

《机器学习实战》学习笔记（八）：预测数值型数据 - 回归

原创 我是管小亮 2019-09-06 14:35:08 1841 收藏 9

版权

分类专栏: Machine Learning 文章标签: 机器学习实战 机器学习 学习笔记 回归 预测数值型数据: 回归

欢迎关注WX公众号：【程序员管小亮】

【机器学习】《机器学习实战》读书笔记及代码 总目录

- <https://blog.csdn.net/TeFuirnever/article/details/99701256>

GitHub代码地址:

- <https://github.com/TeFuirnever/Machine-Learning-in-Action>

目录

欢迎关注WX公众号：【程序员管小亮】

本章内容

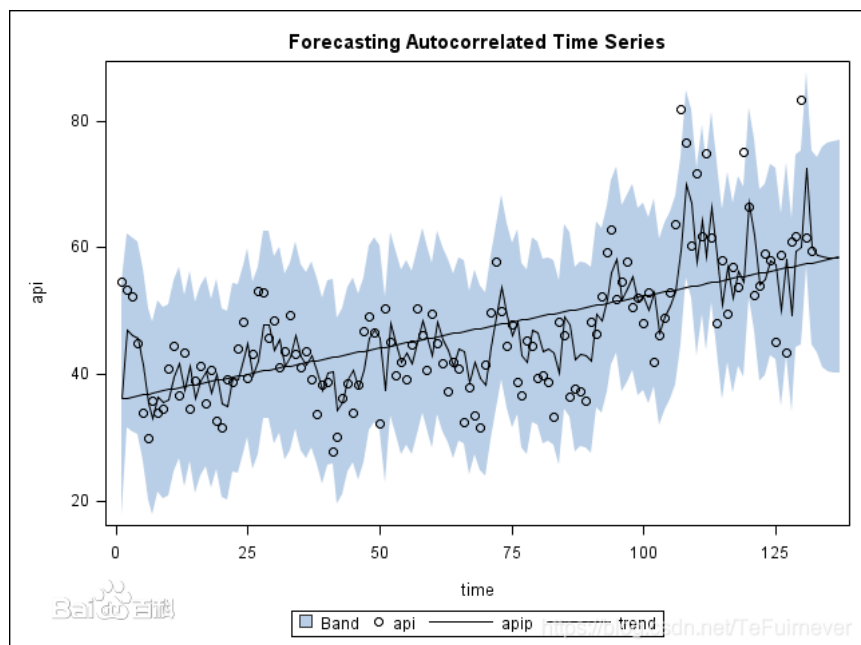
- 1、回归
 - 2、用线性回归找到最佳拟合直线
 - 3、局部加权线性回归
 - 4、示例：预测鲍鱼的年龄
 - 5、缩减系数来“理解”数据
 - 1) 岭回归
 - 2) lasso
 - 3) 前向逐步回归
 - 6、示例：预测乐高玩具套装的价格
 - 1) 收集数据
 - 2) 训练算法：建立模型
 - 7、Sklearn构建岭回归
 - 8、sklearn.linear_model.Ridge
 - 9、总结
- 参考文章

本章内容

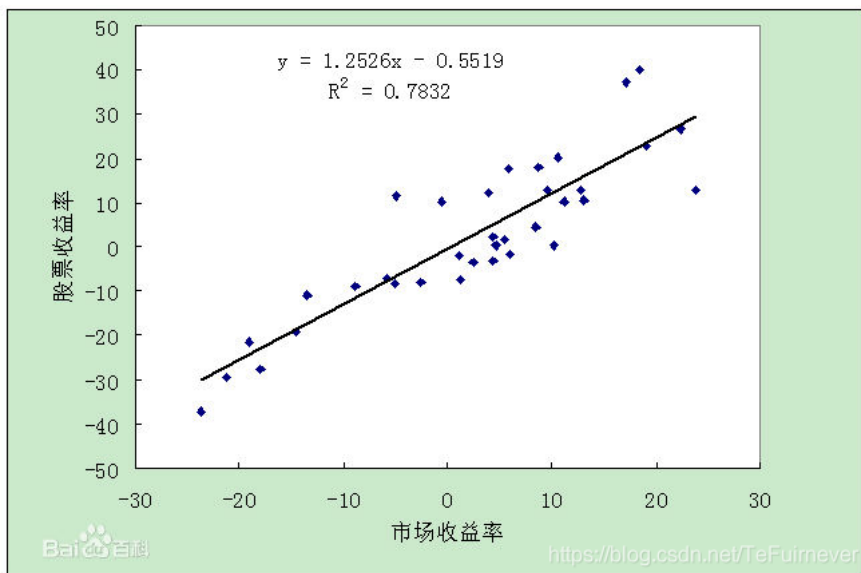
- 线性回归
- 局部加权线性回归
- 岭回归和逐步线性回归
- 预测鲍鱼年龄和玩具售价

1、回归

前面的章节介绍了分类（前七章其实都是分类，不知道你看出来了没有），**分类**的目标变量是**标称型数据**，而这一次我们将会对**连续型数据**做出预测，也就是回归。



很多人的第一想法很可能是：“回归能用来做些什么呢？”。我的观点是，**回归可以做任何事情**。然而大多数公司常常使用回归法做一些比较沉闷的事情，例如销售量预测或者制造缺陷预测。比如：



我最近看到一个比较有新意的应用，就是预测名人的离婚率（我不会被律师函警告吧...）。

2、用线性回归找到最佳拟合直线

线性回归

优点：结果易于理解，计算上不复杂。

缺点：对非线性的数据拟合不好。

适用数据类型：数值型和标称型数据。

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。比如假设你想要预测姐姐男友汽车的功率大小，可能会这么计算：

$$\text{HorsePower} = 0.0015 * \text{annualSalary} - 0.99 * \text{hoursListeningToPublicRadio}$$

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，方程就确定了，如果再给定输入变量，则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。

说到 **回归**，一般都是指 **线性回归 (linear regression)**，所以这里的回归和线性回归代表同一个意思。线性回归意味着可以将输入项分别乘以一些常量，再将结果加起来得到输出。

需要说明的是，存在另一种称为 **非线性回归** 的回归模型，该模型不认同上面的做法，比如认为输出可能是输入的乘积。这样，上面的功率计算公式也可以写做：

$$\text{HorsePower} = 0.0015 * \text{annualSalary} / \text{hoursListeningToPublicRadio}$$

这就是一个非线性回归的例子，不过我们这里不做深入讨论。

回归的一般方法

- (1) 收集数据：采用任意方法收集数据。
- (2) 准备数据：回归需要数值型数据，标称型数据将被转成二值型数据。
- (3) 分析数据：绘出数据的可视化二维图将有助于对数据做出理解和分析，在采用缩减法求得新回归系数之后，可以将新拟合线绘在图上作为对比。
- (4) 训练算法：找到回归系数。
- (5) 测试算法：使用R2或者预测值和数据的拟合度，来分析模型的效果。
- (6) 使用算法：使用回归，可以在给定输入的时候预测出一个数值，这是对分类方法的提升，因为这样可以预测连续型数据而不仅仅是离散类别标签。

那么应当怎样从一大堆数据里求出回归方程呢？假定输入数据存放在矩阵X中，而回归系数存放在向量 w 中。那么对于给定的数据 X_i ，预测结果将会通过 $Y_i = X_i^T w$ 给出。现在的问题是，手里有一些 X 和对应的 Y ，怎样才能找到 w 呢？

一个常用的方法就是找出使误差最小的 w 。这里的误差是指预测 Y 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，所以采用平方误差。

平方误差可以写做：

$$\sum_{i=1}^m (y_i - x_i^T w)^2$$

用矩阵表示还可以写做：

$$(Y - Xw)^T (Y - Xw)$$

如果对 w 求导，得到

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

<http://blog.csdn.net/s1406485762>

令其等于零，解出 w 如下：

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

值得注意的是，上述公式中包含逆矩阵，也就是需要对矩阵求逆，因此这个方程只在逆矩阵存在的时候适用。然而，矩阵的逆可能并不存在，因此必须要在代码中对此作出判断。

\hat{w} 上方的小标记表示，这是当前可以估计出的 w 的最优解。从现有数据上估计出的 w 可能并不是数据中的真实 w 值，所以这里使用了一个“帽”符号来表示它仅是 w 的一个最佳估计。这是统计学中的常见问题，除了矩阵方法外还有很多其他方法可以解决。通过调用NumPy库里的矩阵方法，仅使用几行代码就完成所需功能。该方法也称作OLS，意思是“**普通最小二乘法**” (ordinary least squares) 。

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
1.000000	0.067732	3.176513		
1.000000	0.427810	3.816464		
1.000000	0.995731	4.550095		
1.000000	0.738336	4.256571		
1.000000	0.981083	4.560815		
1.000000	0.526171	3.929515		
1.000000	0.378887	3.526170		
1.000000	0.033859	3.156393		
1.000000	0.132791	3.110301		
1.000000	0.138306	3.149813		
1.000000	0.247809	3.476346		
1.000000	0.648270	4.119688		
1.000000	0.731209	4.282233		
1.000000	0.236833	3.486582		
1.000000	0.969788	4.655492		
1.000000	0.607492	3.965162		
1.000000	0.358622	3.514900		
1.000000	0.147846	3.125947		
1.000000	0.637820	4.094115		
1.000000	0.230372	3.476039		
1.000000	0.070237	3.210610		
1.000000	0.067154	3.190612		
1.000000	0.925577	4.631504		
1.000000	0.717733	4.295890		
1.000000	0.015371	3.085028		
1.000000	0.335070	3.448080		
1.000000	0.040486	3.167440		
1.000000	0.212575	3.364266		
1.000000	0.617218	3.993482		
1.000000	0.541196	3.891471		
1.000000	0.045353	3.143259		
1.000000	0.126762	3.114204		
1.000000	0.556486	3.851484		

先绘制下数据，看下数据分布。编写代码如下：

```

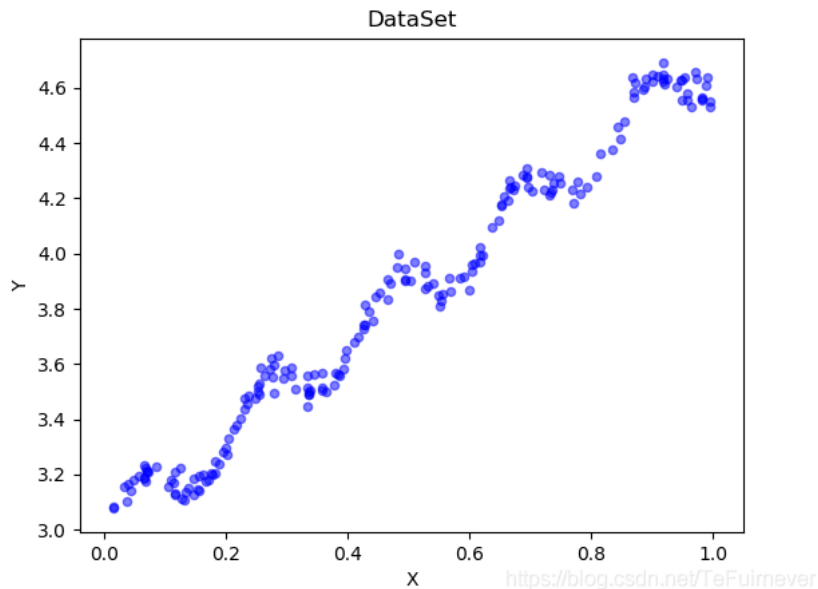
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # 加载数据
5 def loadDataSet(fileName):
6     """
7     Parameters:
8         fileName - 文件名
9     Returns:
10        xArr - x数据集
11        yArr - y数据集
12    """
13    numFeat = len(open(fileName).readline().split('\t')) - 1
14    xArr = []; yArr = []
15    fr = open(fileName)
16    for line in fr.readlines():
17        lineArr = []
18        curLine = line.strip().split('\t')
19        for i in range(numFeat):
20            lineArr.append(float(curLine[i]))
21        xArr.append(lineArr)
22        yArr.append(float(curLine[-1]))
23    return xArr, yArr
24
25 # 绘制数据集
26 def plotDataSet():
27     xArr, yArr = loadDataSet('ex0.txt')
28     n = len(xArr)
29     xcord = []; ycord = []
30     for i in range(n):

```

```

30     xcord.append(xArr[i][1]); ycord.append(yArr[i])    #样本点
31     fig = plt.figure()
32     ax = fig.add_subplot(111)                        #添加subplot
33     ax.scatter(xcord, ycord, s = 20, c = 'blue',alpha = .5) #绘制样本点
34     plt.title('DataSet')                             #绘制title
35     plt.xlabel('X')
36     plt.ylabel('Y')
37     plt.show()
38
39
40 if __name__ == '__main__':
41     plotDataSet()
42

```



通过可视化数据，加上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，编写代码如下：

```

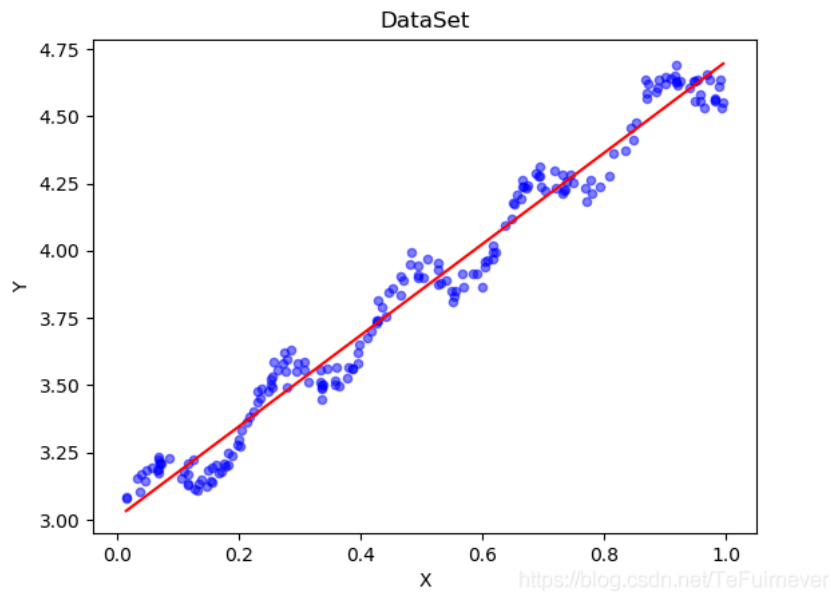
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  # 加载数据
5  def loadDataSet(fileName):
6      """
7      Parameters:
8          fileName - 文件名
9      Returns:
10         xArr - x数据集
11         yArr - y数据集
12      """
13      numFeat = len(open(fileName).readline().split('\t')) - 1
14      xArr = []; yArr = []
15      fr = open(fileName)
16      for line in fr.readlines():
17          lineArr = []
18          curLine = line.strip().split('\t')
19          for i in range(numFeat):
20              lineArr.append(float(curLine[i]))
21          xArr.append(lineArr)
22          yArr.append(float(curLine[-1]))
23      return xArr, yArr
24
25  # 计算回归系数w
26  def standRegres(xArr,yArr):
27      """
28      Parameters:
29         xArr - x数据集
30         yArr - y数据集
31      Returns:
32         ws - 回归系数
33      """
34

```

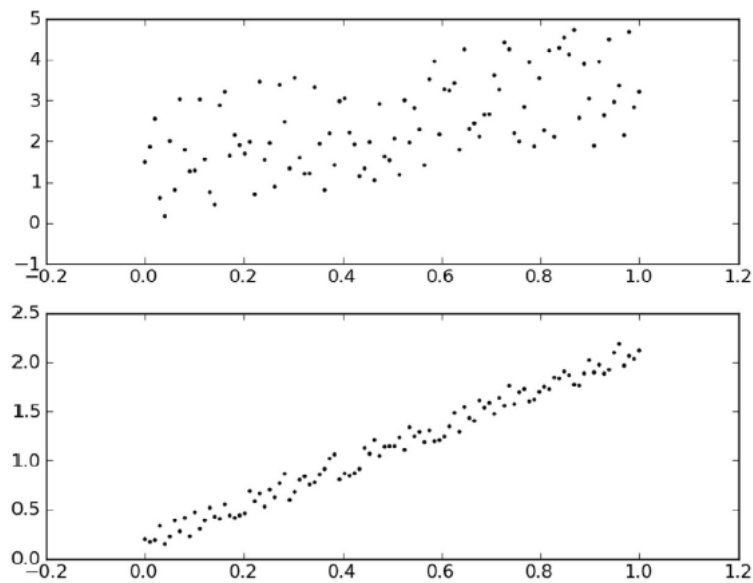
```

33 xMat = np.mat(xArr); yMat = np.mat(yArr).I
34 xTx = xMat.T * xMat #根据文中推导的公式计算回归系数
35 if np.linalg.det(xTx) == 0.0:
36     print("矩阵为奇异矩阵,不能求逆")
37     return
38 ws = xTx.I * (xMat.T*yMat)
39 return ws
40
41 # 绘制回归曲线和数据点
42 def plotRegression():
43     xArr, yArr = loadDataSet('ex0.txt') #加载数据集
44     ws = standRegres(xArr, yArr) #计算回归系数
45     xMat = np.mat(xArr) #创建xMat矩阵
46     yMat = np.mat(yArr) #创建yMat矩阵
47     xCopy = xMat.copy() #深拷贝xMat矩阵
48     xCopy.sort(0) #排序
49     yHat = xCopy * ws #计算对应的y值
50     fig = plt.figure()
51     ax = fig.add_subplot(111) #添加subplot
52     ax.plot(xCopy[:, 1], yHat, c = 'red') #绘制回归曲线
53     ax.scatter(xMat[:,1].flatten().A[0], yMat.flatten().A[0], s = 20, c = 'blue',alpha = .5) #绘制样本点
54     plt.title('DataSet') #绘制title
55     plt.xlabel('X')
56     plt.ylabel('Y')
57     plt.show()
58
59 if __name__ == '__main__':
60     plotRegression()
61

```



几乎任一数据集都可以用上述方法建立模型，那么，如何判断这些模型的好坏呢？



具有相同回归系数（0和2.0）的两组数据。上图的相关系数是0.58，而下图的相关系数是0.99
<https://blog.csdn.net/TeFuirnever>

比较一下上图的两个子图，如果在两个数据集上分别作线性回归，将得到完全一样的模型（拟合直线）。显然两个数据是不一样的，那么模型分别在二者上的效果如何？我们当如何比较这些效果的好坏呢？

有种方法可以计算预测值 \hat{y} 序列和真实值 y 序列的匹配程度，那就是计算这两个序列的相关系数。在Python中，NumPy库提供了相关系数的计算方法：可以通过命令 `corrcoef(yEstimate, yActual)` 来计算预测值和真实值的相关性。

```

1  import numpy as np
2
3  # 加载数据
4  def loadDataSet(fileName):
5      """
6      Parameters:
7          fileName - 文件名
8      Returns:
9          xArr - x数据集
10         yArr - y数据集
11      """
12      numFeat = len(open(fileName).readline().split('\t')) - 1
13      xArr = []; yArr = []
14      fr = open(fileName)
15      for line in fr.readlines():
16          lineArr = []
17          curLine = line.strip().split('\t')
18          for i in range(numFeat):
19              lineArr.append(float(curLine[i]))
20          xArr.append(lineArr)
21          yArr.append(float(curLine[-1]))
22      return xArr, yArr
23
24  # 计算回归系数w
25  def standRegres(xArr, yArr):
26      """
27      Parameters:
28          xArr - x数据集
29          yArr - y数据集
30      Returns:
31          ws - 回归系数
32      """
33      xMat = np.mat(xArr); yMat = np.mat(yArr).T
34      xTx = xMat.T * xMat #根据文中推导的公示计算回归系数
35      if np.linalg.det(xTx) == 0.0:
36          print("矩阵为奇异矩阵,不能求逆")
37          return
38      ws = xTx.I * (xMat.T * yMat)
39      return ws
40
41  if __name__ == '__main__':
42      xArr, yArr = loadDataSet('ex0.txt') #加载数据集
43      ws = standRegres(xArr, yArr) #计算回归系数

```

```

42     xMat = np.mat(xArr)                #创建xMat矩阵
43     yMat = np.mat(yArr)                #创建yMat矩阵
44     yHat = xMat * ws
45     print(np.corrcoef(yHat.T, yMat))
46

```

```

[[1.      0.98647356]
 [0.98647356 1.      ]]

```

该矩阵包含所有两两组合的相关系数。可以看到，对角线上的数据是1.0，因为yMat和自己的匹配是最完美的，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，具有十分不错的表现。但是拟合图像的数据当中似乎还存在其他的潜在模式。那么如何才能利用这些模式呢？我们可以根据数据来局部调整预测，下面就会介绍这种方法。

3、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，因为它求的是 **具有最小均方误差的无偏估计**。显而易见，如果模型欠拟合将不能取得最好的预测效果。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 (Locally Weighted Linear Regression, LWLR)**。在该算法中，我们给待预测点附近的每个点赋予一定的权重；然后在这个子集上基于最小均方差来进行普通的回归。与kNN一样，这种算法每次预测均需要事先选取对对应的数据子集。该算法解出回归系数w的形式如下：

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

其中w是一个矩阵，用来给每个数据点赋予权重。

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。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：

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

这样就构建了一个只含对角元素的权重矩阵w，并且点x与x(i)越近，w(i,i)将会越大。上述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LWLR时唯一需要考虑的参数，代码如下：

```

1  from matplotlib.font_manager import FontProperties
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # 加载数据
6  def loadDataSet(fileName):
7      """
8      Parameters:
9          fileName - 文件名
10     Returns:
11         xArr - x数据集
12         yArr - y数据集
13     """
14     numFeat = len(open(fileName).readline().split('\t')) - 1
15     xArr = []; yArr = []
16     fr = open(fileName)
17     for line in fr.readlines():
18         lineArr = []
19         curLine = line.strip().split('\t')
20         for i in range(numFeat):
21             lineArr.append(float(curLine[i]))
22         xArr.append(lineArr)
23         yArr.append(float(curLine[-1]))
24     return xArr, yArr
25
26 # 绘制多条局部加权回归曲线
27 def plotLwlrRegression():
28     font = FontProperties(fname=r"c:\windows\fonts\simsttc", size=14)
29     xArr, yArr = loadDataSet('ex0.txt')
30     yHat_1 = lwlrTest(xArr, xArr, yArr, 1.0)
31     yHat_2 = lwlrTest(xArr, xArr, yArr, 0.01)
32     yHat_3 = lwlrTest(xArr, xArr, yArr, 0.003)

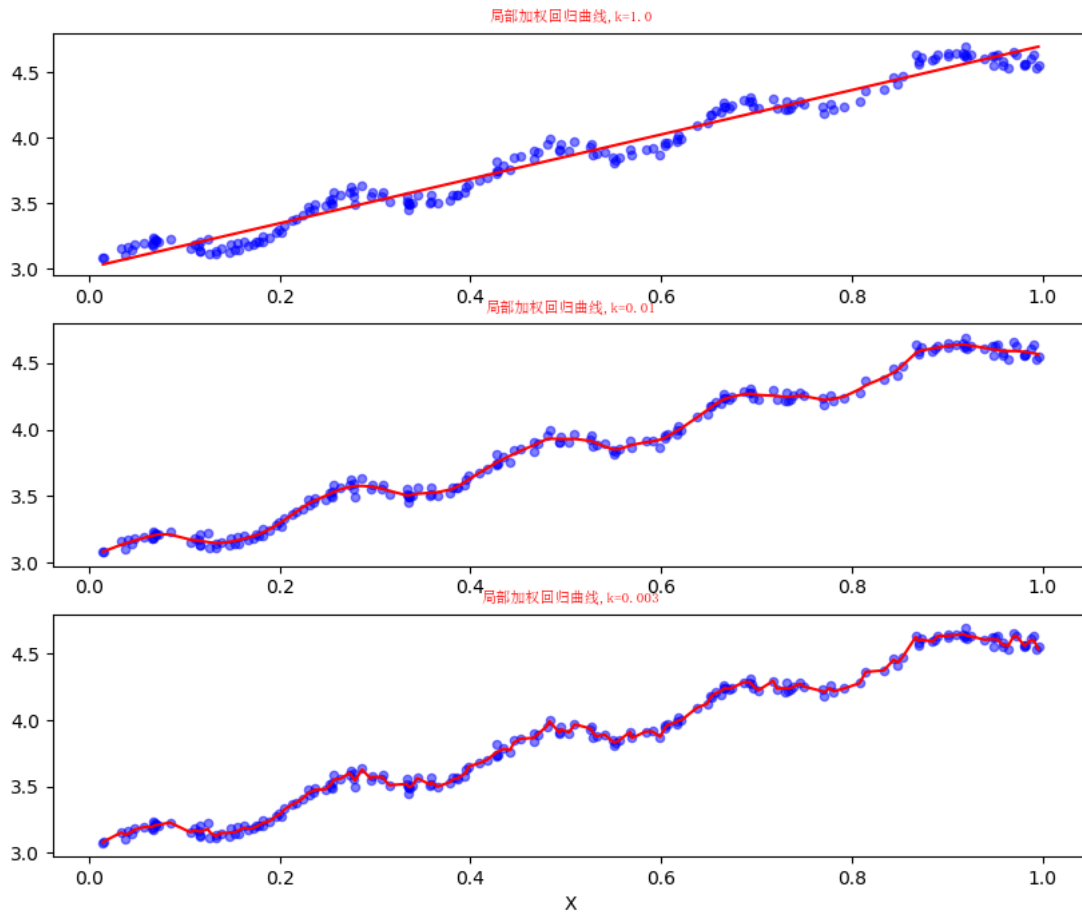
```

#加载数据集
 #根据局部加权线性回归计算yHat
 #根据局部加权线性回归计算yHat
 #根据局部加权线性回归计算yHat


```

32     xMat = np.mat(xArr)                                #创建xMat矩阵
33     yMat = np.mat(yArr)                                #创建yMat矩阵
34     srtInd = xMat[:, 1].argsort(0)                      #排序, 返回索引值
35     xSort = xMat[srtInd][:,0,:]
36     fig, axs = plt.subplots(nrows=3, ncols=1,sharex=False, sharey=False, figsize=(10,8))
37     axs[0].plot(xSort[:, 1], yHat_1[srtInd], c = 'red')  #绘制回归曲线
38     axs[1].plot(xSort[:, 1], yHat_2[srtInd], c = 'red')  #绘制回归曲线
39     axs[2].plot(xSort[:, 1], yHat_3[srtInd], c = 'red')  #绘制回归曲线
40     axs[0].scatter(xMat[:,1].flatten().A[0], yMat.flatten().A[0], s = 20, c = 'blue', alpha = .5) #绘制样本点
41     axs[1].scatter(xMat[:,1].flatten().A[0], yMat.flatten().A[0], s = 20, c = 'blue', alpha = .5) #绘制样本点
42     axs[2].scatter(xMat[:,1].flatten().A[0], yMat.flatten().A[0], s = 20, c = 'blue', alpha = .5) #绘制样本点
43     #设置标题,x轴label,y轴label
44     axs0_title_text = axs[0].set_title(u'局部加权回归曲线,k=1.0',FontProperties=font)
45     axs1_title_text = axs[1].set_title(u'局部加权回归曲线,k=0.01',FontProperties=font)
46     axs2_title_text = axs[2].set_title(u'局部加权回归曲线,k=0.003',FontProperties=font)
47     plt.setp(axs0_title_text, size=8, weight='bold', color='red')
48     plt.setp(axs1_title_text, size=8, weight='bold', color='red')
49     plt.setp(axs2_title_text, size=8, weight='bold', color='red')
50     plt.xlabel('X')
51     plt.show()
52
53 # 使用局部加权线性回归计算回归系数w
54 def lwlr(testPoint, xArr, yArr, k = 1.0):
55     """
56     Parameters:
57         testPoint - 测试样本点
58         xArr - x数据集
59         yArr - y数据集
60         k - 高斯核的k,自定义参数
61     Returns:
62         ws - 回归系数
63     """
64     xMat = np.mat(xArr); yMat = np.mat(yArr).T
65     m = np.shape(xMat)[0]
66     weights = np.mat(np.eye((m)))                        #创建权重对角矩阵
67     for j in range(m):                                    #遍历数据集计算每个样本的权重
68         diffMat = testPoint - xMat[j, :]
69         weights[j, j] = np.exp(diffMat * diffMat.T/(-2.0 * k**2))
70     xTx = xMat.T * (weights * xMat)
71     if np.linalg.det(xTx) == 0.0:
72         print("矩阵为奇异矩阵,不能求逆")
73         return
74     ws = xTx.I * (xMat.T * (weights * yMat))              #计算回归系数
75     return testPoint * ws
76
77 # 局部加权线性回归测试
78 def lwlrTest(testArr, xArr, yArr, k=1.0):
79     """
80     Parameters:
81         testArr - 测试数据集
82         xArr - x数据集
83         yArr - y数据集
84         k - 高斯核的k,自定义参数
85     Returns:
86         ws - 回归系数
87     """
88     m = np.shape(testArr)[0]                              #计算测试数据集大小
89     yHat = np.zeros(m)
90     for i in range(m):                                     #对每个样本点进行预测
91         yHat[i] = lwlr(testArr[i],xArr,yArr,k)
92     return yHat
93
94 if __name__ == '__main__':
95     plotlwlrRegression()

```



<https://blog.csdn.net/TeFuirnever>

使用3种不同平滑值绘出的局部加权线性回归结果。上图中的平滑参数 $k=1.0$ ，中图 $k=0.01$ ，下图 $k=0.003$ 。可以看到， $k=1.0$ 时的模型效果与最小二乘法差不多， $k=0.01$ 时该模型可以挖出数据的潜在规律，而 $k=0.003$ 时则考虑了太多的噪声，进而导致了过拟合现象。

局部加权线性回归也存在一个问题，即增加了计算量，因为它对每个点做预测时都必须使用整个数据集。如果避免这些计算将可以减少程序运行时间，从而缓解因计算量增加带来的问题。

4、示例：预测鲍鱼的年龄

在abalone.txt文件中记录了鲍鱼（一种介壳类水生动物）的年龄，鲍鱼年龄可以从鲍鱼壳的层数推算得到。可以看一下数据。

abalone.txt - 记事本									
文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)					
1	0.455	0.365	0.095	0.514	0.2245	0.101	0.15	15	
1	0.35	0.265	0.09	0.2255	0.0995	0.0485	0.07	7	
-1	0.53	0.42	0.135	0.677	0.2565	0.1415	0.21	9	
1	0.44	0.365	0.125	0.516	0.2155	0.114	0.155	10	
0	0.33	0.255	0.08	0.205	0.0895	0.0395	0.055	7	
0	0.425	0.3	0.095	0.3515	0.141	0.0775	0.12	8	
-1	0.53	0.415	0.15	0.7775	0.237	0.1415	0.33	20	
-1	0.545	0.425	0.125	0.768	0.294	0.1495	0.26	16	
1	0.475	0.37	0.125	0.5095	0.2165	0.1125	0.165	9	
-1	0.55	0.44	0.15	0.8945	0.3145	0.151	0.32	19	
-1	0.525	0.38	0.14	0.6065	0.194	0.1475	0.21	14	
1	0.43	0.35	0.11	0.406	0.1675	0.081	0.135	10	
1	0.49	0.38	0.135	0.5415	0.2175	0.095	0.19	11	
-1	0.535	0.405	0.145	0.6845	0.2725	0.171	0.205	10	
-1	0.47	0.355	0.1	0.4755	0.1675	0.0805	0.185	10	
1	0.5	0.4	0.13	0.6645	0.258	0.133	0.24	12	
0	0.355	0.28	0.085	0.2905	0.095	0.0395	0.115	7	
-1	0.44	0.34	0.1	0.451	0.188	0.087	0.13	10	
1	0.365	0.295	0.08	0.2555	0.097	0.043	0.1	7	
1	0.45	0.32	0.1	0.381	0.1705	0.075	0.115	9	
1	0.355	0.28	0.095	0.2455	0.0955	0.062	0.075	11	
0	0.38	0.275	0.1	0.2255	0.08	0.049	0.085	10	
-1	0.565	0.44	0.155	0.9395	0.4275	0.214	0.27	12	
-1	0.55	0.415	0.135	0.7635	0.318	0.21	0.2	9	
-1	0.615	0.48	0.165	1.1615	0.513	0.301	0.305	10	
-1	0.56	0.44	0.14	0.9285	0.3825	0.188	0.3	11	
-1	0.58	0.45	0.185	0.9955	0.3945	0.272	0.285	11	
1	0.59	0.445	0.14	0.931	0.356	0.234	0.28	12	
1	0.605	0.475	0.18	0.9365	0.394	0.219	0.295	15	
1	0.575	0.425	0.14	0.8635	0.393	0.227	0.2	11	
1	0.58	0.47	0.165	0.9975	0.3935	0.242	0.33	10	
-1	0.68	0.56	0.165	1.639	0.6055	0.2805	0.46	15	
1	0.665	0.525	0.165	1.338	0.5515	0.3575	0.35	18	

可以看到，数据集是多维的，所以很难画出它的分布情况，而且每个维度数据的代表的含义没有给出，不过最后一列是y值。

```
1 from matplotlib.font_manager import FontProperties
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # 加载数据
6 def loadDataSet(fileName):
7     """
8     Parameters:
9         fileName - 文件名
10     Returns:
11         xArr - x数据集
12         yArr - y数据集
13     """
14     numFeat = len(open(fileName).readline().split('\t')) - 1
15     xArr = []; yArr = []
16     fr = open(fileName)
17     for line in fr.readlines():
18         lineArr = []
19         curLine = line.strip().split('\t')
20         for i in range(numFeat):
21             lineArr.append(float(curLine[i]))
22         xArr.append(lineArr)
23         yArr.append(float(curLine[-1]))
24     return xArr, yArr
25
26 # 使用局部加权线性回归计算回归系数w
27 def lwlr(testPoint, xArr, yArr, k = 1.0):
28     """
29     Parameters:
30         testPoint - 测试样本点
31         xArr - x数据集
32         yArr - y数据集
33         k - 高斯核的k, 自定义参数
34     Returns:
35         """
```

```

32     returns:
33         ws - 回归系数
34
35     xMat = np.mat(xArr); yMat = np.mat(yArr).T
36     m = np.shape(xMat)[0]
37     weights = np.mat(np.eye((m))) # 创建权重对角矩阵
38     for j in range(m): # 遍历数据集计算每个样本的权重
39         diffMat = testPoint - xMat[j, :]
40         weights[j, j] = np.exp(diffMat * diffMat.T / (-2.0 * k**2))
41     xTx = xMat.T * (weights * xMat)
42     if np.linalg.det(xTx) == 0.0:
43         print("矩阵为奇异矩阵,不能求逆")
44         return
45     ws = xTx.I * (xMat.T * (weights * yMat)) # 计算回归系数
46     return testPoint * ws
47
48 # 局部加权线性回归测试
49 def lwlrTest(testArr, xArr, yArr, k=1.0):
50     """
51     Parameters:
52         testArr - 测试数据集,测试集
53         xArr - x数据集,训练集
54         yArr - y数据集,训练集
55         k - 高斯核的k,自定义参数
56     Returns:
57         ws - 回归系数
58     """
59     m = np.shape(testArr)[0] # 计算测试数据集大小
60     yHat = np.zeros(m)
61     for i in range(m): # 对每个样本点进行预测
62         yHat[i] = lwlr(testArr[i], xArr, yArr, k)
63     return yHat
64
65 # 计算回归系数w
66 def standRegres(xArr, yArr):
67     """
68     Parameters:
69         xArr - x数据集
70         yArr - y数据集
71     Returns:
72         ws - 回归系数
73     """
74     xMat = np.mat(xArr); yMat = np.mat(yArr).T
75     xTx = xMat.T * xMat # 根据文中推导的公式计算回归系数
76     if np.linalg.det(xTx) == 0.0:
77         print("矩阵为奇异矩阵,不能求逆")
78         return
79     ws = xTx.I * (xMat.T * yMat)
80     return ws
81
82 def rssError(yArr, yHatArr):
83     """
84     误差大小评价函数
85     Parameters:
86         yArr - 真实数据
87         yHatArr - 预测数据
88     Returns:
89         误差大小
90     """
91     return ((yArr - yHatArr)**2).sum()
92
93 if __name__ == '__main__':
94     abX, abY = loadDataSet('abalone.txt')
95     print('训练集与测试集相同:局部加权线性回归,核k的大小对预测的影响:')
96     yHat01 = lwlrTest(abX[0:99], abX[0:99], abY[0:99], 0.1)
97     yHat1 = lwlrTest(abX[0:99], abX[0:99], abY[0:99], 1)
98     yHat10 = lwlrTest(abX[0:99], abX[0:99], abY[0:99], 10)
99     print('k=0.1时,误差大小为:', rssError(abY[0:99], yHat01.T))
100    print('k=1 时,误差大小为:', rssError(abY[0:99], yHat1.T))
101    print('k=10 时,误差大小为:', rssError(abY[0:99], yHat10.T))
102    print('')
103    print('训练集与测试集不同:局部加权线性回归,核k的大小是越小越好吗? 更换数据集,测试结果如下:')
104    yHat01 = lwlrTest(abX[100:199], abX[0:99], abY[0:99], 0.1)
105    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
106    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
107    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
108    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
109    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
110    print('')
111    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
112    yHat01 = lwlrTest(abX[100:199], abX[0:99], abY[100:199], 0.1)
113    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
114    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
115    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
116    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
117    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
118    print('')
119    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
120    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[0:99], 0.1)
121    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[0:99], 1)
122    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[0:99], 10)
123    print('k=0.1时,误差大小为:', rssError(abY[0:99], yHat01.T))
124    print('k=1 时,误差大小为:', rssError(abY[0:99], yHat1.T))
125    print('k=10 时,误差大小为:', rssError(abY[0:99], yHat10.T))
126    print('')
127    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
128    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[0:99], 0.1)
129    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[0:99], 1)
130    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[0:99], 10)
131    print('k=0.1时,误差大小为:', rssError(abY[0:99], yHat01.T))
132    print('k=1 时,误差大小为:', rssError(abY[0:99], yHat1.T))
133    print('k=10 时,误差大小为:', rssError(abY[0:99], yHat10.T))
134    print('')
135    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
136    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
137    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 1)
138    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
139    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
140    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
141    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
142    print('')
143    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
144    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
145    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 1)
146    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
147    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
148    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
149    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
150    print('')
151    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
152    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
153    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 1)
154    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
155    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
156    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
157    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
158    print('')
159    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
160    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
161    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 1)
162    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
163    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
164    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
165    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
166    print('')
167    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
168    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
169    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 1)
170    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
171    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
172    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
173    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
174    print('')
175    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
176    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
177    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 1)
178    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
179    print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
180    print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
181    print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
182    print('')
183    print('训练集与测试集不同:局部加权线性回归,核k的大小是越大越好吗? 更换数据集,测试结果如下:')
184    yHat01 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 0.1)
185    yHat1 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 1)
186    yHat10 = lwlrTest(abX[100:199], abX[100:199], abY[100:199], 10)
187    print('k=0.1时,
```

```

103 yHat1 = lwlrTest(abX[100:199], abX[0:99], abY[0:99], 1)
104 yHat10 = lwlrTest(abX[100:199], abX[0:99], abY[0:99], 10)
105 print('k=0.1时,误差大小为:', rssError(abY[100:199], yHat01.T))
106 print('k=1 时,误差大小为:', rssError(abY[100:199], yHat1.T))
107 print('k=10 时,误差大小为:', rssError(abY[100:199], yHat10.T))
108 print('')
109 print('训练集与测试集不同:简单的线性回归与k=1时的局部加权线性回归对比:')
110 print('k=1时,误差大小为:', rssError(abY[100:199], yHat1.T))
111 ws = standRegres(abX[0:99], abY[0:99])
112 yHat = np.mat(abX[100:199]) * ws
113 print('简单的线性回归误差大小:', rssError(abY[100:199], yHat.T.A))

```

```

1 >>>
2 训练集与测试集相同:局部加权线性回归,核k的大小对预测的影响:
3 k=0.1时,误差大小为: 56.78868743050092
4 k=1 时,误差大小为: 429.89056187038
5 k=10 时,误差大小为: 549.1181708827924
6
7 训练集与测试集不同:局部加权线性回归,核k的大小是越小越好吗? 更换数据集,测试结果如下:
8 k=0.1时,误差大小为: 57913.51550155911
9 k=1 时,误差大小为: 573.5261441895982
10 k=10 时,误差大小为: 517.5711905381903
11
12 训练集与测试集不同:简单的线性回归与k=1时的局部加权线性回归对比:
13 k=1时,误差大小为: 573.5261441895982
14 简单的线性回归误差大小: 518.6363153245542

```

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。这就是经常说道的 **过拟合现象**。我们训练的模型,我们要保证测试集准确率高,这样训练出的模型才可以应用于新的数据,也就是要加强模型的普适性。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。那么最佳的核大小是10吗?或许是,但如果想得到更好的效果,应该用10个不同的样本集做10次测试来比较结果。

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,每次必须要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。

5、缩减系数来“理解”数据

如果数据的特征比样本点还多应该怎么办?是否还可以使用线性回归和之前的方法来做预测?答案是否定的,即不能再使用前面介绍的方法。这是因为在计算 $(X^T X)^{-1}$ 的时候会出错。如果特征比样本点还多 ($n > m$),也就是说输入数据的矩阵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,统计学家引入了 **岭回归 (ridge regression)** 的概念,这就是我们准备介绍的第一种缩减方法。接着是 **lasso 法**,该方法效果很好但计算复杂。最后介绍了第二种缩减方法,称为 **前向逐步回归**,可以得到与lasso差不多的效果,且更容易实现。

1) 岭回归

简单说来,岭回归就是在矩阵 $X^T X$ 上加一个 λI 从而使得矩阵非奇异,进而能对 $X^T X + \lambda I$ 求逆。其中矩阵I是一个m×m的单位矩阵,对角线上元素全为1,其他元素全为0。而λ是一个用户定义的数值,后面会做介绍。在这种情况下,回归系数的计算公式将变成:

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

岭回归最先用来处理特征数多于样本数的情况,现在也用于在估计中加入偏差,从而得到更好的估计。这里通过引入λ来限制了所有w之和,通过引入该惩罚项,能够减少不重要的参数,这个技术在统计学中也叫做 **缩减 (shrinkage)**。

岭回归中的岭是什么?

岭回归使用了单位矩阵乘以常量λ,观察其中的单位矩阵I,可以看到值1贯穿整个对角线,其余元素全是0。形象地,在0构成的平面上有一条1组成的“岭”,这就是岭回归中的“岭”的由来。

缩减方法可以去掉不重要的参数,因此能更好地理解数据。此外,与简单的线性回归相比,缩减法能取得更好的预测效果。

这里通过预测误差最小化得到λ:数据获取之后,首先抽一部分数据用于测试,剩余的作为训练集用于训练参数w。训练完毕后在测试集上测试预测性能。通过选取不同的λ来重复上述测试过程,最终得到一个使预测误差最小的λ。

```

1 from matplotlib.font_manager import FontProperties
2 import matplotlib.pyplot as plt
3

```

```

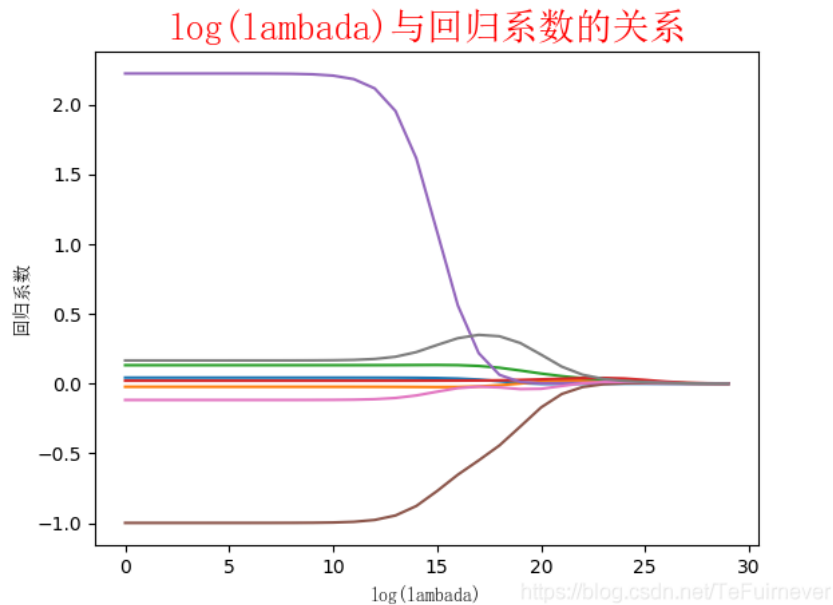
4 import numpy as np
5
6 # 加载数据
7 def loadDataSet(fileName):
8     """
9     Parameters:
10         fileName - 文件名
11     Returns:
12         xArr - x数据集
13         yArr - y数据集
14     """
15     numFeat = len(open(fileName).readline().split('\t')) - 1
16     xArr = []; yArr = []
17     fr = open(fileName)
18     for line in fr.readlines():
19         lineArr = []
20         curLine = line.strip().split('\t')
21         for i in range(numFeat):
22             lineArr.append(float(curLine[i]))
23         xArr.append(lineArr)
24         yArr.append(float(curLine[-1]))
25     return xArr, yArr
26
27 # 岭回归
28 def ridgeRegres(xMat, yMat, lam = 0.2):
29     """
30     Parameters:
31         xMat - x数据集
32         yMat - y数据集
33         lam - 缩减系数
34     Returns:
35         ws - 回归系数
36     """
37     xTx = xMat.T * xMat
38     denom = xTx + np.eye(np.shape(xMat)[1]) * lam
39     if np.linalg.det(denom) == 0.0:
40         print("矩阵为奇异矩阵,不能转置")
41         return
42     ws = denom.I * (xMat.T * yMat)
43     return ws
44
45 # 岭回归测试
46 def ridgeTest(xArr, yArr):
47     """
48     Parameters:
49         xMat - x数据集
50         yMat - y数据集
51     Returns:
52         wMat - 回归系数矩阵
53     """
54     xMat = np.mat(xArr); yMat = np.mat(yArr).T
55     # 数据标准化
56     yMean = np.mean(yMat, axis = 0) # 行与行操作, 求均值
57     yMat = yMat - yMean # 数据减去均值
58     xMeans = np.mean(xMat, axis = 0) # 行与行操作, 求均值
59     xVar = np.var(xMat, axis = 0) # 行与行操作, 求方差
60     xMat = (xMat - xMeans) / xVar # 数据减去均值除以方差实现标准化
61     numTestPts = 30 # 30个不同的Lambda测试
62     wMat = np.zeros((numTestPts, np.shape(xMat)[1])) # 初始回归系数矩阵
63     for i in range(numTestPts): # 改变Lambda计算回归系数
64         ws = ridgeRegres(xMat, yMat, np.exp(i - 10)) # Lambda以e的指数变化, 最初是一个非常小的数,
65         wMat[i, :] = ws.T # 计算回归系数矩阵
66     return wMat
67
68 # 绘制岭回归系数矩阵
69 def plotwMat():
70     font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
71     abX, abY = loadDataSet('abalone.txt')
72     redgeWeights = ridgeTest(abX, abY)
73     fig = plt.figure()
74     ax = fig.add_subplot(111)
75     ax.plot(redgeWeights)
76     ax_title_text = ax.set_title(u'log(lambada)与回归系数的关系', FontProperties = font)
77     ax_xlabel_text = ax.set_xlabel(u'log(lambada)', FontProperties = font)
78
79     # 横轴为log(lambada), 纵轴为回归系数, 横轴为log(lambada)

```

```

75     ax_ylabel_text = ax.set_ylabel(u'回归系数', FontProperties = font)
76     plt.setp(ax_title_text, size = 20, weight = 'bold', color = 'red')
77     plt.setp(ax_xlabel_text, size = 10, weight = 'bold', color = 'black')
78     plt.setp(ax_ylabel_text, size = 10, weight = 'bold', color = 'black')
79     plt.show()
80
81
82 if __name__ == '__main__':
83     plotwMat()

```



运行之后应该看到一个类似上图的结果图，该图绘出了回归系数与 $\log(\lambda)$ 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；在中间部分的某值将可以取得最好的预测效果。为了定量地找到最佳参数值，还需要进行交叉验证。另外，要判断哪些变量对结果预测最具有影响力，在图中观察它们对应的系数大小就可以。

2) 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的一样的公式：

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

上式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当两个或更多的特征相关时，可能会得出一个很大的正系数和一个很大的负系数。正是因为上述限制条件的存在，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：

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

唯一的不同点在于，这个约束条件使用绝对值取代了平方和。虽然约束形式只是稍作变化，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，这个特性可以帮助我们更好地理解数据。这两个约束条件在公式上看起来相差无几，但细微的变化却极大地增加了计算复杂度（为了在这个新的约束条件下解出回归系数，需要使用二次规划算法）。

3) 前向逐步回归

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

该算法的伪代码如下所示：

```

1  数据标准化，使其分布满足0均值和单位方差
2  在每轮迭代过程中：
3      设置当前最小误差lowestError为正无穷
4      对每个特征：
5          增大或缩小：
6              改变一个系数得到一个新的w
7

```



```
8         计算新W下的误差
9         如果误差Error小于当前最小误差lowestError: 设置Wbest等于当前的W
        将W设置为新的Wbest
```

```
1  from matplotlib.font_manager import FontProperties
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # 加载数据
6  def loadDataSet(fileName):
7      """
8      Parameters:
9          fileName - 文件名
10     Returns:
11         xArr - x数据集
12         yArr - y数据集
13     """
14     numFeat = len(open(fileName).readline().split('\t')) - 1
15     xArr = []; yArr = []
16     fr = open(fileName)
17     for line in fr.readlines():
18         lineArr = []
19         curLine = line.strip().split('\t')
20         for i in range(numFeat):
21             lineArr.append(float(curLine[i]))
22         xArr.append(lineArr)
23         yArr.append(float(curLine[-1]))
24     return xArr, yArr
25
26 # 数据标准化
27 def regularize(xMat, yMat):
28     """
29     Parameters:
30         xMat - x数据集
31         yMat - y数据集
32     Returns:
33         inxMat - 标准化后的x数据集
34         inyMat - 标准化后的y数据集
35     """
36     inxMat = xMat.copy() #数据拷贝
37     inyMat = yMat.copy()
38     yMean = np.mean(yMat, 0) #行与行操作, 求均值
39     inyMat = yMat - yMean #数据减去均值
40     inMeans = np.mean(inxMat, 0) #行与行操作, 求均值
41     inVar = np.var(inxMat, 0) #行与行操作, 求方差
42     inxMat = (inxMat - inMeans) / inVar #数据减去均值除以方差实现标准化
43     return inxMat, inyMat
44
45 # 计算平方误差
46 def rssError(yArr, yHatArr):
47     """
48     Parameters:
49         yArr - 预测值
50         yHatArr - 真实值
51     Returns:
52     """
53     return ((yArr - yHatArr)**2).sum()
54
55 # 前向逐步线性回归
56 def stageWise(xArr, yArr, eps = 0.01, numIt = 100):
57     """
58     Parameters:
59         xArr - x输入数据
60         yArr - y预测数据
61         eps - 每次迭代需要调整的步长
62         numIt - 迭代次数
63     Returns:
64         returnMat - numIt次迭代的回归系数矩阵
65     """
66     xMat = np.mat(xArr); yMat = np.mat(yArr).T #数据集
67     xMat, yMat = regularize(xMat, yMat) #数据标准化
68     m, n = np.shape(xMat)
69     returnMat = np.zeros((numIt, n)) #初始化numIt次迭代的回归系数矩阵
70     ws = np.zeros((n, 1)) #初始化回归系数矩阵
```

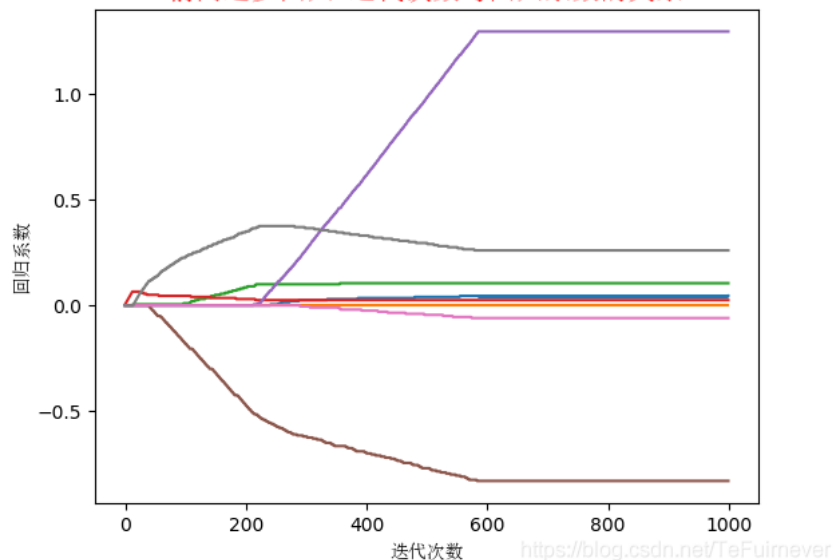


```

67     wsTest = ws.copy()
68     wsMax = ws.copy()
69     for i in range(numIt):
70         # print(ws.T)
71         lowestError = float('inf');
72         for j in range(n):
73             for sign in [-1, 1]:
74                 wsTest = ws.copy()
75                 wsTest[j] += eps * sign
76                 yTest = xMat * wsTest
77                 rssE = rssError(yMat.A, yTest.A)
78                 if rssE < lowestError:
79                     lowestError = rssE
80                     wsMax = wsTest
81     ws = wsMax.copy()
82     returnMat[i, :] = ws.T
83 return returnMat
84
85 # 绘制岭回归系数矩阵
86 def plotstageWiseMat():
87     font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
88     xArr, yArr = loadDataSet('abalone.txt')
89     returnMat = stageWise(xArr, yArr, 0.005, 1000)
90     fig = plt.figure()
91     ax = fig.add_subplot(111)
92     ax.plot(returnMat)
93     ax_title_text = ax.set_title(u'前向逐步回归:迭代次数与回归系数的关系', FontProperties = font)
94     ax_xlabel_text = ax.set_xlabel(u'迭代次数', FontProperties = font)
95     ax_ylabel_text = ax.set_ylabel(u'回归系数', FontProperties = font)
96     plt.setp(ax_title_text, size = 15, weight = 'bold', color = 'red')
97     plt.setp(ax_xlabel_text, size = 10, weight = 'bold', color = 'black')
98     plt.setp(ax_ylabel_text, size = 10, weight = 'bold', color = 'black')
99     plt.show()
100
101 if __name__ == '__main__':
102     plotstageWiseMat()
103

```

前向逐步回归:迭代次数与回归系数的关系



逐步线性回归算法的实际好处并不在于能绘出这样漂亮的图，主要的优点在于它可以帮助人们理解现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。最后，如果用于测试，该算法每100次迭代后就可以构建出一个模型，可以使用类似于10折交叉验证的方法比较这些模型，最终选择使误差最小的模型。

当应用缩减方法（如逐步线性回归或岭回归）时，模型也就增加了偏差（bias），与此同时却减小了模型的方差。下一节将揭示这些概念之间的关系并分析它们对结果的影响。

6、示例：预测乐高玩具套装的价格

你对乐高（LEGO）品牌的玩具了解吗？乐高公司生产拼装类玩具，由很多大小不同的塑料插块组成。这些塑料插块的设计非常出色，不需要任何粘合剂就可以随意拼装起来。除了简单玩具之外，乐高玩具在一些成人中也很流行。一般来说，这些插块都成套出售，它们可以拼装成很多不同的东西，如船、城堡、一些著名建筑，等等。乐高公司每个套装包含的部件数目从10件到5000件不等。一种乐高套装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。Dangler喜欢为乐高套装估价，下面将用本章的回归技术帮助他建立一个预测模型。

示例：用回归法预测乐高套装的价格

- (1) 收集数据：用Google Shopping的API收集数据。
- (2) 准备数据：从返回的JSON数据中抽取价格。
- (3) 分析数据：可视化并观察数据。
- (4) 训练算法：构建不同的模型，采用逐步线性回归和直接的线性回归模型。
- (5) 测试算法：使用交叉验证来测试不同的模型，分析哪个效果最好。
- (6) 使用算法：这次练习的目标就是生成数据模型。

1) 收集数据

```
1  from bs4 import BeautifulSoup
2
3  # 从页面读取数据，生成retX和retY列表
4  def scrapePage(retX, retY, inFile, yr, numPce, origPrc):
5      """
6      Parameters:
7          retX - 数据X
8          retY - 数据Y
9          inFile - HTML文件
10         yr - 年份
11         numPce - 乐高部件数目
12         origPrc - 原价
13     Returns:
14         无
15     """
16     # 打开并读取HTML文件
17     with open(inFile, encoding='utf-8') as f:
18         html = f.read()
19     soup = BeautifulSoup(html)
20     i = 1
21     # 根据HTML页面结构进行解析
22     currentRow = soup.find_all('table', r="%d" % i)
23     while (len(currentRow) != 0):
24         currentRow = soup.find_all('table', r="%d" % i)
25         title = currentRow[0].find_all('a')[1].text
26         lwrTitle = title.lower()
27         # 查找是否有全新标签
28         if (lwrTitle.find('new') > -1) or (lwrTitle.find('nisb') > -1):
29             newFlag = 1.0
30         else:
31             newFlag = 0.0
32         # 查找是否已经标志出售，我们只收集已出售的数据
33         soldUnicde = currentRow[0].find_all('td')[3].find_all('span')
34         if len(soldUnicde) == 0:
35             print("商品 #%d 没有出售" % i)
36         else:
37             # 解析页面获取当前价格
38             soldPrice = currentRow[0].find_all('td')[4]
39             priceStr = soldPrice.text
40             priceStr = priceStr.replace('$', '')
41             priceStr = priceStr.replace(',', '')
42             if len(soldPrice) > 1:
43                 priceStr = priceStr.replace('Free shipping', '')
44             sellingPrice = float(priceStr)
45             # 去掉不完整的套装价格
46             if sellingPrice > origPrc * 0.5:
47                 print("%d\t%d\t%d\t%f\t%f" % (yr, numPce, newFlag, origPrc, sellingPrice))
48                 retX.append([yr, numPce, newFlag, origPrc])
49                 retY.append(sellingPrice)
50             i += 1
51             currentRow = soup.find_all('table', r="%d" % i)
52
53 # 依次读取六种乐高套装的数据，并生成数据矩阵
54 def setDataCollect(retX, retY):
55     """
56     2006年的乐高2000 部件数目2000 原价10.00
57     """
```

```

53         # 2006年的乐高8288, 部件数目800, 原价49.99
54         scrapePage(retX, retY, './setHtml/lego8288.html', 2006, 800, 49.99)
55         # 2002年的乐高10030, 部件数目3096, 原价269.99
56         scrapePage(retX, retY, './setHtml/lego10030.html', 2002, 3096, 269.99)
57         # 2007年的乐高10179, 部件数目5195, 原价499.99
58         scrapePage(retX, retY, './setHtml/lego10179.html', 2007, 5195, 499.99)
59         # 2007年的乐高10181, 部件数目3428, 原价199.99
60         scrapePage(retX, retY, './setHtml/lego10181.html', 2007, 3428, 199.99)
61         # 2008年的乐高10189, 部件数目5922, 原价299.99
62         scrapePage(retX, retY, './setHtml/lego10189.html', 2008, 5922, 299.99)
63         # 2009年的乐高10196, 部件数目3263, 原价249.99
64         scrapePage(retX, retY, './setHtml/lego10196.html', 2009, 3263, 249.99)
65
66
67     if __name__ == '__main__':
68         lgX = []
69         lgY = []
70         setDataCollect(lgX, lgY)

```

部分结果如下：

```

2008→5922→0→299.990000→510.000000
2008→5922→0→299.990000→423.000000
商品 #6 没有出售
商品 #7 没有出售
2008→5922→1→299.990000→599.990000
商品 #9 没有出售
2008→5922→1→299.990000→589.990000
2008→5922→1→299.990000→569.990000
2008→5922→1→299.990000→529.990000
2008→5922→0→299.990000→500.000000
2008→5922→1→299.990000→549.950000
2008→5922→0→299.990000→300.000000
商品 #16 没有出售
2009→3263→1→249.990000→380.000000
2009→3263→1→249.990000→399.000000
2009→3263→1→249.990000→427.990000
2009→3263→0→249.990000→360.000000
商品 #5 没有出售
商品 #6 没有出售
2009→3263→1→249.990000→399.000000
2009→3263→1→249.990000→399.950000
2009→3263→1→249.990000→499.990000
商品 #10 没有出售
2009→3263→0→249.990000→399.950000
商品 #12 没有出售
2009→3263→1→249.990000→331.510000

```

我们对没有的商品做了处理。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）。

2) 训练算法：建立模型

上一节从网上收集到了一些真实的数据，下面将为这些数据构建一个模型。构建出的模型可以对售价做出预测，并帮助我们理解现有数据。

```

1  import numpy as np
2  from bs4 import BeautifulSoup
3
4  # 页面读取数据, 生成retX和retY列表
5  def scrapePage(retX, retY, inFile, yr, numPce, origPrc):
6      """
7      Parameters:
8          retX - 数据X
9          retY - 数据Y
10         inFile - HTML文件
11         yr - 年份
12         numPce - 乐高部件数目
13         origPrc - 原价
14
15     Returns:
16         无
17     """
18     # 打开并读取HTML文件
19     with open(inFile, encoding='utf-8') as f:
20         html = f.read()

```

```

18 soup = BeautifulSoup(html)
19 i = 1
20 # 根据HTML页面结构进行解析
21 currentRow = soup.find_all('table', r = "%d" % i)
22 while(len(currentRow) != 0):
23     currentRow = soup.find_all('table', r = "%d" % i)
24     title = currentRow[0].find_all('a')[1].text
25     lwrTitle = title.lower()
26     # 查找是否有全新标签
27     if (lwrTitle.find('new') > -1) or (lwrTitle.find('nissb') > -1):
28         newFlag = 1.0
29     else:
30         newFlag = 0.0
31     # 查找是否已经标志出售, 我们只收集已售出的数据
32     soldUnicode = currentRow[0].find_all('td')[3].find_all('span')
33     if len(soldUnicode) == 0:
34         print("商品 #%d 没有出售" % i)
35     else:
36         # 解析页面获取当前价格
37         soldPrice = currentRow[0].find_all('td')[4]
38         priceStr = soldPrice.text
39         priceStr = priceStr.replace('$', '')
40         priceStr = priceStr.replace(',', '')
41         if len(soldPrice) > 1:
42             priceStr = priceStr.replace('Free shipping', '')
43             sellingPrice = float(priceStr)
44             # 去掉不完整的套装价格
45             if sellingPrice > origPrc * 0.5:
46                 print("%d\t%d\t%d\t%f\t%f" % (yr, numPce, newFlag, origPrc, sellingPrice))
47                 retX.append([yr, numPce, newFlag, origPrc])
48                 retY.append(sellingPrice)
49             i += 1
50             currentRow = soup.find_all('table', r = "%d" % i)
51
52 # 依次读取六种乐高套装的数据, 并生成数据矩阵
53 def setDataCollect(retX, retY):
54     # 2006年的乐高8288, 部件数目800, 原价49.99
55     scrapePage(retX, retY, './setHtml/lego8288.html', 2006, 800, 49.99)
56     # 2002年的乐高10030, 部件数目3096, 原价269.99
57     scrapePage(retX, retY, './setHtml/lego10030.html', 2002, 3096, 269.99)
58     # 2007年的乐高10179, 部件数目5195, 原价499.99
59     scrapePage(retX, retY, './setHtml/lego10179.html', 2007, 5195, 499.99)
60     # 2007年的乐高10181, 部件数目3428, 原价199.99
61     scrapePage(retX, retY, './setHtml/lego10181.html', 2007, 3428, 199.99)
62     # 2008年的乐高10189, 部件数目5922, 原价299.99
63     scrapePage(retX, retY, './setHtml/lego10189.html', 2008, 5922, 299.99)
64     # 2009年的乐高10196, 部件数目3263, 原价249.99
65     scrapePage(retX, retY, './setHtml/lego10196.html', 2009, 3263, 249.99)
66
67 # 数据标准化
68 def regularize(xMat, yMat):
69     """
70     Parameters:
71         xMat - x数据集
72         yMat - y数据集
73     Returns:
74         inxMat - 标准化后的x数据集
75         inyMat - 标准化后的y数据集
76     """
77     inxMat = xMat.copy() #数据拷贝
78     inyMat = yMat.copy()
79     yMean = np.mean(yMat, 0) #行与行操作, 求均值
80     inyMat = yMat - yMean #数据减去均值
81     inMeans = np.mean(inxMat, 0) #行与行操作, 求均值
82     inVar = np.var(inxMat, 0) #行与行操作, 求方差
83     # print(inxMat)
84     print(inMeans)
85     # print(inVar)
86     inxMat = (inxMat - inMeans) / inVar #数据减去均值除以方差实现标准化
87     return inxMat, inyMat
88
89 # 计算平方误差

```

```

89  def rssError(yArr,yHatArr):
90      """
91      Parameters:
92          yArr - 预测值
93          yHatArr - 真实值
94      Returns:
95      """
96      return ((yArr-yHatArr)**2).sum()
97
98  # 计算回归系数w
99  def standRegres(xArr,yArr):
100      """
101      Parameters:
102          xArr - x数据集
103          yArr - y数据集
104      Returns:
105          ws - 回归系数
106      """
107      xMat = np.mat(xArr); yMat = np.mat(yArr).T
108      xTx = xMat.T * xMat #根据文中推导的公式计算回归系数
109      if np.linalg.det(xTx) == 0.0:
110          print("矩阵为奇异矩阵,不能转置")
111          return
112      ws = xTx.I * (xMat.T*yMat)
113      return ws
114
115  # 使用简单的线性回归
116  def useStandRegres():
117      lgX = []
118      lgY = []
119      setDataCollect(lgX, lgY)
120      data_num, features_num = np.shape(lgX)
121      lgX1 = np.mat(np.ones((data_num, features_num + 1)))
122      lgX1[:, 1:5] = np.mat(lgX)
123      ws = standRegres(lgX1, lgY)
124      print(' %f*年份+ %f*部件数量+ %f*是否为全新+ %f*原价' % (ws[0],ws[1],ws[2],ws[3],ws[4]))
125
126  if __name__ == '__main__':
127      useStandRegres()

```

```

商品 #2 没有出售
2008 → 5922 → 1 → 299.990000 → 599.950000
2008 → 5922 → 0 → 299.990000 → 510.000000
2008 → 5922 → 0 → 299.990000 → 423.000000
商品 #6 没有出售
商品 #7 没有出售
2008 → 5922 → 1 → 299.990000 → 599.990000
商品 #9 没有出售
2008 → 5922 → 1 → 299.990000 → 589.990000
2008 → 5922 → 1 → 299.990000 → 569.990000
2008 → 5922 → 1 → 299.990000 → 529.990000
2008 → 5922 → 0 → 299.990000 → 500.000000
2008 → 5922 → 1 → 299.990000 → 549.950000
2008 → 5922 → 0 → 299.990000 → 300.000000
商品 #16 没有出售
2009 → 3263 → 1 → 249.990000 → 380.000000
2009 → 3263 → 1 → 249.990000 → 399.000000
2009 → 3263 → 1 → 249.990000 → 427.990000
2009 → 3263 → 0 → 249.990000 → 360.000000
商品 #5 没有出售
商品 #6 没有出售
2009 → 3263 → 1 → 249.990000 → 399.000000
2009 → 3263 → 1 → 249.990000 → 399.950000
2009 → 3263 → 1 → 249.990000 → 499.990000
商品 #10 没有出售
2009 → 3263 → 0 → 249.990000 → 399.950000
商品 #12 没有出售
2009 → 3263 → 1 → 249.990000 → 331.510000
55319.970081-27.592822*年份-0.026839*部件数量-11.220848*是否为全新+29576041*原价

```

这个模型的预测效果非常好，但模型本身并不能令人满意。它对于数据拟合得很好，但看上去没有什么道理。从公式看，套装里零部件越多售价反而会越低。另外，该公式对新套装也有一定的惩罚。

```
55319.970081-27.592822*年份-0.026839*部件数量+11.220848*是否为全新+2.576041*原价
```

下面使用缩减法中一种，即岭回归再进行一次实验，通过交叉验证，找到使误差最小的 λ 对应的回归系数。

```
1 import numpy as np
2 from bs4 import BeautifulSoup
3 import random
4
5 # 从页面读取数据，生成retX和retY列表
6 def scrapePage(retX, retY, inFile, yr, numPce, origPrc):
7     """
8     Parameters:
9         retX - 数据X
10        retY - 数据Y
11        inFile - HTML文件
12        yr - 年份
13        numPce - 乐高部件数目
14        origPrc - 原价
15    Returns:
16        无
17    """
18    # 打开并读取HTML文件
19    with open(inFile, encoding='utf-8') as f:
20        html = f.read()
21    soup = BeautifulSoup(html)
22    i = 1
23    # 根据HTML页面结构进行解析
24    currentRow = soup.find_all('table', r = "%d" % i)
25    while(len(currentRow) != 0):
26        currentRow = soup.find_all('table', r = "%d" % i)
27        title = currentRow[0].find_all('a')[1].text
28        lwrTitle = title.lower()
29        # 查找是否有全新标签
30        if (lwrTitle.find('new') > -1) or (lwrTitle.find('nibb') > -1):
31            newFlag = 1.0
32        else:
33            newFlag = 0.0
34        # 查找是否已经标志出售，我们只收集已售出的数据
35        soldUnicde = currentRow[0].find_all('td')[3].find_all('span')
36        if len(soldUnicde) == 0:
37            print("商品 #%d 没有出售" % i)
38        else:
39            # 解析页面获取当前价格
40            soldPrice = currentRow[0].find_all('td')[4]
41            priceStr = soldPrice.text
42            priceStr = priceStr.replace('$', '')
43            priceStr = priceStr.replace(',', '')
44            if len(soldPrice) > 1:
45                priceStr = priceStr.replace('Free shipping', '')
46            sellingPrice = float(priceStr)
47            # 去掉不完整的套装价格
48            if sellingPrice > origPrc * 0.5:
49                print("%d\t%d\t%d\t%f\t%f" % (yr, numPce, newFlag, origPrc, sellingPrice))
50                retX.append([yr, numPce, newFlag, origPrc])
51                retY.append(sellingPrice)
52            i += 1
53            currentRow = soup.find_all('table', r = "%d" % i)
54
55 # 岭回归
56 def ridgeRegres(xMat, yMat, lam = 0.2):
57     """
58     Parameters:
59         xMat - x数据集
60         yMat - y数据集
61         lam - 缩减系数
62    Returns:
63        ws - 回归系数
64    """
65    xTx = xMat.T * xMat
66    denom = xTx + np.eye(np.shape(xMat)[1]) * lam
67    if np.linalg.det(denom) == 0.0:
68        print("矩阵为奇异矩阵，不能转置")
```

```

66         return
67     ws = denom.I * (xMat.T * yMat)
68     return ws
69
70 # 依次读取六种乐高套装的数据，并生成数据矩阵
71 def setDataCollect(retX, retY):
72     # 2006年的乐高8288, 部件数目800, 原价49.99
73     scrapePage(retX, retY, './setHtml/lego8288.html', 2006, 800, 49.99)
74     # 2002年的乐高10030, 部件数目3096, 原价269.99
75     scrapePage(retX, retY, './setHtml/lego10030.html', 2002, 3096, 269.99)
76     # 2007年的乐高10179, 部件数目5195, 原价499.99
77     scrapePage(retX, retY, './setHtml/lego10179.html', 2007, 5195, 499.99)
78     # 2007年的乐高10181, 部件数目3428, 原价199.99
79     scrapePage(retX, retY, './setHtml/lego10181.html', 2007, 3428, 199.99)
80     # 2008年的乐高10189, 部件数目5922, 原价299.99
81     scrapePage(retX, retY, './setHtml/lego10189.html', 2008, 5922, 299.99)
82     # 2009年的乐高10196, 部件数目3263, 原价249.99
83     scrapePage(retX, retY, './setHtml/lego10196.html', 2009, 3263, 249.99)
84
85 # 数据标准化
86 def regularize(xMat, yMat):
87     """
88     Parameters:
89         xMat - x数据集
90         yMat - y数据集
91     Returns:
92         inxMat - 标准化后的x数据集
93         inyMat - 标准化后的y数据集
94     """
95     inxMat = xMat.copy() #数据拷贝
96     inyMat = yMat.copy()
97     yMean = np.mean(yMat, 0) #行与行操作, 求均值
98     inyMat = yMat - yMean #数据减去均值
99     inMeans = np.mean(inxMat, 0) #行与行操作, 求均值
100     inVar = np.var(inxMat, 0) #行与行操作, 求方差
101     # print(inxMat)
102     print(inMeans)
103     # print(inVar)
104     inxMat = (inxMat - inMeans) / inVar #数据减去均值除以方差实现标准化
105     return inxMat, inyMat
106
107 # 计算平方误差
108 def rssError(yArr, yHatArr):
109     """
110     Parameters:
111         yArr - 预测值
112         yHatArr - 真实值
113     Returns:
114     """
115     return ((yArr - yHatArr)**2).sum()
116
117 # 计算回归系数w
118 def standRegres(xArr, yArr):
119     """
120     Parameters:
121         xArr - x数据集
122         yArr - y数据集
123     Returns:
124         ws - 回归系数
125     """
126     xMat = np.mat(xArr); yMat = np.mat(yArr).T
127     xTx = xMat.T * xMat #根据文中推导的公式计算回归系数
128     if np.linalg.det(xTx) == 0.0:
129         print("矩阵为奇异矩阵, 不能转置")
130         return
131     ws = xTx.I * (xMat.T * yMat)
132     return ws
133
134 # 交叉验证岭回归
135 def crossValidation(xArr, yArr, numVal = 10):
136     """
137     Parameters:
138         xArr - x数据集
139         yArr - y数据集

```



```

137     numVal - 交叉验证次数
138 Returns:
139     wMat - 回归系数矩阵
140 """
141 m = len(yArr) #统计样本个数
142 indexList = list(range(m)) #生成索引值列表
143 errorMat = np.zeros((numVal,30)) #create error mat 30columns numVal rows
144 for i in range(numVal): #交叉验证numVal次
145     trainX = []; trainY = [] #训练集
146     testX = []; testY = [] #测试集
147     random.shuffle(indexList) #打乱次序
148     for j in range(m): #划分数据集:90%训练集, 10%测试集
149         if j < m * 0.9:
150             trainX.append(xArr[indexList[j]])
151             trainY.append(yArr[indexList[j]])
152         else:
153             testX.append(xArr[indexList[j]])
154             testY.append(yArr[indexList[j]])
155     wMat = ridgeTest(trainX, trainY) #获得30个不同Lambda下的岭回归系数
156     for k in range(30): #遍历所有的岭回归系数
157         matTestX = np.mat(testX); matTrainX = np.mat(trainX) #测试集
158         meanTrain = np.mean(matTrainX,0) #测试集均值
159         varTrain = np.var(matTrainX,0) #测试集方差
160         matTestX = (matTestX - meanTrain) / varTrain #测试集标准化
161         yEst = matTestX * np.mat(wMat[k,:]).T + np.mean(trainY) #根据ws预测值
162         errorMat[i, k] = rssError(yEst.T.A, np.array(testY)) #统计误差
163     meanErrors = np.mean(errorMat,0) #计算每次交叉验证的平均误差
164     minMean = float(min(meanErrors)) #找到最小误差
165     bestWeights = wMat[np.nonzero(meanErrors == minMean)] #找到最佳回归系数
166     xMat = np.mat(xArr); yMat = np.mat(yArr).T
167     meanX = np.mean(xMat,0); varX = np.var(xMat,0)
168     unReg = bestWeights / varX #数据经过标准化, 因此需要还原
169     print(' %f*年份+%f*部件数量+%f*是否为全新+%f*原价' % ((-1 * np.sum(np.multiply(meanX,unReg)) + np.mean(yMat)), unReg[0,0], u
170 # 岭回归测试
171 def ridgeTest(xArr, yArr):
172     """
173     Parameters:
174         xMat - x数据集
175         yMat - y数据集
176     Returns:
177         wMat - 回归系数矩阵
178     """
179     xMat = np.mat(xArr); yMat = np.mat(yArr).T
180     #数据标准化
181     yMean = np.mean(yMat, axis = 0) #行与行操作, 求均值
182     yMat = yMat - yMean #数据减去均值
183     xMeans = np.mean(xMat, axis = 0) #行与行操作, 求均值
184     xVar = np.var(xMat, axis = 0) #行与行操作, 求方差
185     xMat = (xMat - xMeans) / xVar #数据减去均值除以方差实现标准化
186     numTestPts = 30 #30个不同的Lambda测试
187     wMat = np.zeros((numTestPts, np.shape(xMat)[1])) #初始回归系数矩阵
188     for i in range(numTestPts): #改变Lambda计算回归系数
189         ws = ridgeRegres(xMat, yMat, np.exp(i - 10)) #Lambda以e的指数变化, 最初是一个非常小的数,
190         wMat[i, :] = ws.T #计算回归系数矩阵
191     return wMat
192
193 if __name__ == '__main__':
194     lgX = []
195     lgY = []
196     setDataCollect(lgX, lgY)
197     crossValidation(lgX, lgY)
198
199

```



```

2008 5922 1 299.990000 599.950000
2008 5922 0 299.990000 510.000000
2008 5922 0 299.990000 423.000000
商品 #6 没有出售
商品 #7 没有出售
2008 5922 1 299.990000 599.990000
商品 #9 没有出售
2008 5922 1 299.990000 589.990000
2008 5922 1 299.990000 569.990000
2008 5922 1 299.990000 529.990000
2008 5922 0 299.990000 500.000000
2008 5922 1 299.990000 549.950000
2008 5922 0 299.990000 300.000000
商品 #16 没有出售
2009 3263 1 249.990000 380.000000
2009 3263 1 249.990000 399.000000
2009 3263 1 249.990000 427.990000
2009 3263 0 249.990000 360.000000
商品 #5 没有出售
商品 #6 没有出售
2009 3263 1 249.990000 399.000000
2009 3263 1 249.990000 399.950000
2009 3263 1 249.990000 499.990000
商品 #10 没有出售
2009 3263 0 249.990000 399.950000
商品 #12 没有出售
2009 3263 1 249.990000 331.510000
66003.316441-32.903288*年份+0.000610*部件数量-12.232855*是否为全新+2.143449*原价

```

这里随机选取样本，因为其随机性，所以每次运行的结果可能略有不同。不过整体如上图所示，可以看出，它与常规的最小二乘法，即普通的线性回归没有太大差异。我们本期望找到一个更易于理解的模型，显然没有达到预期效果。

现在，我们看一下在缩减过程中回归系数是如何变化的。编写代码如下：

```

1  import numpy as np
2  from bs4 import BeautifulSoup
3  import random
4
5  # 从页面读取数据，生成retX和retY列表
6  def scrapePage(retX, retY, inFile, yr, numPce, origPrc):
7      """
8      Parameters:
9          retX - 数据X
10         retY - 数据Y
11         inFile - HTML文件
12         yr - 年份
13         numPce - 乐高部件数目
14         origPrc - 原价
15     Returns:
16         无
17     """
18     # 打开并读取HTML文件
19     with open(inFile, encoding='utf-8') as f:
20         html = f.read()
21         soup = BeautifulSoup(html)
22         i = 1
23         # 根据HTML页面结构进行解析
24         currentRow = soup.find_all('table', r = "%d" % i)
25         while(len(currentRow) != 0):
26             currentRow = soup.find_all('table', r = "%d" % i)
27             title = currentRow[0].find_all('a')[1].text
28             lwrTitle = title.lower()
29             # 查找是否有全新标签
30             if (lwrTitle.find('new') > -1) or (lwrTitle.find('nisb') > -1):
31                 newFlag = 1.0
32             else:
33                 newFlag = 0.0
34             # 查找是否已经标志出售，我们只收集已出售的数据
35             soldUnicode = currentRow[0].find_all('td')[3].find_all('span')
36             if len(soldUnicode) == 0:
37                 print("商品 #%d 没有出售" % i)
38             else:
39                 # 解析页面获取当前价格
40                 soldPrice = currentRow[0].find_all('td')[4]

```

```

40     priceStr = soldPrice.text
41     priceStr = priceStr.replace('$', '')
42     priceStr = priceStr.replace(',', '')
43     if len(soldPrice) > 1:
44         priceStr = priceStr.replace('Free shipping', '')
45     sellingPrice = float(priceStr)
46     # 去掉不完整的套装价格
47     if sellingPrice > origPrc * 0.5:
48         print("%d\t%d\t%d\t%f\t%f" % (yr, numPce, newFlag, origPrc, sellingPrice))
49         retX.append([yr, numPce, newFlag, origPrc])
50         retY.append(sellingPrice)
51     i += 1
52     currentRow = soup.find_all('table', r = "%d" % i)
53
54 # 岭回归
55 def ridgeRegres(xMat, yMat, lam = 0.2):
56     """
57     Parameters:
58         xMat - x数据集
59         yMat - y数据集
60         lam - 缩减系数
61     Returns:
62         ws - 回归系数
63     """
64     xTx = xMat.T * xMat
65     denom = xTx + np.eye(np.shape(xMat)[1]) * lam
66     if np.linalg.det(denom) == 0.0:
67         print("矩阵为奇异矩阵,不能转置")
68         return
69     ws = denom.I * (xMat.T * yMat)
70     return ws
71
72 # 依次读取六种乐高套装的数据, 并生成数据矩阵
73 def setDataCollect(retX, retY):
74     # 2006年的乐高8288, 部件数目800, 原价49.99
75     scrapePage(retX, retY, './setHtml/lego8288.html', 2006, 800, 49.99)
76     # 2002年的乐高10030, 部件数目3096, 原价269.99
77     scrapePage(retX, retY, './setHtml/lego10030.html', 2002, 3096, 269.99)
78     # 2007年的乐高10179, 部件数目5195, 原价499.99
79     scrapePage(retX, retY, './setHtml/lego10179.html', 2007, 5195, 499.99)
80     # 2007年的乐高10181, 部件数目3428, 原价199.99
81     scrapePage(retX, retY, './setHtml/lego10181.html', 2007, 3428, 199.99)
82     # 2008年的乐高10189, 部件数目5922, 原价299.99
83     scrapePage(retX, retY, './setHtml/lego10189.html', 2008, 5922, 299.99)
84     # 2009年的乐高10196, 部件数目3263, 原价249.99
85     scrapePage(retX, retY, './setHtml/lego10196.html', 2009, 3263, 249.99)
86
87 # 数据标准化
88 def regularize(xMat, yMat):
89     """
90     Parameters:
91         xMat - x数据集
92         yMat - y数据集
93     Returns:
94         inxMat - 标准化后的x数据集
95         inyMat - 标准化后的y数据集
96     """
97     inxMat = xMat.copy() #数据拷贝
98     inyMat = yMat.copy()
99     yMean = np.mean(yMat, 0) #行与行操作, 求均值
100     inyMat = yMat - yMean #数据减去均值
101     inMeans = np.mean(inxMat, 0) #行与行操作, 求均值
102     inVar = np.var(inxMat, 0) #行与行操作, 求方差
103     # print(inxMat)
104     print(inMeans)
105     # print(inVar)
106     inxMat = (inxMat - inMeans) / inVar #数据减去均值除以方差实现标准化
107     return inxMat, inyMat
108
109 # 计算平方误差
110 def rssError(yArr, yHatArr):
111     """
112     Parameters:
113         yArr - 预测值
114         yHatArr - 实际值

```

```

111         ydataArr = 吴大昌
112     Returns:
113         """
114     return ((yArr-yHatArr)**2).sum()
115
116 # 计算回归系数w
117 def standRegres(xArr,yArr):
118     """
119     Parameters:
120         xArr - x数据集
121         yArr - y数据集
122     Returns:
123         ws - 回归系数
124         """
125     xMat = np.mat(xArr); yMat = np.mat(yArr).T
126     xTx = xMat.T * xMat #根据文中推导的公式计算回归系数
127     if np.linalg.det(xTx) == 0.0:
128         print("矩阵为奇异矩阵,不能转置")
129     return
130     ws = xTx.I * (xMat.T*yMat)
131     return ws
132
133 # 岭回归测试
134 def ridgeTest(xArr, yArr):
135     """
136     Parameters:
137         xMat - x数据集
138         yMat - y数据集
139     Returns:
140         wMat - 回归系数矩阵
141         """
142     xMat = np.mat(xArr);
143     yMat = np.mat(yArr).T
144     # 数据标准化
145     yMean = np.mean(yMat, axis=0) # 行与行操作, 求均值
146     yMat = yMat - yMean # 数据减去均值
147     xMeans = np.mean(xMat, axis=0) # 行与行操作, 求均值
148     xVar = np.var(xMat, axis=0) # 行与行操作, 求方差
149     xMat = (xMat - xMeans) / xVar # 数据减去均值除以方差实现标准化
150     numTestPts = 30 # 30个不同的Lambda测试
151     wMat = np.zeros((numTestPts, np.shape(xMat)[1])) # 初始回归系数矩阵
152     for i in range(numTestPts): # 改变Lambda计算回归系数
153         ws = ridgeRegres(xMat, yMat, np.exp(i - 10)) # Lambda以e的指数变化, 最初是一个非常小的数,
154         wMat[i, :] = ws.T # 计算回归系数矩阵
155     return wMat
156
157 if __name__ == '__main__':
158     lgX = []
159     lgY = []
160     setDataCollect(lgX, lgY)
161     print(ridgeTest(lgX, lgY))

```

```

[-1.44450432e+02  8.55488076e+02 -1.35089285e+00  4.00885735e+04]
[-1.37402474e+02  1.64217093e+03  1.95840783e+00  3.44932120e+04]
[-1.24750588e+02  1.44326171e+03  7.62540167e+00  2.50647592e+04]
[-1.10234679e+02  8.81842164e+02  1.40617304e+01  1.43874420e+04]
[-9.96484167e+01  4.17805568e+02  1.87140361e+01  6.66770425e+03]
[-9.40345090e+01  1.71289137e+02  2.10844952e+01  2.71206176e+03]
[-9.11400659e+01  6.57287394e+01  2.20487105e+01  1.03800465e+03]
[-8.86246985e+01  2.45452725e+01  2.23181664e+01  3.87564774e+02]
[-8.41447674e+01  9.05861459e+00  2.21495534e+01  1.43313895e+02]
[-7.44804291e+01  3.31863501e+00  2.14607512e+01  5.27810178e+01]
[-5.68008473e+01  1.20770663e+00  2.00168153e+01  1.93999701e+01]
[-3.43546503e+01  4.38238026e-01  1.77836684e+01  7.12719906e+00]
[-1.62951276e+01  1.59882766e-01  1.48514658e+01  2.62234165e+00]
[-6.48291858e+00  5.89025383e-02  1.09847950e+01  9.67404902e-01]
[-2.35268585e+00  2.18391027e-02  6.61152257e+00  3.57478187e-01]
[-8.35001919e-01  8.09519290e-03  3.19552087e+00  1.32007993e-01]
[-2.99711902e-01  2.99108326e-03  1.33043325e+00  4.86659524e-02]
[-1.08982743e-01  1.10249682e-03  5.14449554e-01  1.79199150e-02]
[-3.99038878e-02  4.05898142e-04  1.92885670e-01  6.59479857e-03]
[-1.46534283e-02  1.49365062e-04  7.14631760e-02  2.42642878e-03]
[-5.38707897e-03  5.49542829e-05  2.63587872e-02  8.92679468e-04]
[-1.98130399e-03  2.02173589e-05  9.70622185e-03  3.28404700e-04]
[-7.28814363e-04  7.43766021e-06  3.57198872e-03  1.20814188e-04]
[-2.68106796e-04  2.73617711e-06  1.31423307e-03  4.44451712e-05]
[-9.86297565e-05  1.00658531e-06  4.83502591e-04  1.63504803e-05]
[-3.62836944e-05  3.70302314e-07  1.77873811e-04  6.01500768e-06]
[-1.33480028e-05  1.36226645e-07  6.54365445e-05  2.21279795e-06]
[-4.91045279e-06  5.01149871e-08  2.40728171e-05  8.91404291e-07]

```

这些系数是经过不同程度的缩减得到的。首先看第1行，第4项比第2项的系数大5倍，比第1项大57倍。这样看来，如果只能选择一个特征来做预测的话，我们应该选择第4个特征，也就是原始价格。如果可以选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，而哪些特征是不重要的。

7、Sklearn构建岭回归

```

1  import numpy as np
2  from bs4 import BeautifulSoup
3  import random
4
5  # 从页面读取数据，生成retX和retY列表
6  def scrapePage(retX, retY, inFile, yr, numPce, origPrc):
7      """
8      Parameters:
9          retX - 数据X
10         retY - 数据Y
11         inFile - HTML文件
12         yr - 年份
13         numPce - 乐高部件数目
14         origPrc - 原价
15     Returns:
16         无
17     """
18     # 打开并读取HTML文件
19     with open(inFile, encoding='utf-8') as f:
20         html = f.read()
21         soup = BeautifulSoup(html)
22         i = 1
23         # 根据HTML页面结构进行解析
24         currentRow = soup.find_all('table', r = "%d" % i)
25         while(len(currentRow) != 0):
26             currentRow = soup.find_all('table', r = "%d" % i)
27             title = currentRow[0].find_all('a')[1].text
28             lwrTitle = title.lower()
29             # 查找是否有全新标签
30             if (lwrTitle.find('new') > -1) or (lwrTitle.find('nib') > -1):
31                 newFlag = 1.0
32             else:
33                 newFlag = 0.0
34             # 查找是否已经标志出售，我们只收集已售出的数据
35             soldUnicode = currentRow[0].find_all('td')[3].find_all('span')
36             if len(soldUnicode) == 0:
37                 print("商品 %d 没有出售" % i)

```

```

36     else:
37         # 解析页面获取当前价格
38         soldPrice = currentRow[0].find_all('td')[4]
39         priceStr = soldPrice.text
40         priceStr = priceStr.replace('$', '')
41         priceStr = priceStr.replace(',', '')
42         if len(soldPrice) > 1:
43             priceStr = priceStr.replace('Free shipping', '')
44         sellingPrice = float(priceStr)
45         # 去掉不完整的套装价格
46         if sellingPrice > origPrc * 0.5:
47             print("%d\t%d\t%d\t%f\t%f" % (yr, numPce, newFlag, origPrc, sellingPrice))
48             retX.append([yr, numPce, newFlag, origPrc])
49             retY.append(sellingPrice)
50         i += 1
51         currentRow = soup.find_all('table', r = "%d" % i)
52
53 # 依次读取六种乐高套装的数据，并生成数据矩阵
54 def setDataCollect(retX, retY):
55     # 2006年的乐高8288, 部件数目800, 原价49.99
56     scrapePage(retX, retY, './setHtml/lego8288.html', 2006, 800, 49.99)
57     # 2002年的乐高10030, 部件数目3096, 原价269.99
58     scrapePage(retX, retY, './setHtml/lego10030.html', 2002, 3096, 269.99)
59     # 2007年的乐高10179, 部件数目5195, 原价499.99
60     scrapePage(retX, retY, './setHtml/lego10179.html', 2007, 5195, 499.99)
61     # 2007年的乐高10181, 部件数目3428, 原价199.99
62     scrapePage(retX, retY, './setHtml/lego10181.html', 2007, 3428, 199.99)
63     # 2008年的乐高10189, 部件数目5922, 原价299.99
64     scrapePage(retX, retY, './setHtml/lego10189.html', 2008, 5922, 299.99)
65     # 2009年的乐高10196, 部件数目3263, 原价249.99
66     scrapePage(retX, retY, './setHtml/lego10196.html', 2009, 3263, 249.99)
67
68 # 使用sklearn
69 def usesklearn():
70     from sklearn import linear_model
71     reg = linear_model.Ridge(alpha = .5)
72     lgX = []
73     lgY = []
74     setDataCollect(lgX, lgY)
75     reg.fit(lgX, lgY)
76     print('%f*年份+f*部件数量+f*是否为全新+f*原价' % (reg.intercept_, reg.coef_[0], reg.coef_[1], reg.coef_[2], reg.coef_[3])
77
78
79 if __name__ == '__main__':
80     usesklearn()
81
82

```

```

2008 → 5922 → 0 → 299.990000 → 510.000000
2008 → 5922 → 0 → 299.990000 → 423.000000
商品 #6 没有出售
商品 #7 没有出售
2008 → 5922 → 1 → 299.990000 → 599.990000
商品 #9 没有出售
2008 → 5922 → 1 → 299.990000 → 589.990000
2008 → 5922 → 1 → 299.990000 → 569.990000
2008 → 5922 → 1 → 299.990000 → 529.990000
2008 → 5922 → 0 → 299.990000 → 500.000000
2008 → 5922 → 1 → 299.990000 → 549.950000
2008 → 5922 → 0 → 299.990000 → 300.000000
商品 #16 没有出售
2009 → 3263 → 1 → 249.990000 → 380.000000
2009 → 3263 → 1 → 249.990000 → 399.000000
2009 → 3263 → 1 → 249.990000 → 427.990000
2009 → 3263 → 0 → 249.990000 → 360.000000
商品 #5 没有出售
商品 #6 没有出售
2009 → 3263 → 1 → 249.990000 → 399.000000
2009 → 3263 → 1 → 249.990000 → 399.950000
2009 → 3263 → 1 → 249.990000 → 499.990000
商品 #10 没有出售
2009 → 3263 → 0 → 249.990000 → 399.950000
商品 #12 没有出售
2009 → 3263 → 1 → 249.990000 → 331.510000
55235.771966-27.550831*年份-0.026911*部件数量-10.883816*是否为全新+2.576278*原价

```

还是那个部件问题，，，

```

55235.771966-27.550831*年份-0.026911*部件数量-10.883816*是否为全新+2.576278*原价

```

8、sklearn.linear_model.Ridge

sklearn.linear_model.Ridge是一个很好的模型，决策树算法就是通过它实现的，详细的看这个博客——[sklearn.linear_model.Ridge\(\)函数解析（最清晰的解释）](#)

9、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，前者预测连续型变量,而后者预测离散型变量。回归是统计学中最有力的工具之一。在回归方程里，求得特征对应的最佳回归系数的方法是最小化误差的平方和。给定输入矩阵 X ，如果 $X^T X$ 的逆存在并可以求得的话，回归法都可以直接使用。数据集上计算出的回归方程并不一定意味着它是最佳的，可以使用预测值 y_{Hat} 和原始值 y 的相关性来度量回归方程的好坏。

当数据的样本数比特征数还少时候，矩阵 $X^T X$ 的逆不能直接计算。即便当样本数比特征数多时， $X^T X$ 的逆仍有可能无法直接计算，这是因为特征有可能高度相关。这时可以考虑使用岭回归，因为当 $X^T X$ 的逆不能计算时，它仍保证能求得回归参数。

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来求得近似结果。缩减法还可以看做是对一个模型增加偏差的同时减少方差。偏差方差折中是一个重要的概念，可以帮助我们理解现有模型并做出改进，从而得到更好的模型。

本章介绍的方法很有用。但有些时候数据间的关系可能会更加复杂，如预测值与特征之间是非线性关系，这种情况下使用线性的模型就难以拟合。下一章将介绍几种使用树结构来预测数据的方法。