菊安酱的机器学习第7期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2018-12-17

作者：菊安酱

课件内容说明：

- 本文为作者原创，转载请注明作者和出处
- 如果想获得此课件及录播视频，可扫描左边二维码，回复"k"进群
- 如果想获得2小时完整版视频，可扫描右边二维码或点击如下链接
- 若有任何疑问，请给作者留言。

交流群二维码

完整版视频及课件

直播视频及课件：http://www.peixun.net/view/1278.html

完整版视频及课件：http://edu.cda.cn/course/966

12期完整版课纲

直播时间：每周一晚8:00

直播内容：

时间	期数	算法
2018/11/05	第1期	k-近邻算法
2018/11/12	第2期	决策树
2018/11/19	第3期	朴素贝叶斯
2018/11/26	第4期	Logistic回归
2018/12/03	第5期	支持向量机
2018/12/10	第6期	AdaBoost 算法
2018/12/17	第7期	线性回归
2018/12/24	第8期	树回归
2018/12/31	第9期	K-均值聚类算法
2019/01/07	第10期	Apriori 算法
2019/01/14	第11期	FP-growth 算法
2019/01/21	第12期	奇异值分解SVD

线性回归

我们前6期的内容都在讲解分类问题，这一期我们正式进入回归问题。虽然分类问题和回归问题都属于机器学习有监督算法的范畴，但实际上，回归问题要远比分类问题复杂。首先是关于输出结果的对比，分类模型最终输出的结果是离散型变量，而离散变量本身包含信息量较少，其本身并不具备代数运算性质，因此其评价指标体系也较为简单，最常用的就是混淆矩阵和ROC曲线。而回归问题最终输出的是连续变量，其本身不仅能够进行代数运算，而且还具有统计学意义的分布特征，因此其评价指标将更为复杂。同时在描绘客观事物规律上，回归问题是一种更加"精致"的方法，希望对事物运行的更底层原理进行挖掘，也就是说回归类问题的模型更加全面、完善地描绘事物客观规律，从而能够得出更加细粒度的结论。因此回归问题的模型往往更加复杂，建模需要的数据所提供的信息量也越多，进而在建模过程中可能遇到的问题也越多。

一、什么是回归

回归的目的是预测数值型的目标值。最直接的办法是依据输入写出一个目标值的计算公式。假如你想要预测某位小姐姐男友汽车的功率大小，可能会这么计算：

$$HorsePower = 0.0015 * annualSalary - 0.99 * hoursListeningToPublicRadio$$

翻译成中文就是：

$$小姐姐男友汽车功率 = 0.0015 * 男友年薪 - 0.99 * 收听公共广播时间$$

这就是所谓的**回归方程**（regression equation）,其中的0.0015和-0.99称作**回归系数**（regression weights）,求这些回归系数的过程就是回归。一旦有了这些回归系数，再给定输入值，我们计算出回归系数与输入值的乘积之和，就可以得到最后的预测值。

【补充】线性问题中“回归”的含义：https://blog.csdn.net/laputa_ml/article/details/80072570

回归分为线性回归和非线性回归，上述功率计算公式也可以写做：

$$HorsePower = 0.0015 * annualSalary/hoursListeningToPublicRadio$$

这就是一个非线性回归的例子，但我们不对此做深入讨论。这里主要讨论的是**线性回归**。

二、线性回归

1. 简单线性回归

简单线性回归也叫一元线性回归，先来看一元线性方程：

$$y = b + wx$$

这个直线方程相信大家都不陌生，这就是最简单的线性回归模型。其中，w是直线的**斜率**，b是直线的**截距**。

写成矩阵形式为：

$$y = \boldsymbol{X}^T \boldsymbol{\omega}$$

其中

$$\boldsymbol{X} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad (令 x_0 = 1), \quad \boldsymbol{\omega} = \begin{bmatrix} b \\ w \end{bmatrix}$$

假如我们有m个训练样本，则

$$\boldsymbol{X}^T = \begin{bmatrix} x_0^1, x_1^1 \\ x_0^2, x_1^2 \\ \cdots \\ x_0^m, x_1^m \end{bmatrix}$$

2. 多元线性回归

多元线性方程：

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

写成矩阵形式为：

$$y = \boldsymbol{X}^T \boldsymbol{\omega}$$

假如我们有m个训练样本，则

$$\boldsymbol{X}^T = \begin{bmatrix} x_0^1, x_1^1, \ldots, x_n^1 \\ x_0^2, x_1^2, \ldots, x_n^2 \\ \cdots \\ x_0^m, x_1^m, \ldots, x_n^m \end{bmatrix}, \boldsymbol{\omega} = \begin{bmatrix} w_0 \\ w_1 \\ \cdots \\ w_n \end{bmatrix}$$

3. 线性回归的损失函数

在第4期逻辑回归中有跟大家提到过损失函数。损失函数其实就是衡量预测值与真实值之间的差距的函数。这里采用平方误差作为线性回归的损失函数：

$$\begin{aligned} SSE &= \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{m} (y_i - x_i^T w)^2 \end{aligned}$$

【注】用平方而没用误差绝对值是因为：平方对于后续求导比较方便。

用矩阵表示可以写做：

$$(y - \boldsymbol{X}w)^T (y - \boldsymbol{X}w)$$

对w求导，得到$\boldsymbol{X}^T(y - \boldsymbol{X}w)$，令其等于0，解出w如下：

$$\hat{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T y$$

【公式推导过程】：

$$
\begin{aligned}
&\frac{\partial (y-\boldsymbol{X}w)^T(y-\boldsymbol{X}w)}{\partial w} \\
&= \frac{\partial (y^T-(\boldsymbol{X}w)^T)(y-\boldsymbol{X}w)}{\partial w} \\
&= \frac{\partial (y^T-w^T\boldsymbol{X}^T)(y-\boldsymbol{X}w)}{\partial w} \\
&= \frac{\partial (y^Ty-y^T\boldsymbol{X}w-w^T\boldsymbol{X}^Ty+w^T\boldsymbol{X}^T\boldsymbol{X}w)}{\partial w} \\
&= \frac{\partial y^Ty}{\partial w}-\frac{\partial y^T\boldsymbol{X}w}{\partial w}-\frac{\partial w^T\boldsymbol{X}^Ty}{\partial w}+\frac{\partial w^T\boldsymbol{X}^T\boldsymbol{X}w}{\partial w} \\
&= 0-\boldsymbol{X}^Ty-\boldsymbol{X}^Ty+2\boldsymbol{X}^T\boldsymbol{X}w \\
&= -2\boldsymbol{X}^T(y-\boldsymbol{X}w)
\end{aligned}
$$

令其等于0，即：

$$
\begin{aligned}
\frac{\partial (y-\boldsymbol{X}w)^T(y-\boldsymbol{X}w)}{\partial w} &= 0 \\
-2\boldsymbol{X}^T(y-\boldsymbol{X}w) &= 0 \\
\boldsymbol{X}^T(y-\boldsymbol{X}w) &= 0 \\
\boldsymbol{X}^Ty-\boldsymbol{X}^T\boldsymbol{X}w &= 0 \\
\boldsymbol{X}^T\boldsymbol{X}w &= \boldsymbol{X}^Ty
\end{aligned}
$$

当$|\boldsymbol{X}^T\boldsymbol{X}| \neq 0$时，有：

$$\hat{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^Ty$$

此处$\hat{w}$指的是当前可以估计出的w的最优解。

详细说明
1. 这个公式中，y、w为列向量，$\boldsymbol{X}$ 为矩阵
2. 对于$(y-\boldsymbol{X}w)^T(y-\boldsymbol{X}w)$,先不要急着求导，先看看它的形式，是 $u(w)*v(w)$ 的形式，这种形式一般求导较为复杂，因此为了简化运算，我们先把式子展开
3. 展开这个过程中，用到的公式： $(A+B)^T=A^T+B^T$ $(AB)^T=B^TA^T$
4. 展开后的结果为：$(y^Ty-y^T\boldsymbol{X}w-w^T\boldsymbol{X}^Ty+w^T\boldsymbol{X}^T\boldsymbol{X}w)$,然后就可以对4部分分别求导了 这里需要说明的是：累加后求导 = 求导后累加，结果是一样的
5. 说明：这四部分（$\frac{\partial y^Ty}{\partial w}-\frac{\partial y^T\boldsymbol{X}w}{\partial w}-\frac{\partial w^T\boldsymbol{X}^Ty}{\partial w}+\frac{\partial w^T\boldsymbol{X}^T\boldsymbol{X}w}{\partial w}$）有一个共同的特点就是分子部分都是标量，分母部分都是矢量，并且分母为列向量，所以计算结果对应维基百科中Scalar-by-vector identities表格最后一列(即第4列)
6. 计算$\frac{\partial y^Ty}{\partial w}$ = 0 说明：根据维基百科中Scalar-by-vector identities表格第1行第4列，计算结果为0
7. 计算$\frac{\partial y^T\boldsymbol{X}w}{\partial w}=\boldsymbol{X}^Ty$ 根据维基百科中Scalar-by-vector identities表格第11行第4列，计算结果为$\boldsymbol{X}^Ty$
8. 计算$\frac{\partial w^T\boldsymbol{X}^Ty}{\partial w}=\boldsymbol{X}^Ty$ 根据维基百科中Scalar-by-vector identities表格第10行第4列，计算结果为$\boldsymbol{X}^Ty$
9. 计算$\frac{\partial w^T\boldsymbol{X}^T\boldsymbol{X}w}{\partial w}=2\boldsymbol{X}^T\boldsymbol{X}w$ 根据维基百科中Scalar-by-vector identities表格第13行第4列，$\boldsymbol{X}^T\boldsymbol{X}$就相当于**A**，计算结果为$2\boldsymbol{X}^T\boldsymbol{X}w$

关于矩阵的运算，维基百科中介绍非常完备：https://en.wikipedia.org/wiki/Matrix_calculus

维基百科中Scalar-by-vector identities表格如下所示：

Scalar-by-vector identities [edit]

The fundamental identities are placed above the thick black line.

Identities: scalar-by-vector $\frac{\partial y}{\partial \mathbf{x}} = \nabla_{\mathbf{x}} y$

	Condition	Expression	Numerator layout, 分子布局 i.e. by $\mathbf{x}^{\mathrm{T}}$; result is row vector	Denominator layout, 分母布局 i.e. by x; result is column vector
1	a is not a function of $\mathbf{x}$	$\frac{\partial a}{\partial \mathbf{x}} =$	$\mathbf{0}^{\mathrm{T}}$ [4]	$\mathbf{0}$ [4]
2	a is not a function of $\mathbf{x}$, $u = u(\mathbf{x})$	$\frac{\partial au}{\partial \mathbf{x}} =$	$a\frac{\partial u}{\partial \mathbf{x}}$	
3	$u = u(\mathbf{x})$, $v = v(\mathbf{x})$	$\frac{\partial (u+v)}{\partial \mathbf{x}} =$	$\frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}}$	
4	$u = u(\mathbf{x})$, $v = v(\mathbf{x})$	$\frac{\partial uv}{\partial \mathbf{x}} =$	$u\frac{\partial v}{\partial \mathbf{x}} + v\frac{\partial u}{\partial \mathbf{x}}$	
5	$u = u(\mathbf{x})$	$\frac{\partial g(u)}{\partial \mathbf{x}} =$	$\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial \mathbf{x}}$	
6	$u = u(\mathbf{x})$	$\frac{\partial f(g(u))}{\partial \mathbf{x}} =$	$\frac{\partial f(g)}{\partial g}\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial \mathbf{x}}$	
7	$\mathbf{u} = \mathbf{u}(\mathbf{x})$, $\mathbf{v} = \mathbf{v}(\mathbf{x})$	$\frac{\partial (\mathbf{u}\cdot\mathbf{v})}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}^{\top}\mathbf{v}}{\partial \mathbf{x}} =$	$\mathbf{u}^{\top}\frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^{\top}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$ • assumes numerator layout of $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\mathbf{v} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{u}$ • assumes denominator layout of $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$
8	$\mathbf{u} = \mathbf{u}(\mathbf{x})$, $\mathbf{v} = \mathbf{v}(\mathbf{x})$, $\mathbf{A}$ is not a function of $\mathbf{x}$	$\frac{\partial (\mathbf{u}\cdot\mathbf{A}\mathbf{v})}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}^{\top}\mathbf{A}\mathbf{v}}{\partial \mathbf{x}} =$	$\mathbf{u}^{\top}\mathbf{A}\frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^{\top}\mathbf{A}^{\top}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$ • assumes numerator layout of $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\mathbf{A}\mathbf{v} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{A}^{\top}\mathbf{u}$ • assumes denominator layout of $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$
9		$\frac{\partial^2 f}{\partial \mathbf{x}\partial \mathbf{x}^T} =$		$\mathbf{H}$, the Hessian matrix[5]
10	$\mathbf{a}$ is not a function of $\mathbf{x}$	$\frac{\partial (\mathbf{a}\cdot\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial (\mathbf{x}\cdot\mathbf{a})}{\partial \mathbf{x}} =$ $\frac{\partial \mathbf{a}^{\top}\mathbf{x}}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^{\top}\mathbf{a}}{\partial \mathbf{x}} =$	$\mathbf{a}^{\top}$	$\mathbf{a}$
11	$\mathbf{A}$ is not a function of $\mathbf{x}$ $\mathbf{b}$ is not a function of $\mathbf{x}$	$\frac{\partial \mathbf{b}^{\top}\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{b}^{\top}\mathbf{A}$	$\mathbf{A}^{\top}\mathbf{b}$
12	$\mathbf{A}$ is not a function of $\mathbf{x}$	$\frac{\partial \mathbf{x}^{\top}\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{x}^{\top}(\mathbf{A}+\mathbf{A}^{\top})$	$(\mathbf{A}+\mathbf{A}^{\top})\mathbf{x}$
13	$\mathbf{A}$ is not a function of $\mathbf{x}$ $\mathbf{A}$ is symmetric	$\frac{\partial \mathbf{x}^{\top}\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$2\mathbf{x}^{\top}\mathbf{A}$	$2\mathbf{A}\mathbf{x}$
14	$\mathbf{A}$ is not a function of $\mathbf{x}$	$\frac{\partial^2 \mathbf{x}^{\top}\mathbf{A}\mathbf{x}}{\partial \mathbf{x}^2} =$	$\mathbf{A}+\mathbf{A}^{\top}$	
15	$\mathbf{A}$ is not a function of $\mathbf{x}$ $\mathbf{A}$ is symmetric	$\frac{\partial^2 \mathbf{x}^{\top}\mathbf{A}\mathbf{x}}{\partial \mathbf{x}^2} =$	$2\mathbf{A}$	
16		$\frac{\partial (\mathbf{x}\cdot\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^{\top}\mathbf{x}}{\partial \mathbf{x}} =$	$2\mathbf{x}^{\top}$	$2\mathbf{x}$
17	$\mathbf{a}$ is not a function of $\mathbf{x}$, $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$\frac{\partial (\mathbf{a}\cdot\mathbf{u})}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^{\top}\mathbf{u}}{\partial \mathbf{x}} =$	$\mathbf{a}^{\top}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$ • assumes numerator layout of $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\mathbf{a}$ • assumes denominator layout of $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$
18	$\mathbf{a}, \mathbf{b}$ are not functions of $\mathbf{x}$	$\frac{\partial\, \mathbf{a}^{\top}\mathbf{x}\mathbf{x}^{\top}\mathbf{b}}{\partial\, \mathbf{x}} =$	$\mathbf{x}^{\top}(\mathbf{a}\mathbf{b}^{\top}+\mathbf{b}\mathbf{a}^{\top})$	$(\mathbf{a}\mathbf{b}^{\top}+\mathbf{b}\mathbf{a}^{\top})\mathbf{x}$
19	$\mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{D}, \mathbf{e}$ are not functions of $\mathbf{x}$	$\frac{\partial\, (\mathbf{A}\mathbf{x}+\mathbf{b})^{\top}\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{e})}{\partial\, \mathbf{x}} =$	$(\mathbf{D}\mathbf{x}+\mathbf{e})^{\top}\mathbf{C}^{\top}\mathbf{A} + (\mathbf{A}\mathbf{x}+\mathbf{b})^{\top}\mathbf{C}\mathbf{D}$	$\mathbf{D}^{\top}\mathbf{C}^{\top}(\mathbf{A}\mathbf{x}+\mathbf{b}) + \mathbf{A}^{\top}\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{e})$
20	$\mathbf{a}$ is not a function of $\mathbf{x}$	$\frac{\partial\, \|\mathbf{x}-\mathbf{a}\|}{\partial\, \mathbf{x}} =$	$\frac{(\mathbf{x}-\mathbf{a})^{\top}}{\|\mathbf{x}-\mathbf{a}\|}$	$\frac{\mathbf{x}-\mathbf{a}}{\|\mathbf{x}-\mathbf{a}\|}$

上述的求解过程称为**最小二乘法**（ordinary least squares），简称**OLS**。

Python中对于矩阵的各种操作可以通过Numpy库的一些方法来实现，非常方便。但在这个代码实现中需要注意：**X矩阵不能为奇异矩阵，否则是无法求解矩阵的逆的。**

【补充】

名称	英文	公式	别称	英文
残差平方和SSE	Sum of Squares for Error	SSE = $\sum_{i=1}^{m}(y_i-\hat{y}_i)^2$	剩余平方和RSS	residual sum of squares
回归平方和SSR	Sum of Squares for Regression	SSR = $\sum_{i=1}^{m}(\hat{y}_i-\bar{y})^2$	解释平方和ESS	explained sum of squares
总离差平方和SST	Sum of Squares for Total	SST = $\sum_{i=1}^{m}(y_i-\bar{y})^2$	总离差平方和TSS	total sum of squares

三者之间的关系为：

$$SST = SSR + SSE$$

和轮廓系数类似，最终的决定系数指标也同时结合了“组内误差”和“组间误差”两个指标

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

决定系数分布在[0, 1]区间内，且越趋近于1，表明拟合程度越好。

4. 简单线性回归的python实现

4.1 导入相关包

```
import numpy as np
import pandas as pd
import random
import matplotlib as mpl
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['simhei'] #显示中文
plt.rcParams['axes.unicode_minus']=False   #用来正常显示负号
%matplotlib inline
```

4.2 导入数据集并探索数据

```
ex0 = pd.read_table('ex0.txt',header=None)
ex0.head()
ex0.shape
ex0.describe()
```

4.3 构建辅助函数

```
"""函数功能：输入DF数据集（最后一列为标签），返回特征矩阵和标签矩阵"""
def get_Mat(dataSet):
    xMat = np.mat(dataSet.iloc[:,:-1].values)
    yMat = np.mat(dataSet.iloc[:,-1].values).T
    return xMat,yMat
```

```
#查看函数运行结果
xMat,yMat = get_Mat(ex0)
```

```
"""函数功能：数据集可视化"""
def plotShow(dataSet):
    xMat,yMat=get_Mat(dataSet)
    plt.scatter(xMat.A[:,1],yMat.A,c='b',s=5)
    plt.show()
```

```
#可视化ex0数据集
plotShow(ex0)
```

4.5 计算回归系数

```
"""
函数功能：计算回归系数
参数说明：
    dataSet：原始数据集
返回：
    ws：回归系数
"""
def standRegres(dataSet):
    xMat,yMat =get_Mat(dataSet)
    xTx = xMat.T*xMat
    if np.linalg.det(xTx)==0:
        print('矩阵为奇异矩阵，无法求逆')
        return
    ws=xTx.I*(xMat.T*yMat)
    return ws
```

说明：det(A)指的是矩阵A的行列式（determinant），如果det(A)=0，则说明矩阵A是奇异矩阵，不可逆。

```
ws = standRegres(ex0)
ws
```

4.6 绘制最佳拟合直线

```
"""
函数功能：绘制散点图和最佳拟合直线
"""
def plotReg(dataSet):
    xMat,yMat=get_Mat(dataSet)
    plt.scatter(xMat.A[:,1],yMat.A,c='b',s=5)
    ws = standRegres(dataSet)
    yHat = xMat*ws
    plt.plot(xMat[:,1],yHat,c='r')
    plt.show()
```

```
#绘制ex0数据集的散点图和最佳拟合直线
plotReg(ex0)
```

4.7 计算相关系数

在python中，Numpy库提供了相关系数的计算方法：可以通过函数np.corrcoef（yEstimate,yActual）来计算预测值和真实值之间的相关性。这里需要保证的是，输入的两个参数都是**行向量**。

```
xMat,yMat =get_Mat(ex0)
yHat = xMat*ws
ws =standRegres(ex0)
np.corrcoef(yHat.T,yMat.T) #保证两个都是行向量
```

该矩阵包含所有两两组合的相关系数。可以看到，对角线上全部为1.0，因为自身匹配肯定是最完美的，而yHat和yMat的相关系数为0.98。看起来似乎是一个不错的结果。但是仔细观察数据集，会发现数据呈现有规律的波动，但是直线似乎没有很好的捕捉到这些波动。

三、局部加权线性回归

线性回归的一个问题时有可能出现欠拟合现象，为了解决这个问题，我们可以采用的一个方法是**局部加权线性回归**(Locally Weighted Linear Regression)，简称**LWLR**。该算法思想就是给待预测点附近的每个点赋予一定的权重，然后按照简单线性回归求解w方法求解。与KNN一样，这种算法每次预测均需要实现选取出对应的数据子集。该算法解出回归系数w的形式如下：

$$\hat{w} = (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W} y$$

其中，$\boldsymbol{W}$ 是一个矩阵，用来给每个数据点赋予权重。

【公式推导过程】：

$$\begin{aligned} SSE &= \sum_{i=1}^{m} W_i (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{m} W_i (y_i - x_i^T w)^2 \end{aligned}$$

写成矩阵形式即为：

$$SSE=(y-\boldsymbol{X}w)^T\boldsymbol{W}(y-\boldsymbol{X}w)$$

对其求导：

$$\begin{aligned}
&\frac{\partial(y-\boldsymbol{X}w)^T\boldsymbol{W}(y-\boldsymbol{X}w)}{\partial w}\\
&=\frac{\partial(y-\boldsymbol{X}w)^T(\boldsymbol{W}y-\boldsymbol{W}\boldsymbol{X}w)}{\partial w}\\
&=\frac{\partial(y^T-(\boldsymbol{X}w)^T)(\boldsymbol{W}y-\boldsymbol{W}\boldsymbol{X}w)}{\partial w}\\
&=\frac{\partial(y^T-w^T\boldsymbol{X}^T)(\boldsymbol{W}y-\boldsymbol{W}\boldsymbol{X}w)}{\partial w}\\
&=\frac{\partial(y^T\boldsymbol{W}y-y^T\boldsymbol{W}\boldsymbol{X}w-w^T\boldsymbol{X}^T\boldsymbol{W}y+w^T\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}w)}{\partial w}\\
&=\frac{\partial y^T\boldsymbol{W}y}{\partial w}-\frac{\partial y^T\boldsymbol{W}\boldsymbol{X}w}{\partial w}-\frac{\partial w^T\boldsymbol{X}^T\boldsymbol{W}y}{\partial w}+\frac{\partial w^T\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}w}{\partial w}\\
&=0-\boldsymbol{X}^T\boldsymbol{W}y-\boldsymbol{X}^T\boldsymbol{W}y+2\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}w\\
&=-2\boldsymbol{X}^T\boldsymbol{W}y+2\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}w\\
&=-2\boldsymbol{X}^T\boldsymbol{W}(y-\boldsymbol{X}w)
\end{aligned}$$

令其等于0，可得：

$$\begin{aligned}
\frac{\partial(y-\boldsymbol{X}w)^T\boldsymbol{W}(y-\boldsymbol{X}w)}{\partial w}&=0\\
-2\boldsymbol{X}^T\boldsymbol{W}(y-\boldsymbol{X}w)&=0\\
\boldsymbol{X}^T\boldsymbol{W}(y-\boldsymbol{X}w)&=0\\
\boldsymbol{X}^T\boldsymbol{W}y-\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}w&=0\\
\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}w&=\boldsymbol{X}^T\boldsymbol{W}y
\end{aligned}$$

当$|\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}|\neq 0$时，有：

$$\hat{w}=(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}y$$

此处$\hat{w}$指的是当前可以估计出的w的最优解，$\boldsymbol{W}$是一个矩阵，用来给每个数据点赋予权重你。

LWLR使用"核"（与支持向量机中的核类似）来对附近的点赋予更高的权重。

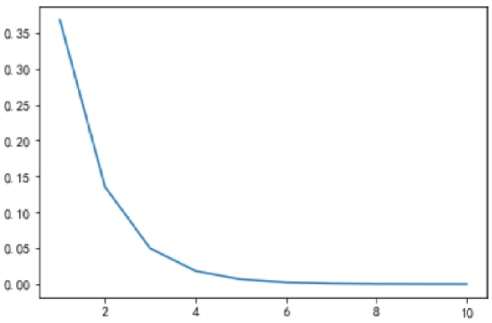
【说明】支持向量机中的核函数用来计算高维空间中的点积，而**LWLR**中的核仅仅用来给数据点赋予权重。

核的类型可以自由选择，最常用的就是**高斯核**，高斯核对应的权重如下：

$$w(i,i)=exp\left[\frac{|x^i-x|^2}{-2k^2}\right]$$

这样我们就构建了一个只含对角元素的权重矩阵$\boldsymbol{W}$，并且点x与x(i)越近，w(i,i)将会越大。

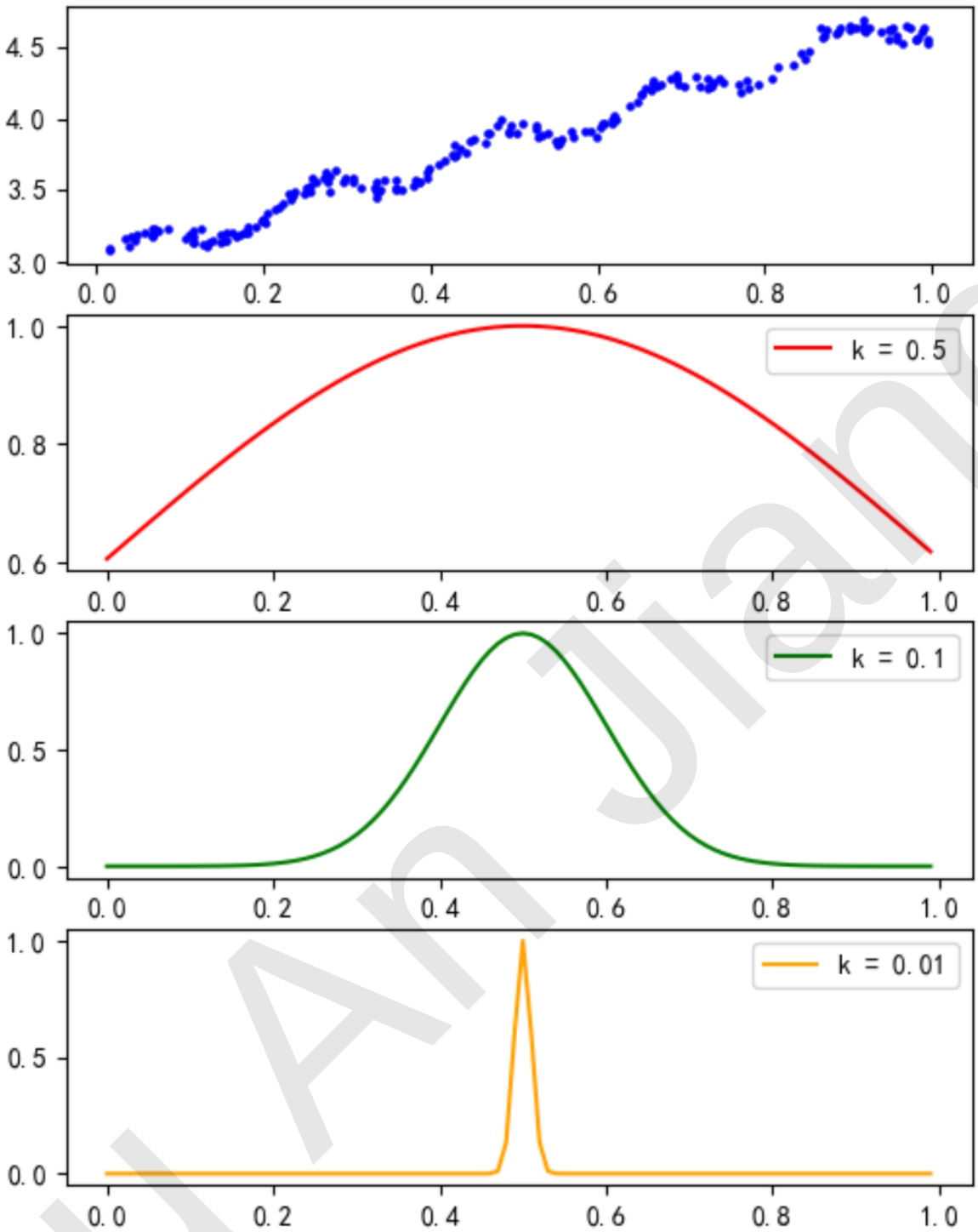
```
a = np.arange(1,11,1)
plt.plot(a,np.exp(-a));
```



这个公式包含一个需要用户指定的参数k,它决定了对附近的点赋予多大的权重，这也是使用局部加权线性回归（LWLR）时唯一需要考虑的参数。下面我们来看一下不同参数k与权重的关系。

```
#此段代码供大家参考
xMat,yMat = get_Mat(ex0)
x=0.5
xi = np.arange(0,1.0,0.01)
k1,k2,k3=0.5,0.1,0.01
w1 = np.exp((xi-x)**2/(-2*k1**2))
w2 = np.exp((xi-x)**2/(-2*k2**2))
w3 = np.exp((xi-x)**2/(-2*k3**2))
#创建画布
fig = plt.figure(figsize=(6,8),dpi=100)
#子画布1，原始数据集
fig1 = fig.add_subplot(411)
plt.scatter(xMat.A[:,1],yMat.A,c='b',s=5)
#子画布2，w=0.5
fig2 = fig.add_subplot(412)
plt.plot(xi,w1,color='r')
plt.legend(['k = 0.5'])
#子画布3，w=0.1
fig3 = fig.add_subplot(413)
plt.plot(xi,w2,color='g')
plt.legend(['k = 0.1'])
#子画布4，w=0.01
fig4 = fig.add_subplot(414)
plt.plot(xi,w3,color='orange')
plt.legend(['k = 0.01'])
plt.show()
```

运行结果如下所示：



这里假定我们预测的点是x=0.5，最上面的图是原始数据集，从下面三张图可以看出随着k的减小，被用于训练模型的数据点越来越少。

1. 构建LWLR函数

这个过程与简单线性函数的基本一致，唯一不同的是加入了权重weights，这里我将权重参数求解和预测yHat放在了一个函数里面。

```
"""
```

```
    函数功能：计算局部加权线性回归的预测值
    参数说明：
        testMat:测试集
        xMat:训练集的特征矩阵
        yMat:训练集的标签矩阵
    返回：
        yHat：函数预测值
    """
    def LWLR(testMat,xMat,yMat,k=1.0):
        n=testMat.shape[0]
        m=xMat.shape[0]
        weights =np.mat(np.eye(m))
        yHat = np.zeros(n)
        for i in range(n):
            for j in range(m):
                diffMat = testMat[i]-xMat[j]
                weights[j,j]=np.exp(diffMat*diffMat.T/(-2*k**2))
            xTx = xMat.T*(weights*xMat)
            if np.linalg.det(xTx)==0:
                print('矩阵为奇异矩阵，不能求逆')
                return
            ws = xTx.I*(xMat.T*(weights*yMat))
            yHat[i]= testMat[i]*ws
        return ws,yHat
```

2. 不同k值的结果图

我们调整k值，然后查看不同k值对模型的影响

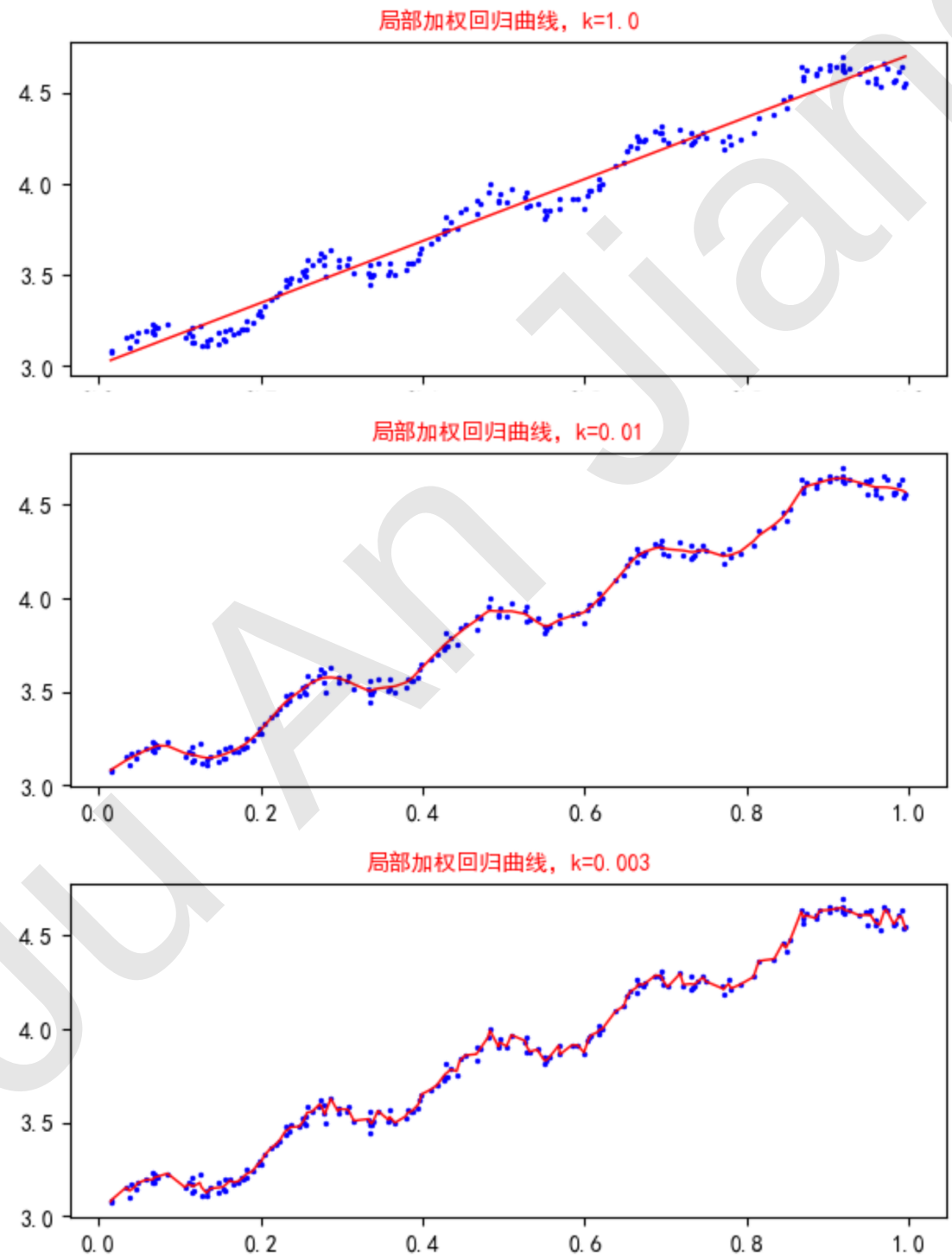
```
xMat,yMat = get_Mat(ex0)
#将数据点排列（argsort()默认升序排列，返回索引）
srtInd = xMat[:,1].argsort(0)
xSort=xMat[srtInd][:,0]

#计算不同k取值下的y估计值yHat
ws1,yHat1 = LWLR(xMat,xMat,yMat,k=1.0)
ws2,yHat2 = LWLR(xMat,xMat,yMat,k=0.01)
ws3,yHat3 = LWLR(xMat,xMat,yMat,k=0.003)

#创建画布
fig = plt.figure(figsize=(6,8),dpi=100)
#子图1绘制k=1.0的曲线
fig1=fig.add_subplot(311)
plt.scatter(xMat[:,1].A,yMat.A,c='b',s=2)
plt.plot(xSort[:,1],yHat1[srtInd],linewidth=1,color='r')
plt.title('局部加权回归曲线，k=1.0',size=10,color='r')
#子图2绘制k=0.01的曲线
fig2=fig.add_subplot(312)
plt.scatter(xMat[:,1].A,yMat.A,c='b',s=2)
plt.plot(xSort[:,1],yHat2[srtInd],linewidth=1,color='r')
plt.title('局部加权回归曲线，k=0.01',size=10,color='r')
```

```
#子图3绘制k=0.003的曲线
fig3=fig.add_subplot(313)
plt.scatter(xMat[:,1].A,yMat.A,c='b',s=2)
plt.plot(xSort[:,1],yHat3[srtInd],linewidth=1,color='r')
plt.title('局部加权回归曲线，k=0.003',size=10,color='r')
#调整子图的间距
plt.tight_layout(pad=1.2)
plt.show()
```

运行结果如下：



这三个图是不同平滑值绘出的局部加权线性回归结果。当k=1.0时，模型的效果与最小二乘法差不多；k=0.01时，该模型基本上已经挖出了数据的潜在规律，当继续减小到k=0.003时，会发现模型考虑了太多的噪音，进而导致了过拟合现象。

```
#四种模型相关系数比较
np.corrcoef(yHat.T,yMat.T)    #最小二乘法
np.corrcoef(yHat1,yMat.T)     #k=1.0模型
np.corrcoef(yHat2,yMat.T)     #k=0.01模型
np.corrcoef(yHat3,yMat.T)     #k=0.003模型
```

局部加权线性回归也存在一个问题——增加了计算量，因为它对每个点预测都要使用整个数据集。从不同k值的结果图中可以看出，当k=0.01时模型可以很好地拟合数据潜在规律，但是同时看一下，k值与权重关系图，可以发现，当k=0.01时，大部分数据点的权重都接近0，也就是说他们基本上可以不用带入计算。所以如果一开始就能去掉这些数据点的计算，那么就可以大大减少程序的运行时间了，从而缓解计算量增加带来的问题。后面我们会讲解这个操作。

四、案例：预测鲍鱼的年龄

接下来，我们将回归用于真实数据。此案例所用数据集来自UCI数据集，记录了鲍鱼（一种介壳类水生生物）的一些相关属性，根据这些属性来预测鲍鱼的年龄。

鲍鱼年龄的计算主要是通过一些比较容易获得的量测数据，以及通过锥体切割贝壳，染色并在显微镜下数出环数来确定的 - 这是一项无聊且耗时的任务（数据提供者的吐槽……）。

特征名	数据类型	单位	描述
性别	离散型	——	公，母，婴儿（1,-1,0）
长度	连续型	毫米	贝壳最长的部分
直径	连续型	毫米	垂直于长度
高度	连续型	毫米	壳里的肉的高度
整体重量	连续型	克	整个鲍鱼的重量
肉重量	连续型	克	鲍鱼肉的重量
内脏重量	连续型	克	内脏的重量 (去血后)
壳重	连续型	克	干了之后的壳重
年龄	整数型	——	鲍鱼的年龄

1. 导入数据集

```
abalone = pd.read_table('abalone.txt',header=None)
abalone.columns=['性别','长度','直径','高度','整体重量','肉重量','内脏重量','壳重','年龄']
abalone.head()
abalone.shape
abalone.info()
abalone.describe()
```

2. 查看数据分布状况

```
"""
函数功能：绘制散点图来查看特征和标签的数据分布
"""
def dataPlot(dataSet):
    m,n=dataSet.shape
    fig = plt.figure(figsize=(8,20),dpi=100)
    colormap = mpl.cm.rainbow(np.linspace(0, 1, n))
    for i in range(n):
        fig_ = fig.add_subplot(n,1,i+1)
        plt.scatter(range(m),dataSet.iloc[:,i].values,s=2,c=colormap[i])
        plt.title(dataSet.columns[i])
        plt.tight_layout(pad=1.2)
```

运行函数，查看数据分布：

```
dataPlot(abalone)
```

可以从数据分布散点图中看出：

1）除“性别”之外，其他数据明显存在规律性排列

2）“高度”这一特征中，有两个异常值

从看到的现象，我们可以采取以下两种措施：

1）切分训练集和测试集时，需要打乱原始数据集来进行随机挑选

2）剔除"高度"这一特征中的异常值

```
#剔除高度特征中≥0.4的异常值
aba = abalone.loc[abalone['高度']<0.4,:]

#再次查看数据集的分布
dataPlot(aba)
```

2. 切分训练集和测试集

```
"""
函数功能：随机切分训练集和测试集
参数说明：
    dataSet：原始数据集
    rate:训练集比例
返回：
    train,test：切分好的训练集和测试集
"""
def randSplit(dataSet,rate):
    l = list(dataSet.index)
    random.seed(123)
    random.shuffle(l)
    dataSet.index = l
    m = dataSet.shape[0]
    n = int(m*rate)
    train = dataSet.loc[range(n),:]
    test = dataSet.loc[range(n,m),:]
    test.index=range(test.shape[0])
    dataSet.index =range(dataSet.shape[0])
    return train,test
```

```
train,test = randSplit(aba,0.8)
#探索训练集
train.head()
train.shape
train.describe()  #统计描述
dataPlot(train)   #查看训练集数据分布

#探索测试集
test.head()
test.shape
test.describe()
dataPlot(test)
```

3. 构建辅助函数

```
"""
函数功能：计算误差平方和SSE
参数说明：
    dataSet：真实值
    regres：求回归系数的函数
返回：
    SSE：误差平方和
"""
def sseCal(dataSet, regres):
    xMat,yMat = get_Mat(dataSet)
    ws = regres(dataSet)
    yHat = xMat*ws
    sse = ((yMat.A.flatten() - yHat.A.flatten())**2).sum()
    return sse
```

以ex0数据集为例，查看函数运行结果：

```
#简单线性回归的SSE
sseCal(ex0, standRegres)
```

构建相关系数R2计算函数

```
"""
函数功能：计算相关系数R2
"""
def rSquare(dataSet,regres):
    xMat,yMat=get_Mat(dataSet)
    sse = sseCal(dataSet,regres)
    sst = ((yMat.A-yMat.mean())**2).sum()
    r2 = 1 - sse / sst
    return r2
```

同样以ex0数据集为例，查看函数运行结果：

```
#简单线性回归的R2
rSquare(ex0, standRegres)
```

4. 构建加权线性模型

因为数据量太大，计算速度极慢，所以此处选择训练集的前100个数据作为训练集，测试集的前100个数据作为测试集。

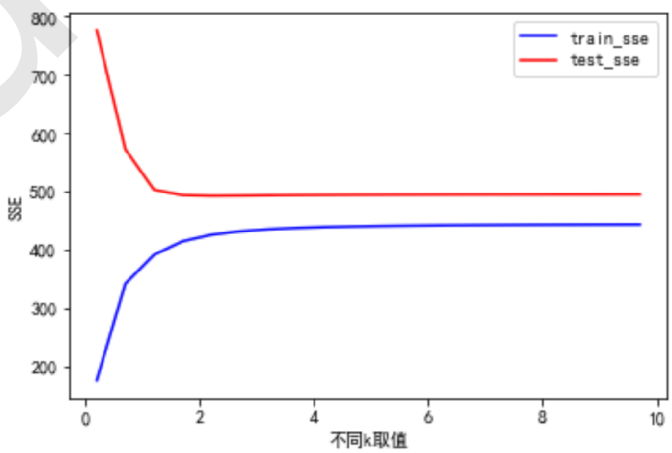
```
"""
函数功能：绘制不同k取值下，训练集和测试集的SSE曲线
"""
def ssePlot(train,test):
    X0,Y0 = get_Mat(train)
    X1,Y1 =get_Mat(test)
    train_sse = []
    test_sse = []
    for k in np.arange(0.2,10,0.5):
        ws1,yHat1 = LWLR(X0[:99],X0[:99],Y0[:99],k)
        sse1 = ((Y0[:99].A.T - yHat1)**2).sum()
        train_sse.append(sse1)

        ws2,yHat2 = LWLR(X1[:99],X0[:99],Y0[:99],k)
        sse2 = ((Y1[:99].A.T - yHat2)**2).sum()
        test_sse.append(sse2)

    plt.plot(np.arange(0.2,10,0.5),train_sse,color='b')
    plt.plot(np.arange(0.2,10,0.5),test_sse,color='r')
    plt.xlabel('不同k取值')
    plt.ylabel('SSE')
    plt.legend(['train_sse','test_sse'])
```

运行结果：

```
ssePlot(train,test)
```



这个图的解读应该是这样的：从右往左看，当K取较大值时，模型比较稳定，随着K值的减小，训练集的SSE开始逐渐减小，当K取到2左右，训练集的SSE与测试集的SSE相等，当K继续减小时，训练集的SSE也越来越小，也就是说，模型在训练集上的表现越来越好，但是，模型在测试集上的表现却越来越差了，这就说明模型开始出现过拟合了。其实，这个图与前面不同k值的结果图是吻合的，K=1.0, 0.01, 0.003这三张图也表明随着K的减小，模型会逐渐出现过拟合。所以这里可以看出，K在2左右的取值最佳。

我们再将K=2带入局部线性回归模型中，然后查看预测结果：

```
train,test = randSplit(aba,0.8)
trainX,trainY = get_Mat(train)
testX,testY = get_Mat(test)
ws0,yHat0 = LWLR(testX,trainX,trainY,k=2)
```

绘制真实值与预测值之间的关系图

```
y=testY.A.flatten()
plt.scatter(y,yHat0,c='b',s=5);
```

封装一个函数来计算SSE和R方，方便后续调用

```
"""
函数功能：计算加权线性回归的SSE和R方
"""
def LWLR_pre(dataSet):
    train,test = randSplit(dataSet,0.8)
    trainX,trainY = get_Mat(train)
    testX,testY = get_Mat(test)
    ws,yHat = LWLR(testX,trainX,trainY,k=2)
    sse = ((testY.A.T - yHat)**2).sum()
    sst = ((testY.A-testY.mean())**2).sum()
    r2 = 1 - sse / sst
    return sse,r2
```

查看模型预测结果

```
LWLR_pre(aba)
```

从结果可以看出，SSE达4000+，相关系数只有0.52，模型效果并不是很好。

【此段代码供参考】通过剔除3倍标准差之外的数据，重新建模看看SSE是否有降低。（注意这段代码要一行一行运行，否则会报错，无法出现你想要的结果）

```
#计算标签的3倍标准差
aba['年龄'].min()
aba['年龄'].max()
delt0 = aba['年龄'].mean()-3*aba['年龄'].std()
delt1 = aba['年龄'].mean()+3*aba['年龄'].std()
#剔除3倍标准差之外的数据
aba1=aba.loc[(aba['年龄']>delt0)&(aba['年龄']<delt1),:]
```

```
aba1.shape
#计算预测结果
train1,test1 = randSplit(aba1,0.8)
trainX1,trainY1 = get_Mat(train1)
testX1,testY1 = get_Mat(test1)
yHat0_1 = LWLR(testX1,trainX1,trainY1,k=2)
#绘制真实值与预测值之间的关系图
y1=testY1.A.flatten()
plt.scatter(y1,yHat0_1,c='b',s=5);
#计算SSE和R2
LWLR_pre(aba1)
```

运行了这段代码你会发现，SSE减小到了3000+，但相关系数R2还是只有0.52。

五、岭回归

在前面线性回归的求解过程中，我们使用最小二乘法来求解最优解，但是其中有一个隐形的条件：$\boldsymbol{X}^T\boldsymbol{X}$ 的行列式 $|\boldsymbol{X}^T\boldsymbol{X}| \neq 0$，也就是说必须要求$\boldsymbol{X}^T\boldsymbol{X}$ 为满秩矩阵或者正定阵。

矩阵知识	矩阵知识点回顾 (以非零m*n矩阵A为例)
秩	矩阵A的**列秩**是A的最大线性无关列集合的大小，**行秩**是A的最大线性无关行集合的大小。 对于任意矩阵A，其行秩等于其列秩，可以统称为**秩**。 所以A的秩是[0，min(m,n)]内的整数。
满秩	一个矩阵A是列满秩的，当且仅当该矩阵不存在空向量。
正定矩阵	对于任意列满秩的矩阵A，矩阵A^TA是正定的。

1. 什么是岭回归

岭回归（Ridge Regression）是线性回归的一种进阶算法，它最先主要用来处理那种特征比样本点还多，也就是说，输入数据的矩阵**X** 不是满秩矩阵的情况，现在也用于估计中加入偏差，从而得到更好的估计。岭回归的算法核心是：在矩阵$\boldsymbol{X}^T\boldsymbol{X}$ 上加一个$\lambda\boldsymbol{I}$ 从而使得矩阵非奇异，进而能对$\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I}$ 求逆。假如我们的数据集有m个样本点n个特征，回归系数的计算公式变为：

$$\hat{w} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T y$$

其中λ是一个人为定义的参数，矩阵$\boldsymbol{I}$ 是一个m*m的单位矩阵。对角线上全是1，其他元素全是0.

这里通过引入λ来限制了所有w之和，λ可称之为惩罚项，它能够减少不重要的参数，这个技术在统计学中也叫作**缩减（shrinkage）**。

岭回归中"岭"的由来
岭回归中使用了单位矩阵乘以常亮λ，观察单位矩阵可以发现：数值1贯穿了整个对角线，其余元素全是0。形象地，在全是0构成的平面上有一条1组成的"岭"。

2. 构建岭回归模型

接下来编写岭回归函数，此处默认设置岭回归系数为0.2，当该系数不为0的时候不会存在不可逆的情况，因此可免去if语句判别是否满秩的设置。对于单位矩阵的生成，需要借助NumPy中的eye函数，该函数需要输入对角矩阵规模参数

```
"""
函数功能：计算回归系数
参数说明：
    dataSet:原始数据集
    lam:人为设定的惩罚系数(默认0.2)
返回：
    ws:回归系数
"""
def ridgeRegres(dataSet, lam=0.2):
    xMat,yMat=get_Mat(dataSet)
    xTx = xMat.T * xMat
    denom = xTx + np.eye(xMat.shape[1])*lam
    ws = denom.I * (xMat.T * yMat)
    return ws
```

比较线性回归和岭回归计算结果

```
#回归系数比较
standRegres(aba)          #线性回归
ridgeRegres(aba)          #岭回归

#相关系数R2比较
rSquare(aba,standRegres) #线性回归
rSquare(aba,ridgeRegres) #岭回归
```

```
In [90]: ridgeRegres(aba).T

Out[90]: matrix([[  0.08198911,   5.69131271,  10.57399489,  25.34980646,
                    8.08962134, -20.08857516, -10.94149058,   6.49285968]])

In [91]: standRegres(aba).T

Out[91]: matrix([[  0.08242085,   4.39330848,  10.92381422,  29.91312222,
                    8.72554079, -20.60349624, -12.09968274,   5.54043538]])

In [92]: rSquare(aba,standRegres)

Out[92]: 0.5239481479176779

In [93]: rSquare(aba,ridgeRegres)

Out[93]: 0.5233308835711574
```

从结果可以看出，两者效果差不多，那我们改变λ的值，看看回归系数如何改变。我们可以通过绘制岭迹图来更直观的查看回归系数的变化。

3. 绘制岭迹图

这里我们需要对特征做标准化处理。因为，我们需要使每个维度特征具有相同的重要性。这里用了比较简单的标准化处理手段——**所有特征都减去各自的均值并除以方差。**

```
"""
函数功能：计算岭回归的回归系数(默认λ以指数级变化)
参数说明：
    dataSet:原始数据集
    k：λ的个数
返回：
    wMat:回归系数矩阵
"""
def ridgeTest(dataSet,k=30):
    xMat,yMat=get_Mat(dataSet)
    m,n=xMat.shape
    wMat = np.zeros((k,n))
    #特征标准化
    yMean = yMat.mean(0)
    xMeans = xMat.mean(0)
    xVar = xMat.var(0)
    yMat = yMat-yMean
    xMat = (xMat-xMeans)/xVar
    for i in range(k):
        xTx = xMat.T*xMat
        lam = np.exp(i-10)
        denom = xTx+np.eye(n)*lam
        ws=denom.I*(xMat.T*yMat)
        wMat[i,:]=ws.T
    return wMat
```

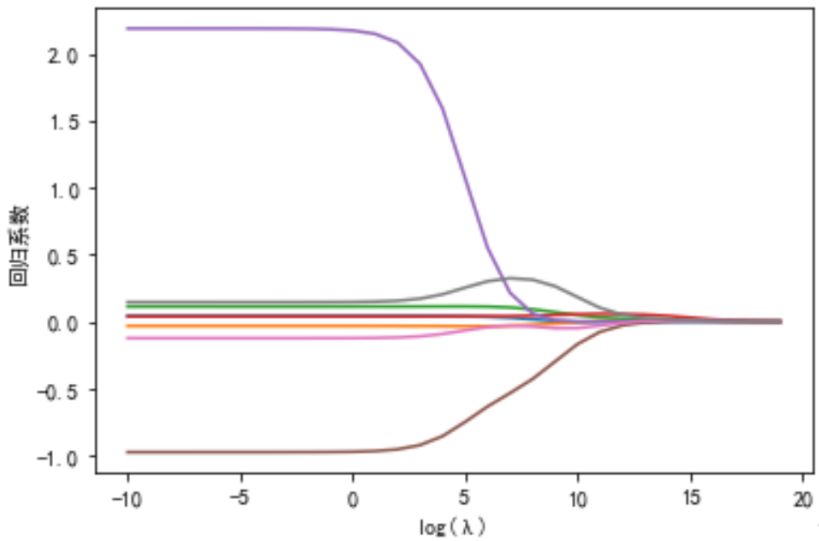
k与λ关系：

```
k = np.arange(0,30,1)
lam = np.exp(k-10)
plt.plot(lam);
```

运行函数，查看结果：

```
#回归系数矩阵
wMat = ridgeTest(aba,k=30)

#绘制岭迹图
plt.plot(np.arange(-10,20,1),wMat)
plt.xlabel('log(λ)')
plt.ylabel('回归系数');
```



通过岭迹图我们可以：

1. 观察**较佳的λ取值**：通常取喇叭口的值，大家看岭迹图可以发现，当λ增大到某个值时，所有的回归系数开始发生变化，那这个λ值就是我们要找寻的较佳惩罚系数，此时准确率会比较高（前提是数据有线性关系），并且SSE比较小，模型泛化能力会比较好。
2. 观察变量是否有**多重共线性**：随着λ的增加，回归系数会出现增大再减小的波动状况，则说明变量存在共线性，并且曲线交点越多说明共线性越强。

小Tips：

numpy中的log函数有三种：
1. np.log() 默认是以e为底，例如：e = np.exp(1)，np.log(e)=1
2. np.log2() 就是以2为底，例如：np.log2(2)=1
3. np.log10() 就是以10为底，例如：np.log10(10)=1

六、lasso

不难证明，在增加如下约束时，普通的最小二乘法回归会得到与岭回归一样的公式：

$$\sum_{k=1}^{n} w_k^2 \leq \lambda$$

这个式子限定了所有回归系数的平方和不能大于λ。使用普通的最小二乘法回归，在有共线性或者多重共线性出现的情况下，可能会出现一个值很大的正系数或者负系数。正是因为这种限制条件的存在，使用岭回归可以避免这个问题。

与岭回归类似，另一个缩减方法**LASSO**(Least Absolute Shrinkage and Selection Operator)也对回归系数做了限定，对应的约束条件如下：

$$\sum_{k=1}^{n} |w_k| \leq \lambda$$

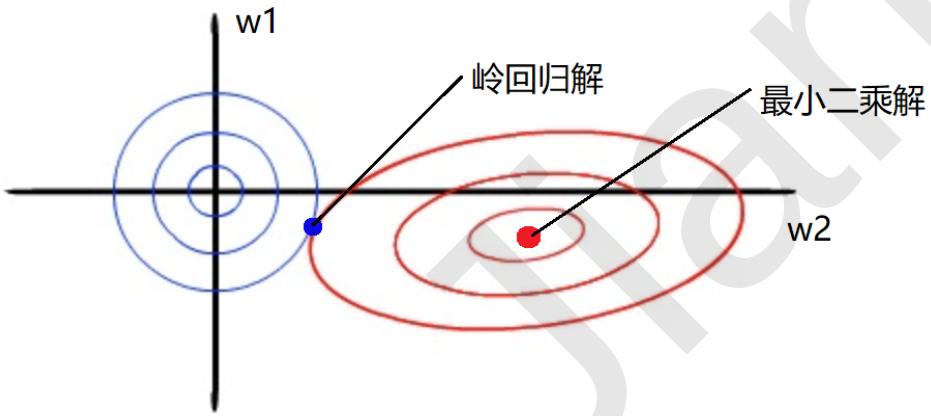
唯一不同的是，这个约束条件使用绝对值取代了平方和。虽然只是约束形式稍作变化，结果却大相径庭：在λ足够小的时候，一些系数会因此被迫缩减到0而岭回归却很难使得某个系数**恰好**缩减为0，这个特性可以帮助我们更好地理解数据。我们可以通过几何解释看到LASSO与岭回归之间的不同。

1. 岭回归和lasso的几何意义

以两个变量为例, 残差平方和可以表示为 w_1, w_2 的一个二次函数，是一个在三维空间中的抛物面，可以用等值线来表示。

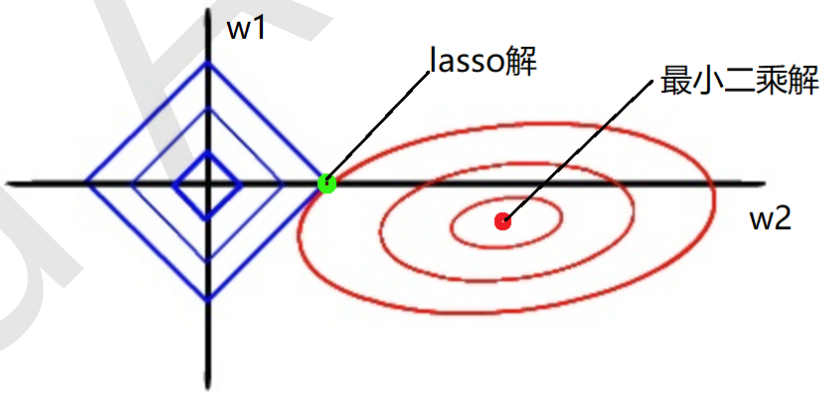
岭回归的几何解释

岭回归的限制条件 $w_1^2 + w_2^2 \leq \lambda$，相当于在二维平面的一个圆。这个时候等值线与圆相切的点便是在约束条件下的最优点，如下图所示：



lasso的几何解释

标准线性回归的损失函数还是可以用二维平面的等值线表示，而约束条件则与岭回归的圆不同，LASSO的约束条件可以用菱形表示，如下图：



相比圆，菱形的顶点更容易与抛物面相交，顶点就意味着对应的很多系数为0，而岭回归中的圆上的任意一点都很容易与抛物面相交很难得到正好等于0的系数。这也就意味着，lasso起到了很好的筛选变量的作用。

但是因为lasso的约束条件是绝对值形式，所以极大地增加了计算复杂度。这里我们使用scikit-learn调库来简单了解lasso最后返回的结果

```
#lasso是在linear_model下
from sklearn.linear_model import Lasso

las = Lasso(alpha = 0.01)   #alpha为惩罚系数，值越大惩罚力度越大
las.fit(aba.iloc[:, :-1], aba.iloc[:, -1])

las.coef_
```

七、向前逐步回归

向前逐步回归算法可以得到与lasso差不多的效果，但是更加简单。它属于一种贪心算法，即每一步都尽可能减少误差。一开始所有的权重都设置为1，然后每一步所做的决策是对某个权重增加或者减少一个很小的值。

该算法伪代码如下：

```
数据标准化，使其分布满足均值为0，方差为1
在每轮迭代中：
    设置当前最小误差lowestError为正无穷
    对每个特征：
        按步长增大或者缩小：
            改变一个系数得到一个新的w
            计算新w下的误差
            如果误差Error小于lowestError:将当前w设置为wbest
        将w设置为新的wbest
```

1. 构建辅助函数

数据标准化，使其分布满足均值为0，方差为1

```
"""
函数功能：数据标准化
"""
def regularize(xMat,yMat):
    inxMat = xMat.copy()                    #数据拷贝
    inyMat = yMat.copy()
    yMean = yMat.mean(0)                    #行与行操作，求均值
    inyMat = inyMat - yMean                 #数据减去均值
    xMeans = inxMat.mean(0)                 #行与行操作，求均值
    xVar = inxMat.var(0)                    #行与行操作，求方差
    inxMat = (inxMat - xMeans) / xVar       #数据减去均值除以方差实现标准化
    return inxMat, inyMat
```

计算SSE

```
def rssError(yMat, yHat):
    sse = ((yMat.A-yHat.A)**2).sum()
    return sse
```

2. 向前逐步线性回归

```
"""
函数说明:前向逐步线性回归
参数说明:
    dataSet:原始数据集
    eps：每次迭代需要调整的步长
    numIt：迭代次数
返回:
    wsMat：numIt次迭代的回归系数矩阵
"""
def stageWise(dataSet, eps = 0.01, numIt = 100):
    xMat0,yMat0 = get_Mat(dataSet)
    xMat,yMat = regularize(xMat0, yMat0)            #数据标准化
    m, n = xMat.shape
    wsMat = np.zeros((numIt, n))                    #初始化numIt次迭代的回归系数矩阵
    ws = np.zeros((n, 1))                           #初始化回归系数矩阵
    wsTest = ws.copy()
    wsMax = ws.copy()
    for i in range(numIt):                          #迭代numIt次
        # print(ws.T)                               #打印当前回归系数矩阵
        lowestError = np.inf                        #正无穷
        for j in range(n):                          #遍历每个特征的回归系数
            for sign in [-1, 1]:
                wsTest = ws.copy()
                wsTest[j] += eps * sign             #微调回归系数
                yHat = xMat * wsTest                #计算预测值
                sse = rssError(yMat, yHat)          #计算平方误差
                if sse < lowestError:               #如果误差更小，则更新当前的最佳回归系数
                    lowestError = sse
                    wsMax = wsTest
        ws = wsMax.copy()
        wsMat[i,:] = ws.T                           #记录numIt次迭代的回归系数矩阵
    return wsMat
```

步长设为0.01，循环200次查看结果：

```
stageWise(aba, eps = 0.01, numIt = 200)
```

从结果中看出，w1始终为0，说明它不对目标值造成任何影响，也就是说这个特征可能是不需要的。另外，从结果中可以看到，第一个权重值在0.04和0.05之间震荡，可能是步长太长了。

下面试着用更小的步长和更多的迭代次数：

```
wsMat = stageWise(aba, eps = 0.001, numIt = 5000)
wsMat
```

与最小二乘法进行比较

这里对最小二乘法的回归系数计算函数做简单修改，增加数据标准化这一步

```
def standRegres0(dataSet):
    xMat0,yMat0 =get_Mat(dataSet)
    xMat,yMat = regularize(xMat0, yMat0) #增加标准化这一步
    xTx = xMat.T*xMat
    if np.linalg.det(xTx)==0:
        print('矩阵为奇异矩阵，无法求逆')
        return
    ws=xTx.I*(xMat.T*yMat)
    yHat = xMat*ws
    return ws
```

比较两者最后结果：

```
weights = standRegres0(aba)
weights.T

wsMat[-1]
```

从结果可以看到，在5000次迭代之后向前逐步线性回归的结果与常规最小二乘法的效果相似。

绘制结果图

```
plt.plot(wsMat)
plt.xlabel('迭代次数')
plt.ylabel('回归系数');
```

逐步线性回归算法的优点在于它可以帮助人们理解有的模型并做出改进。当构建了一个模型后，可以运行该算法找出重要的特征，这样就有可能及时停止对那些不重要特征的收集。

八、案例：预测乐高玩具套装的价格

乐高（LEGO）公司生产拼装类玩具，由很多大小不同的塑料插块组成。一般来说，这些插块都是成套出售，它们可以拼装成很多不同的东西，如船、城堡、一些著名建筑等。乐高公司每个套装包含的部件数目从10件到5000件不等。乐高示意图如下所示：

一种乐高套件基本上在几年后就会停产，但乐高的收藏者之间仍会在停产后彼此交易。所以这里我们使用回归方法对**收藏者之间的交易价格进行预测。**

1. 获取数据

书中使用的方法是通过Google提供的API进行获取数据，但是现在这个API已经关闭，我们无法通过API获取数据了。不过幸运的是，我在网上找到了书上用到的那些html文件。我把它们下载下来放在了lego文件夹中。

这里使用的是爬虫技术，由于用的是非联网的网页，会比我们实际使用爬虫的时候少了一些步骤，比如自动打开网页、最大化网页之类的内容，但是过程基本上是一样的。另外，爬虫技术中，解析数据的方式有很多种，比如BeautifulSoup、Xpath等，这里我用BeautifulSoup来给大家讲解。

```
from bs4 import BeautifulSoup
"""
函数功能：抓取一个网页的信息
参数说明：
    data:用来盛放抓取的所有信息
    infile：html文件名
    yr：年份
    numPce：部件数目
    origPrc：出厂价格
"""

def scrapePage(data, infile, yr, numPce, origPrc):
    HTML_DOC = open(infile,encoding = 'utf-8').read()
    soup = BeautifulSoup(HTML_DOC,'lxml')
    i=1
    #根据HTML页面结构进行解析
    currentRow = soup.find_all('table', r = f'{i}')
    while(len(currentRow) != 0):
        currentRow = soup.find_all('table', r = f'{i}')
```

```
        title = currentRow[0].find_all('a')[1].text
        lwrTitle = title.lower()
        #查找是否有全新标签
        if (lwrTitle.find('new') > -1):
            newFlag = 1
        else:newFlag = 0
        #查找是否已经标志出售，我们只收集已出售的数据
        soldbutt = currentRow[0].find_all('td')[3].find_all('span')
        if len(soldbutt) == 0:
            print(f"商品 #{i} 没有出售")
        else:
            #解析页面获取当前价格
            soldPrice = currentRow[0].find_all('td')[4]
            priceStr = soldPrice.text
            priceStr = priceStr.replace('$','')
            priceStr = priceStr.replace(',','')
            if len(soldPrice) > 1:
                priceStr = priceStr.replace('Free shipping', '')
            sellingPrice = float(priceStr)
            #去掉不完整的套装价格
            if  sellingPrice > origPrc * 0.5:
                data.append([yr, numPce, newFlag, origPrc,sellingPrice])
        i+=1
        currentRow = soup.find_all('table', r = f'{i}')
```

我们有6种乐高的网页数据，依次爬取：

```
def setDataCollect(data):
    scrapePage(data, 'lego/lego8288.html', 2006, 800, 49.99)
    scrapePage(data, 'lego/lego10030.html', 2002, 3096, 269.99)
    scrapePage(data, 'lego/lego10179.html', 2007, 5195, 499.99)
    scrapePage(data, 'lego/lego10181.html', 2007, 3428, 199.99)
    scrapePage(data, 'lego/lego10189.html', 2008, 5922, 299.99)
    scrapePage(data, 'lego/lego10196.html', 2009, 3263, 249.99)
```

运行函数，查看结果

```
data = []
setDataCollect(data)
data
```

2. 建立模型

对原始数据集进行处理：

```
#把数据变为dataframe形式
df = pd.DataFrame(data)
df.columns = ['出品年份','部件数目','全新否','原价','二手售价']
df.head()

#探索数据
```

```
df.info()
df.describe()

#在第0列增加常数项特征X0=1
col_name = df.columns.tolist()
col_name.insert(0,'X0')
df = df.reindex(columns=col_name)
df['X0']=1
df.head()
```

用简单线性回归计算回归系数：

```
ws = standRegres(df)
ws
```

计算模型预测效果：

```
xMat,yMat = get_Mat(df)
yHat = xMat*ws

#画出真实值和预测值的散点图
plt.scatter(range(df.shape[0]),yMat.A)
plt.scatter(range(df.shape[0]),yHat.A);
```

从散点图中可以看出，模型将价格分段预测，看起来效果还不错。

所以我们最后得到的模型就是：

$$\hat{y} = 55254.34 - 27.56 * 出品年份 - 0.03 * 部件数目 - 12.29 * 全新否 + 2.57 * 原价$$

虽然说我们这个模型效果还不错，但是从实际意义上讲，模型的解释性不是很好。从公式中看，模型对"出品年份"、"部件数目"和"全新否"这三个特征进行了惩罚，出品年份还比较好解释：年份越大说明距现在时间越短，二手成交价就会有所降低，但是部件数目和是否全新这两个特征就不太好解释了，部件数越多二手成交价反而越低？全新的套装成交价反而越低？

再来看看我们的相关系数R2：

```
#简单线性回归
rSquare(df,standRegres)
#岭回归
rSquare(df,ridgeRegres)
```

用sklearn中的树回归跑一遍，看看效果如何：

```
from sklearn.tree import DecisionTreeRegressor as DTR
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
dtr = DTR().fit(X, y)
dtr.score(X, y)
```

其他

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 下周一（2018/12/24）将讲解**树回归**，欢迎各位进入菊安酱的直播间观看直播
- 如有问题，可以给我留言哦~