2, yTrain )
print ( "二阶训练数据集残差：", model2._residues )
print ( "二阶测试数据集R方：", model2.score ( new_xTest2, yTe
一阶训练数据集残差： 227.2965381111449
一阶测试数据集R方： 0.8090280549127149
二阶训练数据集残差： 38.38456969539346
二阶测试数据集R方： 0.8714891148616254      )
三阶训练数据集残差： 16.488230700095926      w_xTest3, yTe
三阶测试数据集R方： 0.9885676670452613
四阶训练数据集残差： 5.498381308907731e-18
四阶测试数据集R方： 0.8830805089175938

print ( "四阶训练数据集残差：", model4._residues )
print ( "四阶测试数据集R方：", model4.score ( new_xTest4, yTe
st ) )
```

绘制图形



回顾整个过程

1. 问题建模

- 回归 ...

2. 收集数据

- 回归：特征数据 X ，连续数据 y

3. 特征预处理

- 归一化
- 类别特征
- 时间特征
- 图像数据、序列数据、图结构数据

4. 构建模型

- 模型选择：线性回归
- 损失函数：均方误差

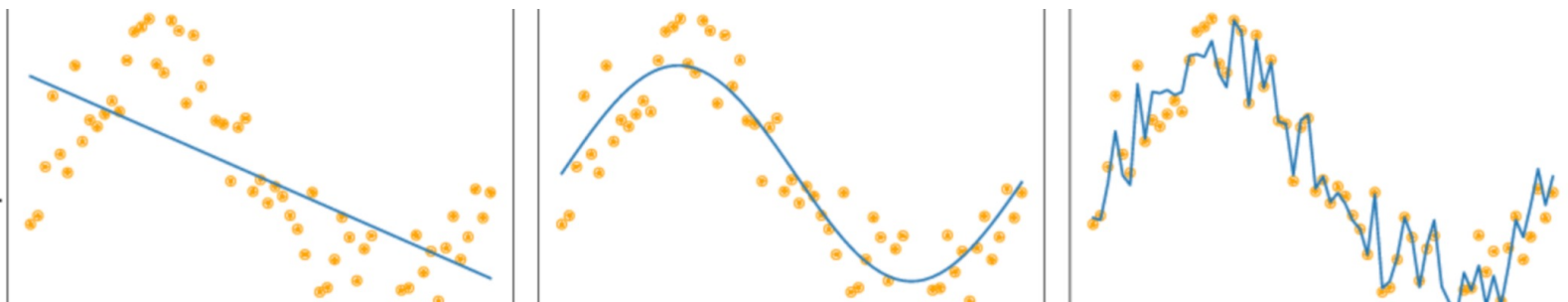
5. 模型验证&参数调优

- 使用训练数据训练模型，使用验证数据进行参数调优
- 验证指标：回归（ w_{mape} 、 R^2 、均方误差）

6. 模型上线/AB测试

- 在测试数据上进行模型测试（测试数据和训练数据来自于同一分布）

- 本章介绍了线性回归问题的定义，以及求解线性回归问题的多种方法，包括Scikit-Learn库函数求解法、最小二乘法、梯度下降法等。通过本章的学习，读者能够对线性回归问题有一个基本的了解，并能掌握通过Python编码求解线性回归问题的基本方法，为后续章节的学习打下良好的基础。



以上三个模型，哪个拟合的最好？