



**QG工作室数据挖掘组中期考核**

**题 目\_\_\_实现机器学习算法\_\_**

**学 院\_\_\_\_自动化学院 \_\_\_\_\_\_\_**

**专 业\_数据科学与大数据技术\_**

**年级班别\_\_\_18届 2班\_\_\_\_\_\_\_\_\_\_**

**学 号\_\_\_\_3118001552\_\_\_\_\_\_\_\_**

**学生姓名\_\_\_\_李炜乐\_\_\_\_\_\_\_\_\_\_\_\_**

**2019年04月18日**

# K-means算法

* 1. 考核要求

1. 基本任务：

内容一：理解k-means算法的思想。

内容二：使用Python实现k-means算法。

内容三：使用UCI上面的Iris数据集进行算法测试。

内容四：对参数k进行调整，记录结果。

1. 进阶任务：

实现二分K-means代码并进行测试，与原有作比较。

1. **数据集**：
2. 网址：<http://archive.ics.uci.edu/ml/index.php>
3. 内容：Iris数据集
4. **数据处理分析工具：**

Pycharm(python3.7)

1. **数据集分析：**
2. 数据集介绍：

Iris.data数据集主要有如下：

|  |  |
| --- | --- |
| sl | 花萼长度 |
| Sw | 花萼宽度 |
| Pl | 花瓣长度 |
| Pw | 花瓣宽度 |
| variety | 花的品种 |

2）数据特征相关性分析：

采用Person相关系数对四个特征值进行两两之间的相关性分析：分析结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | sl | sw | pl | pw |
| sl | 1.000000 | -0.103784 | 0.871283 | 0.816971 |
| sw | -0.103784 | 1.000000 | -0.415218 | -0.350733 |
| pl | 0.871283 | -0.415218 | 1.000000 | 0.962314 |
| pw | 0.816971 | -0.350733 | 0.962314 | 1.000000 |

从表格可以看出，四个特征值之间的相关度有些存在着很强的线性相关，有些存在着很弱的线性相关。

3）数据特征画图：

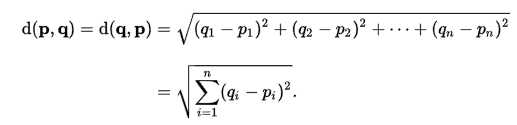
使用pythonmatplotlib库对数据每两两个特征之间进行绘图。

1. **基础任务的实现：**
2. 算法思路：

采用k-means聚类算法，此算法不需要标签变量，需要通过四个特征变量将Iris进行聚类。目标：通过Iris的四个特征值进行聚类，得到每个聚类中的质心，并把聚类结果写入文件中。

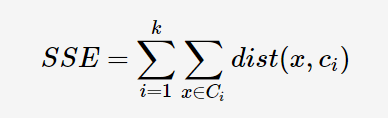
1. 算法原理：

采用欧式公式：



Ps:其中p为n维空间上的点，q为另一个n维向量空间的点。

才用欧式公式计算样本点到质心的距离。通过距离平方和得到我们的损失函数（目标函数）SSE：



误差平方和（SSE）：表示样本中的每个样本点到质心的距离的平方和，最优解为：聚类结果应使SSE达到最小值。

用通俗易懂的话讲：就是每个样本中有一个质心，当样本中的点到质心的距离平方和最小的时候，该质心为我们所求的质心。

1. 算法步骤：

1, 选择合适k值，从数据集可以得出k=3

2, 随机生成k个质点作为起始质点

3, 将数据集中的每一个点分配到每一个簇，即每一个点找到距离近 的质心作为所对应的簇

1，对每一个簇，计算簇中所有点的均值将其作为新的质心

2, 重复cd两个过程直到分配的所有点不再改变

1. 算法的相关函数实现：

1, loadDataSet()： 读入数据，得到四个特征变量

2, distEclud(): 欧式距离公式

3, randCent(): 生成随机k个质心

4, Kmeans(): 主要算法，得到局部最优质心，和分类结果

5，chooseK(): 画出肘部图

6．writeTxt(): 写入文件

1. 算法实现结果：

1.得到局部最优质心(存放于consequeue02.txt文件中)：

k-means的聚类后的质心

第1个质心为：[6.63618347 4.02746118 5.75781118 0.71860106]

第2个质心为：[7.35676933 2.29807678 1.78559103 1.98393566]

第3个质心为：[4.43143218 3.70293815 4.8214567 1.30253784]

2.得到分类结果和与质心的距离平方和(存放于consequeue02.txt文件中)：

kmeans的分类结果

第1个属性被归类为：2 距离为：1.4976817461035465

第2个属性被归类为：2 距离为：1.5922110740206163

第3个属性被归类为：2 距离为：1.9061768368579015

第4个属性被归类为：2 距离为：1.3967556703684567

第5个属性被归类为：2 距离为：1.429697059707002

第6个属性被归类为：2 距离为：1.5978449919959896

第7个属性被归类为：2 距离为：1.3980399630110116

… …

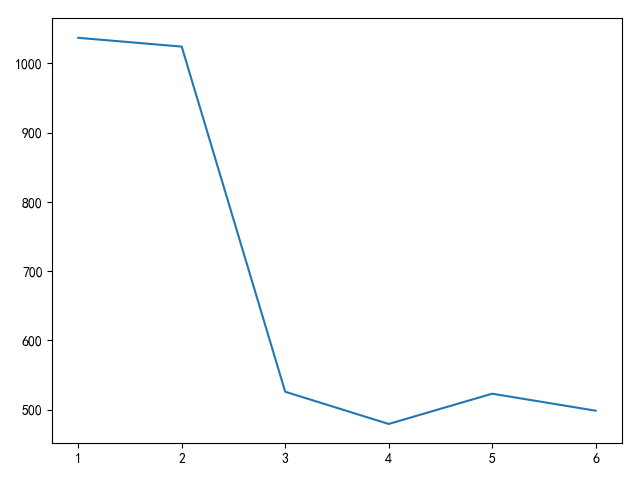
6) 算法存在的不足：

1, K值的取值，簇的数目是用户预先定义的参数，如何选择正确的k是这个算法需要改进的地方

2, K-均值算法收敛到局部最小值，而不是全局最小值。原因是由于随机初始化质心导致收敛效果比较差。

1. 对算法的改进：

1，采用肘部图进行对k值得选择。原理：让k从1取值到7，计算其SSE，利用绘图很明显的看出当k取一定值的时候，SSE函数下降最快，而后下降最小。（函数：chooseK()）:



缺陷：可能由于数据原因，当k取值为4的时候，分类完后数据只有3个类别，导致了k取3，4，5时候SSE并不一定会递减。这个问题改了很多种方法，都不能够解决。

2，利用二分k-means对数据进行聚类：

会在进阶任务进行讲解。

**6. 进阶任务的实现：**

1) 算法思路：

二分k-means算法，此算法不需要标签变量，在k-means算法的基础上需要通过四个特征变量将Iris进行聚类。目标：通过Iris的四个特征值进行聚类，得到每个聚类中的质心，并把聚类结果写入文件中。

1. 算法原理基础：

在原理上跟二分k-means上差不多相同。

1. 算法步骤：

1，把整个数据集看成一个簇，计算质心

2，将这个簇分成两个簇

3，选择满足条件的可以分解的簇，选择条件为簇元素的个数和SSE大小4，使用k-mean算法将可分裂的簇分成两个簇

5，重复（2）（3）步，直到满足k值

1. 算法相关函数的实现：

1, loadDataSet()： 读入数据，得到四个特征变量

2, distEclud(): 欧式距离公式

3, randCent(): 生成随机k个质心

4, Kmeans(): k-means函数

5, chooseK(): 画出肘部图

6, biKmeans():二分k-means的主函数，主要算法

7, writeTxt(): 写入文件

5) 算法实现的结果

1.得到最优质心（存放于consequeue01.data中）：

二分k-means的聚类后的质心

第1个质心为：[[5.66634158 2.32713105 2.34596829 1.23555396]]

第2个质心为：[[5.79714303 2.52272632 6.09696109 2.21276629]]

第3个质心为：[[7.13406605 2.44192342 6.06867267 1.65025568]]

2.得到分类结果（存放于consequeue02.data中）：

聚类结果:

第1个属性被归类为：0 距离为：3.0072600351369587

第2个属性被归类为：0 距离为：3.862137902635603

第3个属性被归类为：0 距离为：3.5224451121278646

第4个属性被归类为：0 距离为：4.031434453899792

… ….

1. **kNN算法**
   1. **考核要求**
2. 基本任务：

内容一：理解KNN算法的思想。

内容二：使用Python实现KNN算法 。

内容三：使用UCI上面的Iris数据集进行算法测试。

内容四：记录测试结果

1. 进阶任务：

自行优化KNN算法并进行代码测试。

1. **数据集**：
2. 网址：<http://archive.ics.uci.edu/ml/index.php>
3. 内容：Iris数据集
4. **数据处理分析工具：**

Pycharm(python3.7)

1. **数据集分析：**

数据集的分析跟上面的k-means的大多一样。

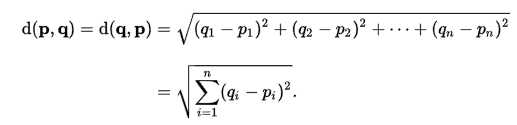
Iris数据集中的花的品种是有规律的，在测试数据集的时候，会对正确率有很大影响。所以在处理数据的时候，需要对数据进行打乱操作

1. **基础任务的实现：**
2. 算法思路：

采用k-近邻算法，将四个连续型变量作为特征值，将标签变量作为标签值。目标：一，将数据集分成训练集和测试集两部分，用测试集的测试结果和真实结果进行匹配，从而得到kNN算法的错误率。二，实现kNN分类器，进行用户输入四个花的类别从而得到花的品种。

1. 算法原理：

采用欧式距离公式：



Ps:其中p为n维空间上的点，q为另一个n维向量空间的点。

通过欧式距离公式，我们可以算出在n维平面上两点之间的距离。在kNN算法中，采用欧式距离公式计算出测试集中的一个点到训练集中的所有点的距离，找到其中k个距离最短的点，而测试点的标签最大概率等于k个点的标签中频率最高的标签。

1. 算法实现步骤：
2. 计算出每个点到测试集的点
3. 进行排序取索引值
4. 定义一个记录标签类别次数的字典
5. 取出前面k个元素
6. 找到出现次数最多的类别
7. 算法的相关函数：
8. loadDataSet(): 得到数据
9. autoNorm(): 对数据进行归一化
10. classify0(): kNN分类器，得到分类结果
11. classifyPerson() : 使用分类器，用户输入花的四个特征，输出花的品种
12. datingClassTest(): 对分类器进行测试
13. adjustK(): 获得到最优参数k
14. 算法的不足：

kNN算法中，需要一个参数k，然而kNN分类器中，k这个参数需要我们人工进行输入，然后找到最优的一个参数，比较麻烦。

1. 对算法的优化：

在算法中加入adjustK()这个函数，从而得到最优的k，adjustK()的原理很简单，让k从1取到10，使用分类器计算出每一个的正确率，找出最大的正确率所对应的k，返回到数据中。

1. 算法的实现结果：
2. 测试集的错误率：

错误率： 0% ~ 2.272727%

1. 分类器的使用结果：

请输入花萼长度：1.1

请输入花萼宽度：1.2

请输入花瓣长度：1.3

请输入花瓣宽度：1.4

该花的类型可能为setosa

1. **Apriori算法**
2. **考核要求：**

内容一：理解Apriori算法的思想。

内容二：使用Python实现Apriori算法 。

内容三：使用UCI上面的mushroom数据集进行算法测试。

内容四：修改支持度和置信度，进行多次测试。

内容五：记录测试结果

1. **数据集：**

1) 网址：<http://archive.ics.uci.edu/ml/index.php>

2) 内容：mushroom数据集

1. **数据处理分析工具：**

Pycharm(python3.7)

1. **数据集分析：**
2. 数据集内容：

Agaricus-lepiota.data数据很庞大，例如：

p,x,s,n,t,p,f,c,n,k,e,e,…,w,w,p,w,o,p,k,s,u

e,x,s,y,t,a,f,c,b,k,e,c,…,s,w,w,p,w,o,p,n,n,

e,b,s,w,t,l,f,c,b,n,e,c,…,w,w,p,w,o,p,n,n,m

p,x,y,w,t,p,f,c,n,n,e,e,…,w,w,p,w,o,p,k,s,u

e,x,s,g,f,n,f,w,b,k,t,e,…,w,w,p,w,o,e,n,a,g

e,x,y,y,t,a,f,c,b,n,e,c,…,w,w,p,w,o,p,k,n,g

跟上面一样的字母和逗号组成。一共8123行，23列数据

1. 数据介绍：

|  |  |
| --- | --- |
| classes | 有毒为p，无毒为e |
| cap-shape | 钟型为b，conical锥为p等 |
| cap-color | 纤维为b，等 |
| … | … |

这里数据集里面的信息太多了，简单介绍一部分。

1. 数据处理：

由于数据里面不同列之间的重复出现的字母会对Apriori算法产生不小的影响。所以需要将数据集中的字母转换成不同的字符。在这里我将数据转换成数字。比如:第一列的无毒为1，有毒为2，钟型为3，依次类推，到最后一列。处理后的数据输出如下：

Classes cap-shape … population habitat

0 2 5 … 109 116

1 1 5 … 108 112

2 1 3 … 108 114

3 2 5 … 109 116

这样就生成了不同的特征变量，就不会造成两两特征之间的相互影响了。不过转换过程太过于麻烦和繁琐。不过这是我想到的比较暴力的方法。

1. **算法的实现：**
2. 算法介绍：

Apriori算法是经典的挖掘频繁项集和关联规则的数据挖掘算法。

* 支持度(关联规则的支持度定义如下)：



其中A和B交集为空，P（A U B）表示事务包含集合A和B的并集的概率

* 置信度(关联规则的置信度定义如下)：



1. 算法(大方面)的实现步骤：
2. 找出所有频繁项集
3. 由频繁项集产生强关联规则
4. 挖掘频繁项集的步骤：
5. 先搜索出候选1项集及对应的支持度，剪枝去掉低于最小支持度的项集，得到频繁1项集。
6. 搜索出候选2项集及对应的支持度，再剪枝去掉低于最小支持度的项集，得到频繁2项集
7. 以此类推，一直迭代下取，直到频繁k+1项集位置。对应的k项集即为算法的输出结果。
8. 由频繁项集产生关联规则的步骤：
9. 对于每个频繁项集，产生该项集的所有非空子集（这些非空子集一定是频繁项集）
10. 对于每一个非空子集，如果confidence(A=>B)>=confmin（最小置信度）则输出A=>B。

称为强关联规则。

1. 算法的相关函数：
2. loadDataSet()：读入数据，并将数据转换成数字
3. createC1(): 构建初始候选项集的列表，即所有候选集只包含一个元素
4. scanD():计算Ck中项集在数据集的支持度，返回满足最小支持度的集合和所有支持度信息的字典
5. aprioriGen():由初始候选集的集合生成心得候选集，k参数表示生成新项集中所含有的元素的个数
6. apriori(): Apriori算法重要函数，重要目的是返回所有满足条件的频繁项集的列表和所有选项集的支持度信息。
7. generateRules():根据频繁项集和最小可信度生成规则
8. calcConf():计算规则的可信度，返回满足最小可信度的规则
9. rulesFromConseq():利用递归合并频繁项集和可以出现规则右部的元素
10. 算法的不足：

Apriori由于因为是逐层搜索，所以耗时特别大。时间复杂度比较大。

1. 算法结果：
2. 满足最小支持度为0.7时生成的频繁项集：

[[frozenset({'97'}),frozenset({'94'}),frozenset({'90'}), frozenset({'38'}), frozenset({'36'}), frozenset({'104'})]…]],

1. 满足最小可信度为0.7时生成的关联规则：

frozenset({'104'}) --> frozenset({'36'}) conf: 0.9650698602794411

frozenset({'36'}) --> frozenset({'104'}) conf: 0.733131159969674

frozenset({'36'}) --> frozenset({'38'}) conf: 0.834217841799343

frozenset({'38'}) --> frozenset({'36'}) conf: 0.9691720493247211

frozenset({'104'}) --> frozenset({'90'}) conf: 1.0

…

1. 总耗时：

Apriori算法生成频繁项集：0.9065756797790527秒

生成关联规则后耗时：1.0472471714019775秒

1. **线性回归算法**
2. **考核要求：**

内容一：理解LinearRegression算法的思想。

内容二：使用Python实现LinearRegression算法 。

内容三：使用housing数据集进行算法测试。

内容四：记录测试结果

1. **数据集：**

数据链接：https://pan.baidu.com/s/1OvS48Qn1f69hraWpdC9mgg 提取码：hw4r

数据类型：房屋数据集

1. **数据处理分析工具：**

pycharm（python3.7）

1. **数据分析：**
2. 数据集介绍：

房屋数据集主要由13个可能影响房屋的因素和1个自住房的平均房价组成。

1. 数据集的内容：

|  |  |
| --- | --- |
| CRIM | 城镇人均犯罪率。 |
| ZN | 住宅用地超过 25000 sq.ft. 的比例。 |
| INDUS | 城镇非零售商用土地的比例。 |
| CHAS | 查理斯河空变量（如果边界是河流，则为1；否则为0）。 |
| NOX | 一氧化氮浓度。 |
| RM | 住宅平均房间数。 |
| AGE | 1940 年之前建成的自用房屋比例。 |
| DIS | 到波士顿五个中心区域的加权距离。 |
| RAD | 辐射性公路的接近指数。 |
| TAX | 每 10000 美元的全值财产税率 |
| PTRATIO | 城镇师生比例。 |
| B:1000(Bk-0.63)^2 | 其中 Bk 指代城镇中黑人的比例 |
| LSTAT | 人口中地位低下者的比例。 |
| MEDV | 自住房的平均房价，以千美元计。 |

1. 对数据处理过程：
2. 对数据进行相关性分析：

**先对计算每个自变量对MEDV的相关性（person系数）：**

CRIM 和MEDV的相关系数为 -0.3882494148489865

ZN 和MEDV的相关系数为 0.36039332455431233

INDUS 和MEDV的相关系数为 -0.48412584197626285

CHAS 和MEDV的相关系数为 0.17536370524182765

NOX 和MEDV的相关系数为 -0.42729475659279587

RM 和MEDV的相关系数为 0.6953645190298522

AGE 和MEDV的相关系数为 -0.3769315073572471

DIS 和MEDV的相关系数为 0.24989557064633722

RAD 和MEDV的相关系数为 -0.3816903035349454

TAX 和MEDV的相关系数为 -0.4685431589928918

PTRATIO和MEDV的相关系数为 -0.508410806363853

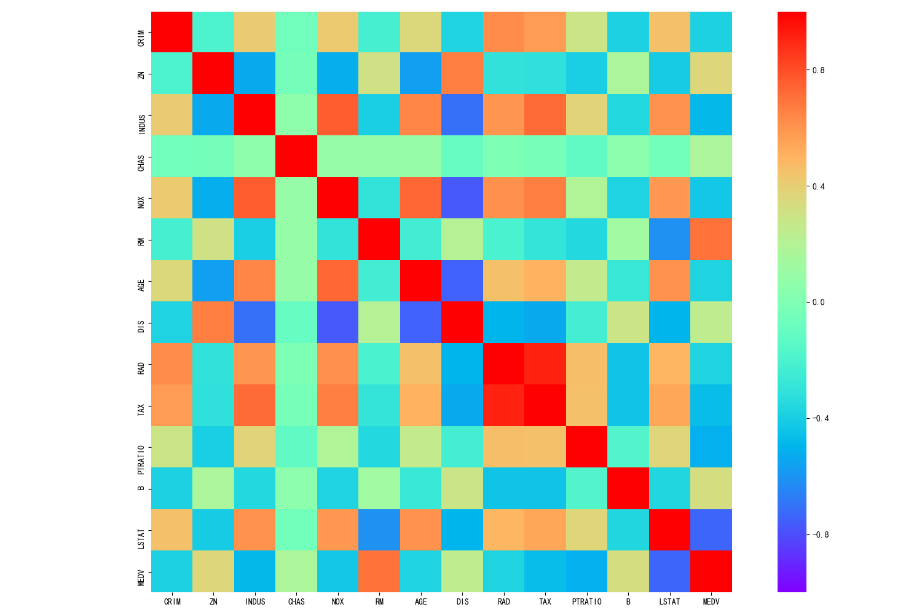
B 和MEDV的相关系数为 0.3333939887967591

LSTAT 和MEDV的相关系数为 -0.7381872293702556

MEDV 和MEDV的相关系数为 1.0

从输出结果可以看出来，其中CHAS，DIS等影响因素对MEDV的的相关性特别小。这样看比较难看。

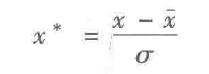
**画出每个变量之间两两的相关性热力图：**



颜色越浅，相关性越低，颜色越深，相关性越高。只看最后一列的话，可以明显看得出来，CHAS和DIS分别对MEDV的相关性很低。从全部图来看，DIS和INDUS,NOX,AGE的相关性很高。这个可能会影响我们对做加权回归的时候的影响。

在这里，我们提取前相关性前九的影响因素进行分析。

1. 对数据进行规范化处理：

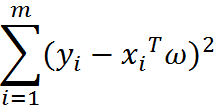


1. 对数据分类：

由于数据一共由505条数据，我将数据分成两份，前252项作为得到w系数的训练集，后252项作为测试集计算误差平方和。因为：防止因为过拟合而影响算法的优劣性。

1. **简单线性回归的实现：**
2. 算法原理：

* 目标函数（损失函数）：



前一部分为y的真实值，后一部分为预测值，其中w为我们需要求的参数。我们需要求出最优的参数w使得目标函数损失值最小。

* 利用矩阵表示的平方误差对w进行求导：



最小二乘法解法，此时候的w值为估计值。

1. 算法实现步骤：
2. 利用矩阵求逆法求出估计的w值
3. 利用现有的x值求出预测的y值
4. 与真实值相减求平方和（算出损失函数）
5. 与算法相关的函数：
6. loadDataSet():得到x向量和y向量
7. standRegres():计算回归系数
8. rssError():计算误差平方和
9. testAlgrim():计算预测值和输出误差平方和
10. 算法的结果：
11. 得到回归系数：

[[ 7.61613814e+00]

[ 3.38949843e+00]

[ 5.30985002e-03]

[ 4.65715808e+00]

[-2.21759459e+01]

[-3.31673095e-01]

[-3.76525358e+02]

[-3.18723195e+00]

[-6.81656330e+00]]

2. 得到误差大小：

简单的线性回归误差大小: 15140.760658810585

1. 算法的不足：

普通线性回归可能存在欠拟合的现象，因为它求的是具有小均方误差的无偏估计。所以需要引入一些偏差，从而降低预测的均方误差。

1. 算法改进：

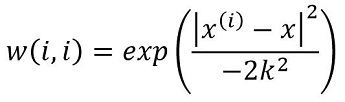
使用局部加权线性回归。

1. **局部线性回归的实现：**
2. 算法原理：

* 我们给待预测点附近的每一个点赋予一定的权重。回归系数计算形式如下：



这里通常使用“核”来对附近的点赋予权重，核的类型通常为高斯核，高斯核的权重如下：



1. 算法实现步骤：

1. 利用局部加权求出估计的w值

2. 利用现有的x值求出预测的y值

3. 与真实值相减求平方和（算出损失函数）

3） 算法相关的函数：

1. loadDataSet():得到x向量和y向量

2. lwlr():计算回归系数

3. rssError():计算误差平方和

4. testAlgrim():计算预测值和输出误差平方和

4） 算法实现结果：

可能由于数据的原因，在使用加权回归的时候，会出现矩阵为奇异矩阵不能求逆的情况。使用pca降维后还是会出现这样的情况。有可能矩阵X不是一个满秩的矩阵，非满秩矩阵求逆会出现问题。

1. 算法改进：

使用岭回归。

1. **岭回归的实现：**
2. 算法的原理：

* 岭回归即L2正则线性回归。在一般的线性回归最小均方误差的基础上加上一个参数w的L2范数罚项。简单来说，岭回归就是在普通线性回归的基础上引入单位矩阵。



目的：使矩阵非奇异。进而可以求逆。

* 优点：通过引入λ来限制了所有的w之和，通过引入该惩罚项，可以减少不必要的参数。

1. 算法实现的步骤：

1. 利用岭回归求出估计的w值

2. 利用现有的x值求出预测的y值

3. 与真实值相减求平方和（算出损失函数）

3) 与算法相关的函数：

1. loadDataSet():加载数据

2. ridgeRegres():岭回归求出w

3. rssError():计算误差平方和

4. testAlgrim():计算预测值和输出误差平方和

4） 算法结果：

1.得到回归系数：

[[-2.58485515]

[ 1.1507946 ]

[-0.05030816]

[ 4.60149814]

[-9.59753609]

[-7.44674707]

[-2.14895974]

[-3.47745037]

[-6.91527309]]

2.得到误差平方和：

不用交叉验证的岭回归误差大小： 14007.510404118051

5） 结论：

从数据可以看出，使用岭回归与普通的线性回归相比，误差平方有一些下降。

1. 进一步优化：

采用向前逐步线性回归的方法

1. **向前逐步线性回归实现:**
2. 算法原理：

属于一种贪心算法，每一步都尽可能减少误差，每次微调各个回归系数，返回误差最小的回归系数。

1. 算法使用函数：

1. loadDataSet():加载数据

2. stageWise ():求出迭代次数中最优的参数w

3. rssError():计算误差平方和

4. testAlgrim():计算预测值和输出误差平方和

3） 算法注意点：

如果微调系数过低或者迭代次数过高都会出现过拟合的情况

1. 算法结果：
2. 当步长为0.01，迭代次数为100次的时候，误差为：

步长为0.01,迭代次数为100次的逐步线性回归的误差大小： 23582.138787724187

1. 当步长为0.2， 迭代次数为20次的时候，误差为：

步长为0.01， 迭代次数为20次的逐步线性回归的误差大小： 16374.478517887446

1. 结论：

步长多的话会导致函数过拟合，导致误差反而更大。

1. 外加：因为之前对数据降维，尝试解决局部加权问题，可是并没有成功，顺便将降维后的数据做其他回归。得到结果：

降维后简单的线性回归误差大小: 22234.65721761735

降维后经过10次交叉验证的岭回归误差大小： 23012.255621162396

降维后不用交叉验证的岭回归误差大小：22251.41740457761

误差反而更大了。这里还没有时间取研究。

1. **结束语**

以上便是中期考核期间完成的任务。以上可能会有错误或疏忽点，望师兄指点。