**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。

例如:

y = 0.0015 \* x1 - 0.99 \* x2

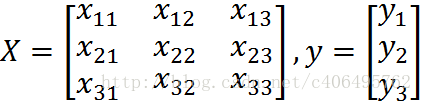
这就是所谓的回归方程，其中的0.0015和-0.99称为回归系数，求这些回归系数的过程就是回归。一旦有了这些回归系数，再给定输入，做预测就非常容易了。

具体的做法是用回归系数乘以输入值，再将结果全部加在一起，就得到了预测值。

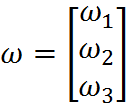
## 

## **用线性回归找到最佳拟合直线**

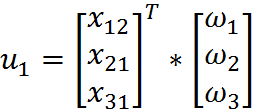
假定输入数据存放在矩阵X中，结果存放在向量y中：



而回归系数存放在向量w中：



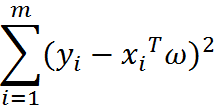
那么对于给定的数据x1，即矩阵X的第一列数据，预测结果u1：



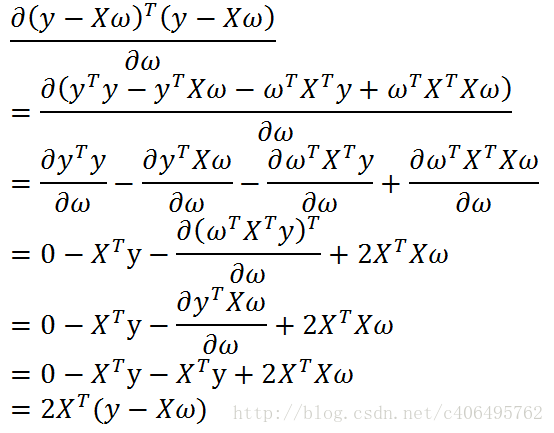
怎么才能找到w呢？

找出使误差最小的w。误差是指预测u值和真实y值之间的差值，使用该误差的简单累加将使得正差值和负差值相互抵消，所以我们采用平方误差。

平方误差和：



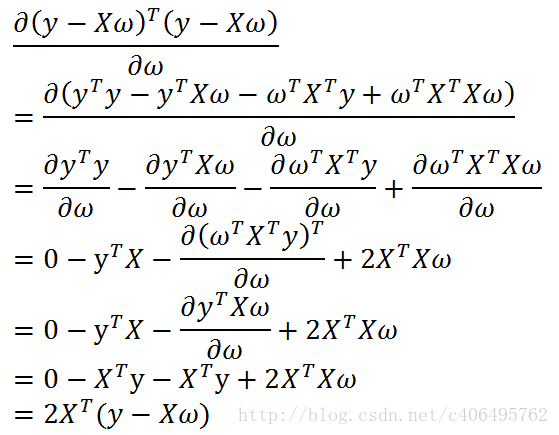
用矩阵表示还可以写做：



若x为向量，则默认x为列向量，x^T为行向量。

在继续推导之前，我们要先明确一个目的：找到w，使平方误差和最小。因为我们认为平方误差和越小，说明线性回归拟合效果越好。

现在，用矩阵表示的平方误差和对w进行求导：



令上述公式等于0，得到：

IMG_260

w上方的小标记表示，这是当前可以估计出的w的最优解。

公式中包含逆矩阵，也就是说，这个方程只在逆矩阵存在的时候使用，也即是这个矩阵是一个方阵，并且其行列式不为0。

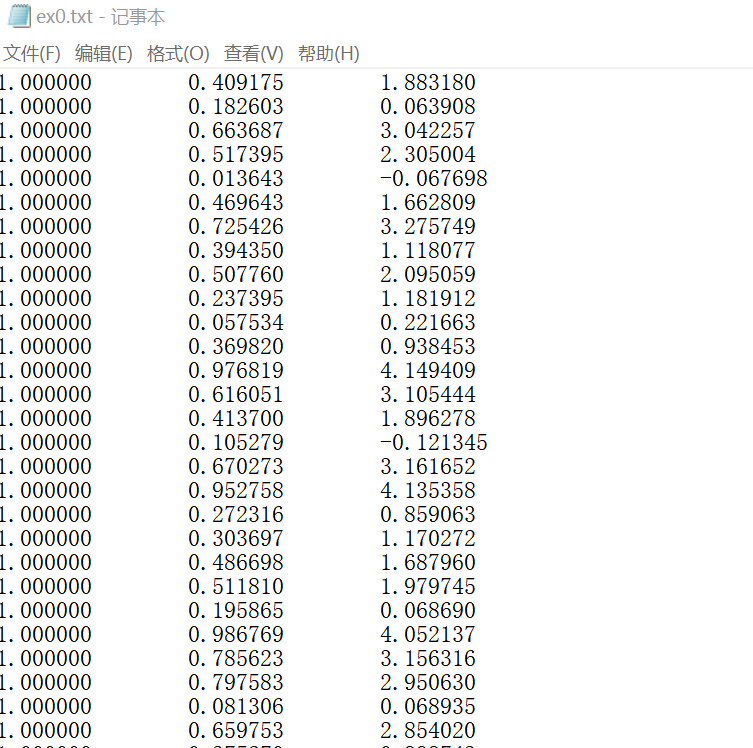
通过调用NumPy库里的矩阵方法，我们可以仅使用几行代码就完成所需功能。该方法也称作OLS， 意思是“普通小二乘法”（ordinary least squares）。

**Linear.py**

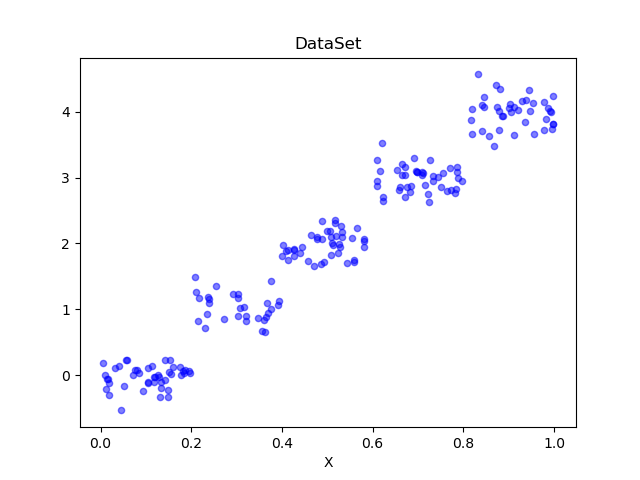
看一下数据：

一列都为1.0，即x0。第二列为x1，即x轴数据。第三列为x2，即y轴数据。

首先绘制下数据，看下数据分布。



代码输出如下：

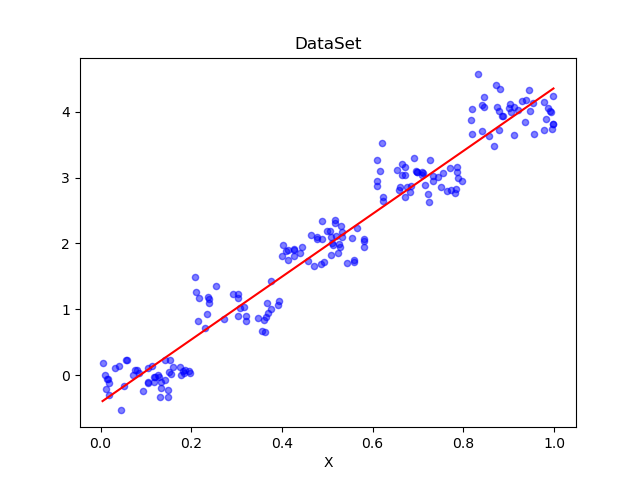


通过可视化数据，看到了数据的分布情况。

**Linear-1.py**

接下来，根据上文中推导的回归系数计算方法，求出回归系数向量，并根据回归系数向量绘制回归曲线

代码输出如下：



解析xCopy = xMat.copy()

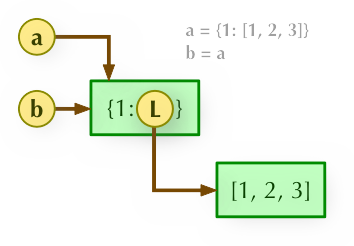
直接赋值：其实就是对象的引用（别名）。

浅拷贝(copy)：拷贝父对象，不会拷贝对象的内部的子对象。

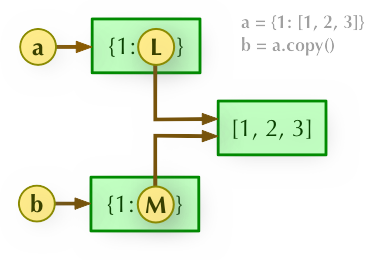
深拷贝(deepcopy)：copy 模块的 deepcopy 方法，完全拷贝了父对象及其子对象。

解析

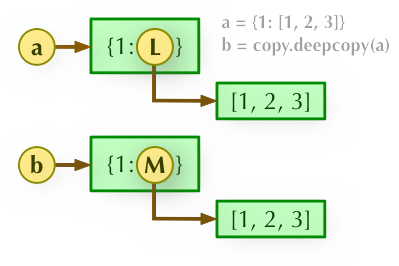
b = a: 赋值引用，a 和 b 都指向同一个对象。



b = a.copy(): 浅拷贝, a 和 b 是一个独立的对象，但他们的子对象还是指向统一对象（是引用）。



b = copy.deepcopy(a): 深度拷贝, a 和 b 完全拷贝了父对象及其子对象，两者是完全独立的。

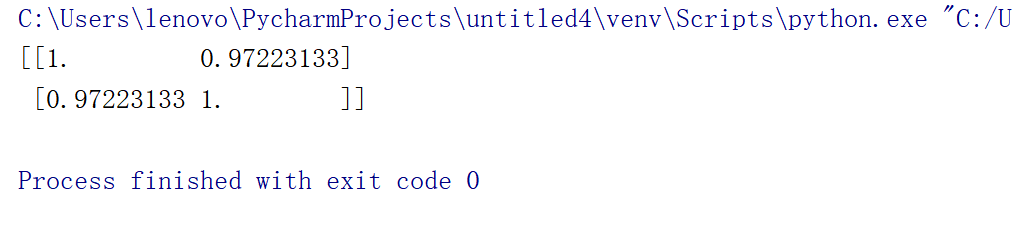


**Linear-2.py**

**如何判断拟合曲线的拟合效果的如何呢？**

可以根据自己的经验进行观察，除此之外，我们还可以使用corrcoef方法，来比较预测值和真实值的相关性。

代码输出如下：



可以看到，对角线上的数据是1.0，因为yMat和自己的匹配是完美的，而YHat和yMat的相关系数为0.98。

最佳拟合直线方法将数据视为直线进行建模。

**Linear-3.py**

## **局部加权线性回归**

线性回归的一个问题是有可能出现欠拟合现象。

显而易见，如果模型欠拟合将不能取得好的预测效果。所以有些方法允许在估计中引入一些偏差，从而降低预测的均方误差。

其中的一个方法是局部加权线性回归。

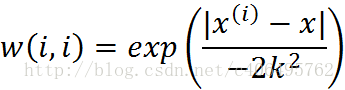
在该方法中，我们给待预测点附近的每个点赋予一定的权重。与kNN一样，这种算法每次预测均需要事先选取出对应的数据子集。

该算法解除回归系数W的形式如下：

IMG_256

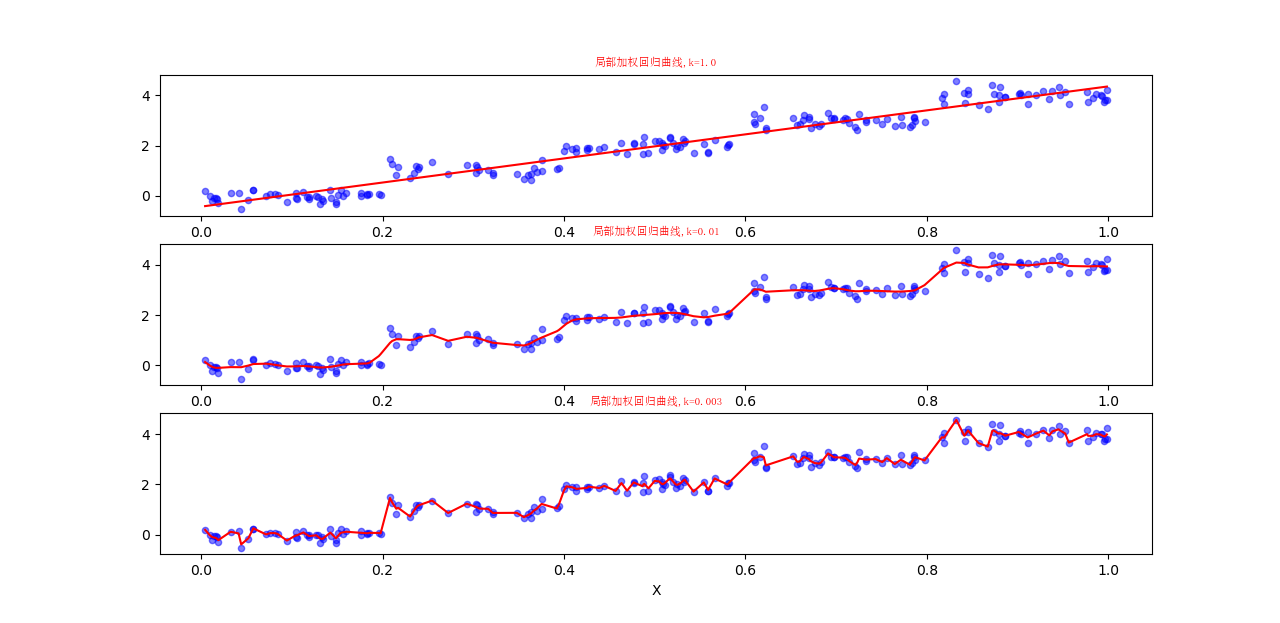
其中W是一个矩阵，这个公式跟我们上面推导的公式的区别就在于W，它用来给每个点赋予权重。

LWLR使用"核"来对附近的点赋予更高的权重。核的类型可以自由选择，最常用的核就是高斯核，高斯核对应的权重如下：



这样就可以根据上述公式，编写局部加权线性回归，我们通过改变k的值，可以调节回归效果。

代码输出如下：



可以看到，当k越小，拟合效果越好。但是当k过小，会出现过拟合的情况，例如k等于0.003的时候。

xMat[:,1].flatten().A[0]

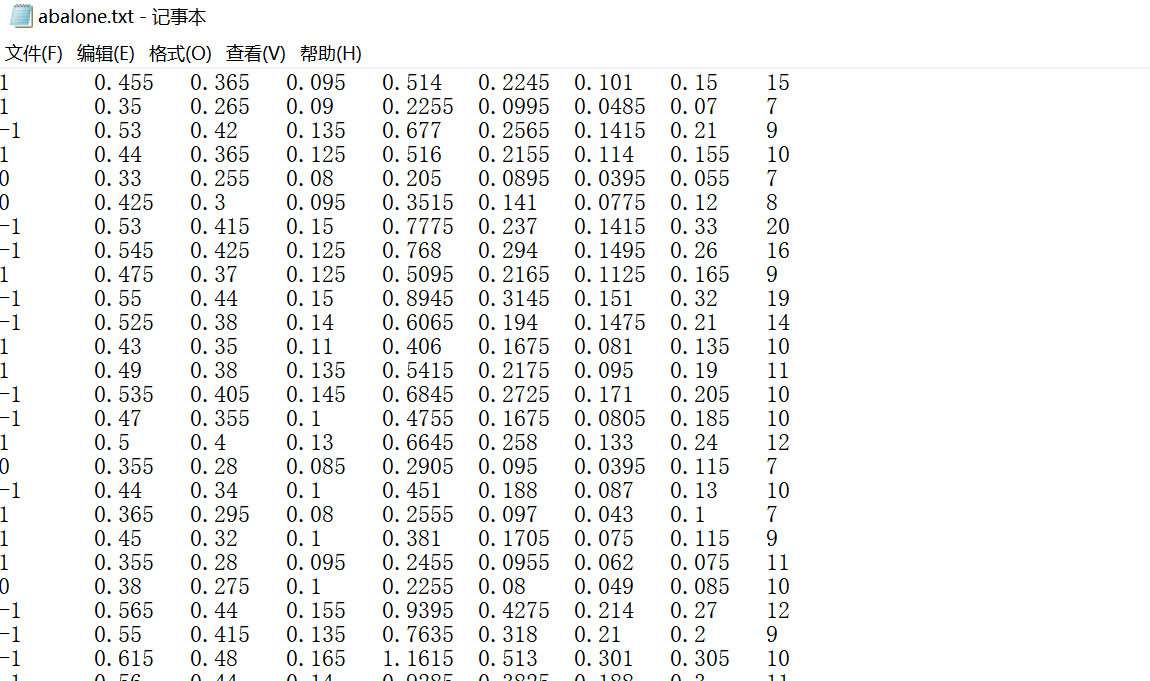
flatten是numpy.ndarray.flatten的一个函数，即返回一个折叠成一维的数组。但是该函数只能适用于numpy对象，即array或者mat。

xMat折叠成一维数组，而且是按A的方式，进行折叠，然后[0]是取第一个元素

**Linear-4.py**

# **预测鲍鱼的年龄**

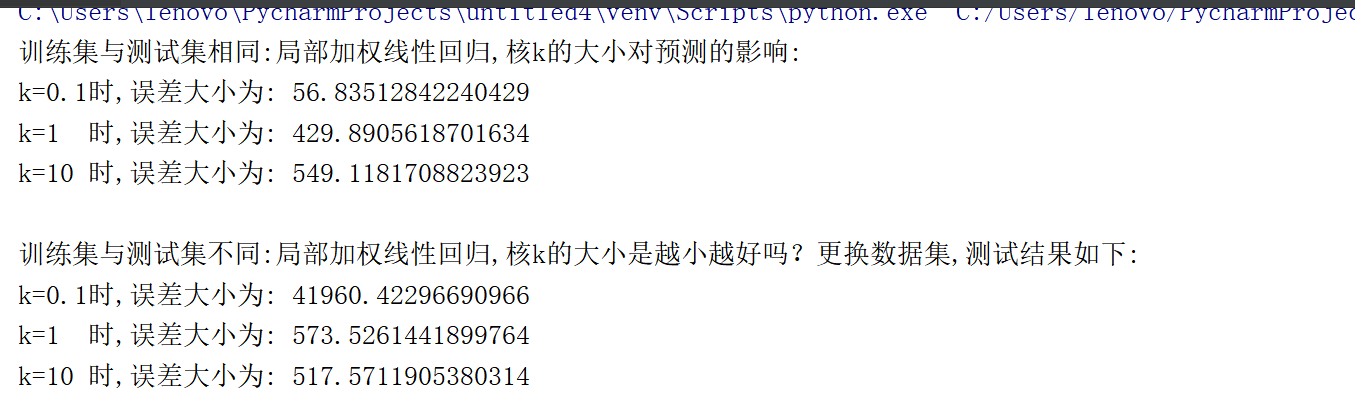
在abalone.txt文件中，数据如下图所示：

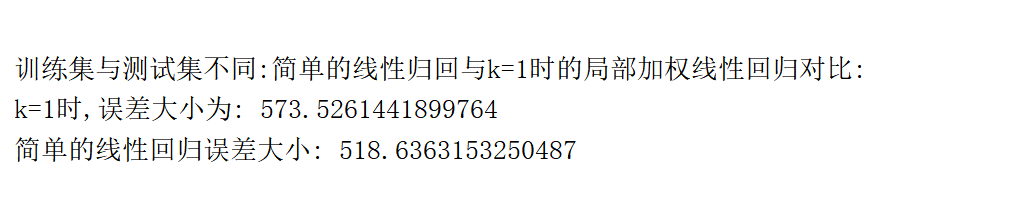


数据集是多维的，所以很难画出它的分布情况。

每个维度数据的代表的含义没有给出，不过最后一列的数据是y值就可以了，最后一列代表的是鲍鱼的真实年龄。

代码输出如下：





可以看到，当k=0.1时，训练集误差小，但是应用于新的数据集之后，误差反而变大了。这就是经常说的过拟合现象。

可以看到，当k=1时，局部加权线性回归和简单的线性回归得到的效果差不多。这也表明一点，必须在未知数据上比较效果才能选取到最佳模型。

本示例展示了如何使用局部加权线性回归来构建模型，可以得到比普通线性回归更好的效果。局部加权线性回归的问题在于，每次必须在整个数据集上运行。也就是说为了做出预测，必须保存所有的训练数据。

**总结**

在局部加权线性回归中，过小的核可能导致过拟合现象，即测试集表现良好，训练集表现就渣渣了。

训练的模型要在测试集比较它们的效果，而不是在训练集上。

**岭回归**

如果数据的特征比样本点还多应该怎么办？

不能再使用上文的方法进行计算了，因为矩阵X不是满秩矩阵，非满秩矩阵在求逆时会出现问题。为了解决这个问题，引入岭回归的概念。

**什么是岭回归？**

在一般的线性回归最小化均方误差的基础上增加了一个参数w的L2范数的罚项，从而最小化罚项残差平方和：

IMG_256

岭回归就是在普通线性回归的基础上引入单位矩阵。回归系数的计算公式变形如下：

IMG_257

首先需要对特征做标准化处理。

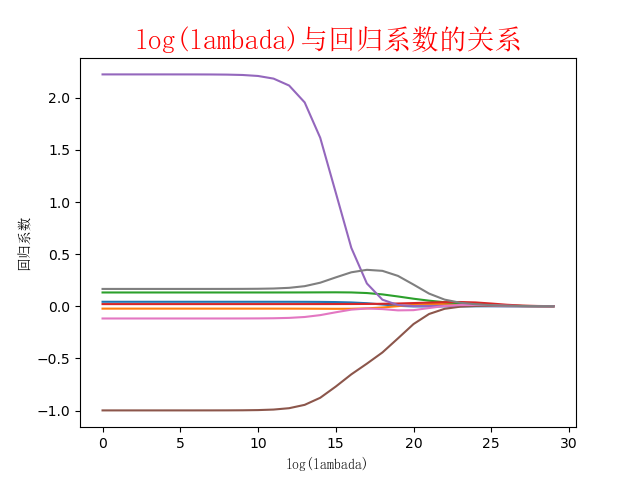
因为，我们需要使每个维度特征具有相同的重要性。

使用的标准化处理比较简单，将所有特征都减去各自的均值并除以方差。

**Linear-5.py**

其中，λ为模型的参数。

我们先绘制一个回归系数与log(λ)的曲线图，看下它们的规律



图绘制了回归系数与log(λ)的关系。

在最左边，即λ最小时，可以得到所有系数的原始值（与线性回归一致）；而在右边，系数全部缩减成0；

在中间部分的某个位置，将会得到最好的预测结果。想要得到最佳的λ参数，可以使用交叉验证的方式获得.

np.eye()

函数的原型：numpy.eye(N,M=None,k=0,dtype=<class 'float'>,order='C)

返回的是一个二维2的数组(N,M)，对角线的地方为1，其余的地方为0.

参数介绍：

**N:**int型，表示的是输出的行数

**M：**int型，可选项，输出的列数，如果没有就默认为N

**k：**int型，可选项，对角线的下标，默认为0表示的是主对角线，负数表示的是低对角，正数表示的是高对角。

**dtype：**数据的类型，可选项，返回的数据的数据类型

**order：**{‘C’，‘F'}，可选项，也就是输出的数组的形式是按照C语言的行优先’C'，还是按照Fortran形式的列优先‘F'存储在内存中

**前向逐步线性回归**

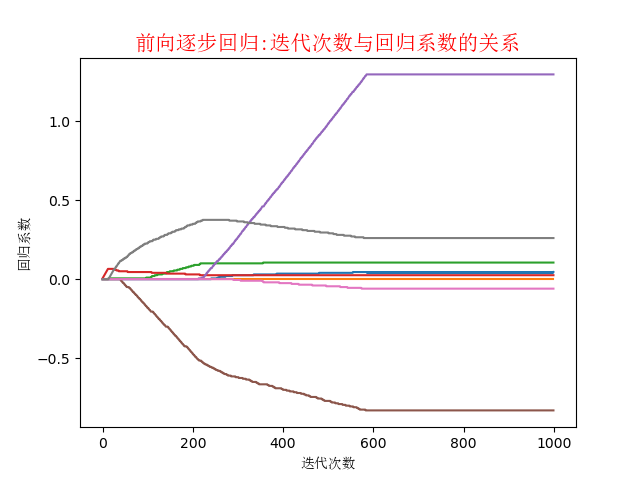
**Linear-6.py**

前向逐步线性回归算法属于一种贪心算法，即每一步都尽可能减少误差。

我们计算回归系数，不再是通过公式计算，而是通过每次微调各个回归系数，然后计算预测误差,使误差最小的一组回归系数，就是我们需要的最佳回归系数。

前向逐步线性回归实现也很简单。当然，还是先进行数据标准化.

代码输出如下：



可以看到，有些系数从始至终都是约为0的，这说明它们不对目标造成任何影响，也就是说这些特征很可能是不需要的。

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。

**总结：**

缩减方法（逐步线性回归或岭回归），就是将一些系数缩减成很小的值或者直接缩减为0。

这样做，就增大了模型的偏差（减少了一些特征的权重），通过把一些特征的回归系数缩减到0，同时也就减少了模型的复杂度。

消除了多余的特征之后，模型更容易理解，同时也降低了预测误差。但是当缩减过于严厉的时候，就会出现过拟合的现象，即用训练集预测结果很好，用测试集预测就糟糕很多。

## **预测乐高玩具套件的价格**

**Linear-7.py**

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。本次实例，就是使用回归方法对收藏者之间的交易价格进行预测。

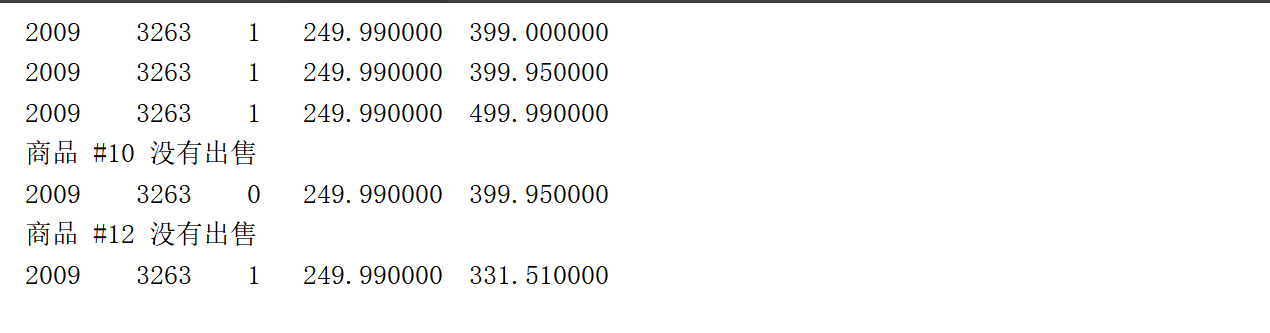
**获取数据**

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。

但是，在网上找到了书上用到的那些html文件。

我们通过解析html文件，来获取我们需要的信息。

代码输出如下：



Error:

C:/Users/lenovo/PycharmProjects/untitled4/Linear Regression/Linear-7.py:25: GuessedAtParserWarning: No parser was explicitly specified, so I'm using the best available HTML parser for this system ("lxml"). This usually isn't a problem, but if you run this code on another system, or in a different virtual environment, it may use a different parser and behave differently.

The code that caused this warning is on line 25 of the file C:/Users/lenovo/PycharmProjects/untitled4/Linear Regression/Linear-7.py. To get rid of this warning, pass the additional argument 'features="lxml"' to the BeautifulSoup constructor.

soup = BeautifulSoup(html)

把soup = BeautifulSoup(html)改成了soup = BeautifulSoup(html，‘lxml’)也报错了，发现因为没有解析库安装包，导入以后就没有报错了。

第二种方法：soup = BeautifulSoup(html, 'html.parser')

这些特征分别为：出品年份、部件数目、是否为全新、原价、售价（二手交易）。

find:返回第一个符合条件的节点

find\_all:以列表形式将符合条件的节点全部返回

find\_all可以传入字符串，正则表达式，列表（用于多个标签）

查找所有这个class的内容，返回为列表

print(soup.find\_all(class\_='easyList'))

re匹配已d开头的内容

print(soup.find\_all(re.compile('^d')))

text搜索

print(soup.find\_all(text='租房'))

title() 以首字母大写的方式显示每个单词。

lower（）用法：所有字母小写

upper（）用法：所有字母大写

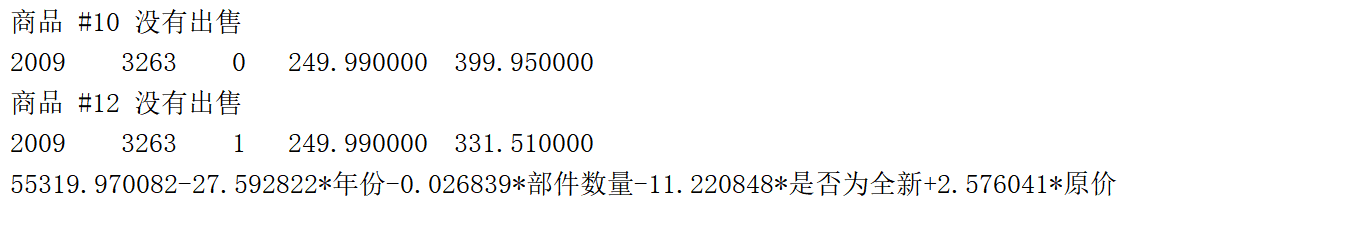
**Linear-8.py**

### **建立模型**

处理好了数据集，接下来是训练模型。

首先我们需要添加全为0的特征X0列。因为线性回归的第一列特征要求都是1.0，然后使用最简单的普通线性回归i。

代码输出如下：



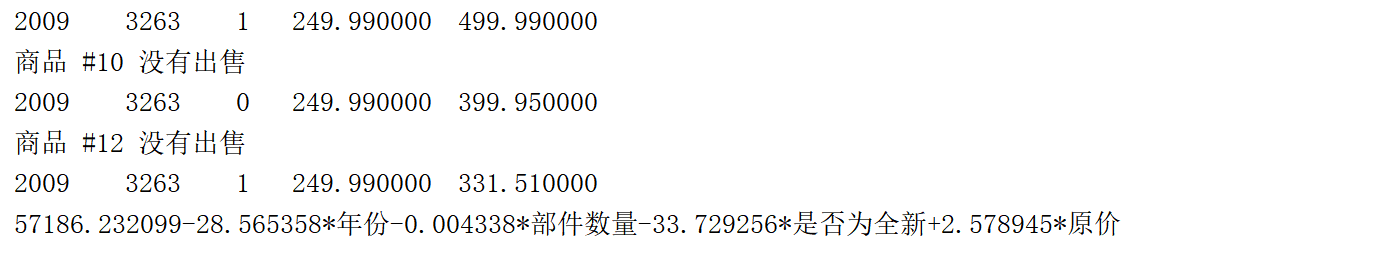
**Linear-9.py**

可以看到，模型采用的公式如上图所示。

虽然这个模型对于数据拟合得很好，但是套件里的部件数量越多，售价反而降低了，这是不合理的。

我们使用岭回归，通过交叉验证，找到使误差最小的λ对应的回归系数。

代码输出如下：



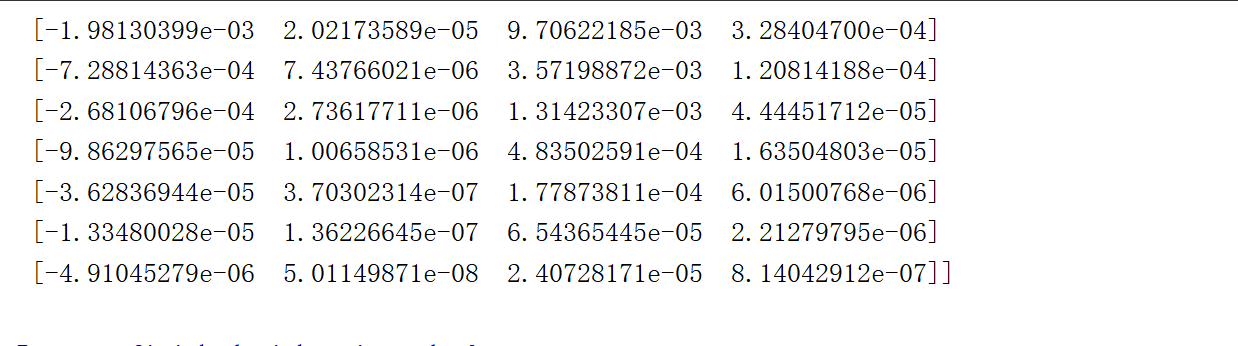
随机选取样本，因为其随机性，所以每次运行的结果可能略有不同。

但是，看到，它与常规的最小二乘法，即普通的线性回归没有太大差异。

**Linear-10.py**

现在，我们看一下在缩减过程中回归系数是如何变化的。

代码输出如下：



看运行结果的第一行，可以看到最大的是第4项，第二大的是第2项。

因此，如果只选择一个特征来做预测的话，我们应该选择第4个特征。如果可以选择2个特征的话，应该选择第4个和第2个特征。

这种分析方法使得我们可以挖掘大量数据的内在规律。在仅有4个特征时，该方法的效果也许并不明显；但如果有100个以上的特征，该方法就会变得十分有效：它可以指出哪个特征是关键的，而哪些特征是不重要的。

## **使用Sklearn的linear\_model**

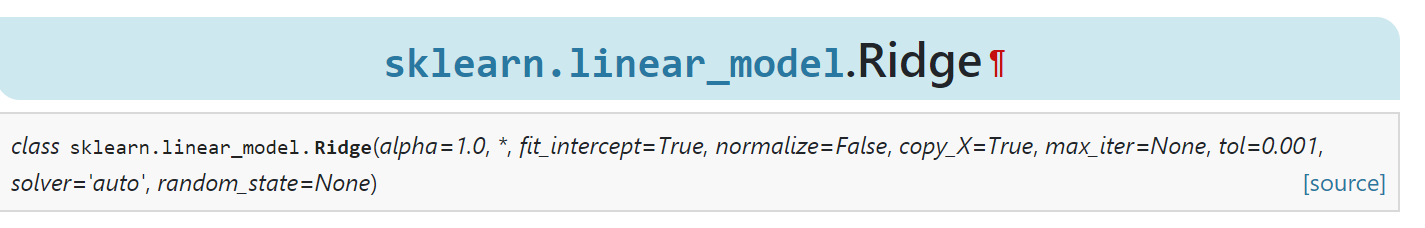
使用sklearn实现下岭回归吧。

sklearn.linear\_model提供了很多线性模型，包括岭回归、贝叶斯回归、Lasso等。本文主要讲解岭回归Ridge。



### **Ridge**

让我们先看下Ridge这个函数，一共有8个参数：



参数说明如下：

**alpha**：正则化系数，float类型，默认为1.0。

正则化改善了问题的条件并减少了估计的方差。较大的值指定较强的正则化。

**fit\_intercept：**是否需要截距，bool类型，默认为True。也就是是否求解b。

**normalize：**是否先进行归一化，bool类型，默认为False。

如果为真，则回归X将在回归之前被归一化。

当fit\_intercept设置为False时，将忽略此参数。

**copy\_X**：是否复制X数组，bool类型，默认为True，

如果为True，将复制X数组; 否则，它覆盖原数组X。

**max\_iter：**最大的迭代次数，int类型，默认为None，最大的迭代次数，对于sparse\_cg和lsqr而言，默认次数取决于scipy.sparse.linalg，对于sag而言，则默认为1000次。

**tol：**精度，float类型，默认为0.001。就是解的精度。

**solver：**求解方法，str类型，默认为auto。

可选参数为：auto、svd、cholesky、lsqr、sparse\_cg、sag。

auto根据数据类型自动选择求解器。

svd使用X的奇异值分解来计算Ridge系数。对于奇异矩阵比cholesky更稳定。

cholesky使用标准的scipy.linalg.solve函数来获得闭合形式的解。

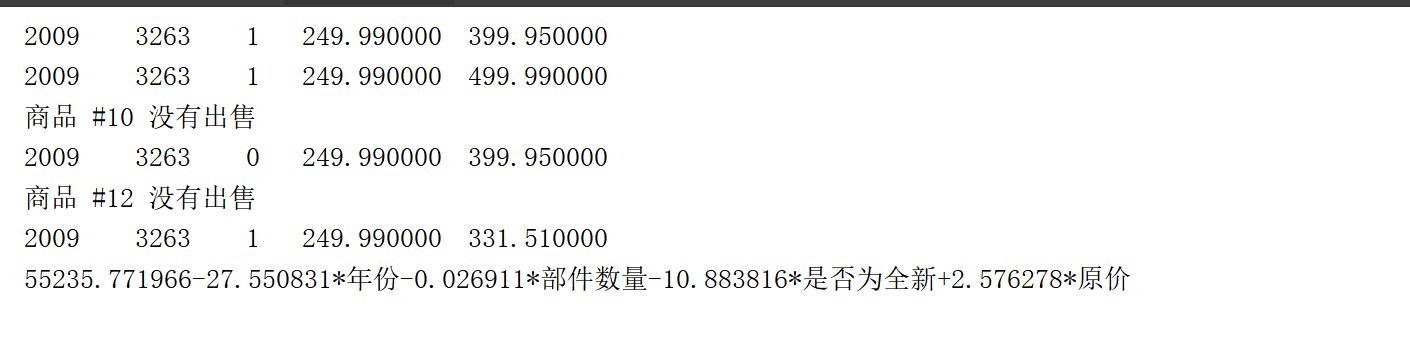
sparse\_cg使用在scipy.sparse.linalg.cg中找到的共轭梯度求解器。

lsqr使用专用的正则化最小二乘常数scipy.sparse.linalg.lsqr。

sag使用随机平均梯度下降。它也使用迭代过程，并且当n\_samples和n\_feature都很大时，通常比其他求解器更快。

**random\_state：**sag的伪随机种子。

代码输出如下：



正则化项系数设为0.5，其余参数使用默认即可。

获得的结果与前面的结果类似。

**总结**

与分类一样，回归也是预测目标值的过程。回归与分类的不同点在于，前者预测连续类型变量，而后者预测离散类型变量。

岭回归是缩减法的一种，相当于对回归系数的大小施加了限制。

另一种很好的缩减法是lasso。lasso难以求解，但可以使用计算简便的逐步线性回归方法求的近似解。

缩减法还可以看做是对一个模型增加偏差的同时减少方法。

# **线性回归和逻辑回归的区别**

回归问题的前提：

1） 收集的数据

2） 假设的模型，即一个函数，这个函数里含有未知的参数，通过学习，可以估计出参数。然后利用这个模型去预测/分类新的数据。

**1. 线性回归**  
假设特征和结果都满足线性。

收集的数据中，每一个分量，就可以看做一个特征数据。每个特征至少对应一个未知的参数。这样就形成了一个线性模型函数。

接下来，就是求解这个函数的方法，有最小二乘法，梯度下降法。

**最小二乘法**

是一个直接的数学求解公式，不过它要求X是列满秩的，

**梯度下降法**

分别有梯度下降法，批梯度下降法，增量梯度下降。

本质上，都是偏导数，步长/最佳学习率，更新，收敛的问题。

**2. 逻辑回归**

逻辑回归的模型是一个非线性模型，sigmoid函数，又称逻辑回归函数。

但是它本质上又是一个线性回归模型，因为除去sigmoid映射函数关系，其他的步骤，算法都是线性回归的。

线性模型，无法做到sigmoid的非线性形式，sigmoid可以轻松处理0/1分类问题。

1. **一般线性回归**  
   线性回归是以高斯分布为误差分析模型；

逻辑回归 采用的是伯努利分布分析误差。

而高斯分布、伯努利分布、贝塔分布、迪特里特分布，都属于指数分布。

而一般线性回归，在x条件下，y的概率分布 p(y|x) 就是指 指数分布.

经历最大似然估计的推导，就能导出一般线性回归的 误差分析模型（最小化误差模型）。

softmax回归就是 一般线性回归的一个例子。

1. **拟合：拟合模型/函数**  
   由测量的数据，估计一个假定的模型/函数。

过拟合的问题如何解决？

模型太复杂，参数过多，特征数目过多。

方法：

1） 减少特征的数量，有人工选择，或者采用模型选择算法

2） 正则化，即保留所有特征，但降低参数的值的影响。

正则化的优点是，特征很多时，每个特征都会有一个合适的影响因子。