



装

订

线

本科生毕业论文（设计）

题目**：二手车交易价格分析及异常检测**

学 院 数学与信息科学学院

学科门类 工学

专 业 软件工程

学 号 20161004086

姓 名 艾乐

指导教师 张辉

2020年5月

摘 要

二手车交易价格分析与异常的检测，在分析与检测的过程中，主要是利用了两个常用的方法；一个是统计学的方式，利用二手车的一些数据，如价格、里程、年份、自动挡手动挡等参考因素，来构建我们的一个多维的线性回归模型，但是一维线性回归方程，主要是通过数字化的方式实现的，而对于那些非数字的如手动挡自动挡、车发动机等因素，我们需要对其做一些特殊的处理，这里我们对其进行one-hot编码的处理方式，构建完成我们one-hot编码的格式之后，我们就需要对纯数字化的参考因素做一个线性回归模型，然后利用线性回归方程，得到我们结果，再与我们每个真实价格做对比，相差较大的，我们就可以认为该价格是异常价格；第二种方式就是我们的，局部离群点检测方法，该方法有基于多种形式的，我们使用一种基于距离的离群检测，而LOF(Local Outlier Factor)，局部异常因子，就是一种基于距离的异常检测方法，我们通过获得每一个样本的LOF值，通过比较其与特定值值的大小，来判断该样本的值，是正常的价格，还是异常的价格

关键字：多维线性回归模型；one-hot编码；局部离群点；局部异常因子

Abstract

Used car transaction price analysis and anomaly detection. In the analysis and detection process, two commonly used methods are used; one is a statistical method, using some data of used cars, such as price, mileage, year, automatic Reference factors such as manual transmission are used to build a multi-dimensional linear regression model, but the one-dimensional linear regression equation is mainly realized by digital methods. , We need to do some special processing on it, here we do one-hot encoding processing method, after building our one-hot encoding format, we need to do a linear regression model of pure digital reference factors, Then use the linear regression equation to get our results, and then compare it with each of our real prices. If the difference is large, we can think that the price is an abnormal price; the second way is ours, the local outlier detection method, This method is based on multiple forms, we use a distance-based outlier detection, and LOF (L ocal Outlier Factor), a local anomaly factor, is a distance-based anomaly detection method. We obtain the LOF value of each sample and compare the value with the "1" value to determine the value of the sample. It is normal Price, or abnormal price.

Key words: Multidimensional linear regression model;generator; one-hot encoding; Local outliers; Local Outlier Factor

目录

[1、引言 1](#_Toc11170)

[2、one-hot编码 3](#_Toc21130)

[2.1 概念 3](#_Toc20281)

[2.2 处理方式 3](#_Toc20136)

[3、线性回归模型 6](#_Toc21206)

[3.1一维线性回归方程 6](#_Toc22025)

[3.2多维线性回归方程 6](#_Toc6061)

[4 多元线性回归的推导过程以及求解 8](#_Toc16828)

[4.1向量表示形式 8](#_Toc17549)

[4.2最大似然估计 8](#_Toc3916)

[4.3 概率密度函数 9](#_Toc29421)

[4.4 多元线性模型回归函数解析解表达 11](#_Toc1056)

[4.5 多元线性函数系数求解 12](#_Toc17717)

[4.6 线性回归模型离群值检测 14](#_Toc26903)

[5 局部异常因子（LOF）算法 15](#_Toc14462)

[5.1 LOF算法简介 15](#_Toc2005)

[5.2 LOF算法实现过程 16](#_Toc21788)

[5.3 LOF算法的实验结果的分析 18](#_Toc30086)

[参考文献 22](#_Toc18829)

[致 谢 23](#_Toc1265)

1、引言

近年来随着人们生活水平的提高，人们的出行方式也发生了翻天覆地的变化。又以前的步行、自动车，慢慢的变成了电动车、摩托车；而现如今也越来越多的人选择小轿车的出行方式了，且对小轿车的需求量日益增大；但是也有那么一部分的人，选择了相对较便宜的二手车，虽然二手车的价格相对较便宜，但是二手车与崭新出厂的车的最大的区别，就是它的价格定位难以掌握，并且影响二手车价格的因素较多，比如车的里程、上市年份、车发动机、以及挡位方式是手动挡还是自动挡，二手车的价格都将是围绕这些因素来产生偏差的，但是也不乏有个人的人为因素，所以这个时候，就会出现我们所谓的异常价格，价格偏高的，或者说偏低的这些异常值，我们需要对这些价格做一定必要的整理，来做一些价格的分析与异常值的检测；

异常价格，我们称之为离群点，通俗来说，就是离其他的数据点比较远的数据点，这样的价格，或者说点，会严重的影响我们后续对价格的分析结果，甚至产生有误导的错误的分析结果，对于二手车来说，其有价格（price）、里程（mileage）、上市年份（year）、档次（trim）、引擎缸数（engine）、换挡方式（transmission）这些影响因素来判断该车的数据是否为异常的数据；而我们的异常值检测方法也有很多种，从统计学的角度出发的话，我们可以利用线性回归模型【1~3】的方式，在上述众多的影响因素中，不乏有一些非数字型的数据，但是在做线性回归模型方程的时候，我们只能出数字型的数据下手，但那些非数字型的因素又不能忽略，所以我们可以采用一种叫做one-hot编码【4~5】方式对非数字型的因素进行数字化，这种编码方式主要是采用N位状态寄存器进来对N个状态进行编码，每个每个状态都有他独立的寄存器位，并且任意位置有且只有一位有效，他们以分类变量作为二进制向量【6】表示，首先将分类值映射到整数值，然后每个整数被表示为二进制向量，除了整数的索引之外，其他的都是零值，而它被标记为1；这样，我们就将非数字化的因素转化为数字化了，由于影响因素过多，所以这是一个多维的线性回归模型方程，当我们得出一个关于价格为因变量，而其他影响因素为自变量的线性回归方程后，我们通过方程的方式重新计算一个价格，然后与我们数据中的价格作为一个对比，当二者相差较大时，我们有理由的认为，该价格是异常值，即该数据样本是异常值；而另一种方法是利用一种局部异常因子（LOF）【7~12】这种概念来展开的，异常检测的实质无非就是寻找观测值和参考值之间有意义的偏差，因为某些原有，我们的样本数据会有一些异常的值，我们不可能对其做一个人工检测，所以利用算法解决，而离群点检测是异常检测中最常用的方法之一，离群点检测的主要目的就是为了检测那些与正常数据差别较大的异常数据；而我们的算法也有分类。基于统计的离群检测、基于聚类的离群检测、基于分类的离群检测、基于距离的离群检测、基于密度的离群检测和基于信息熵的离群检测方法，而LOF则是基于距离的离群检测方法之一。我们通过点与其旁边局部的点通过他们的第k距离、距离领域、可达距离等数据，来得出我们的局部可达密度；从而获取我们的局部异常因子，通过该因子与某个常量特定值的大小来判断该点是不是异常点；总结来说，就是比较每个点和其领域点的密度来判断该点是不是异常点，如果点的密度月底，越可能被认为是异常点，而所谓的密度，都是通过距离来计算的，点之间的距离越远，密度越低；距离越近，密度越高，且这些数据都是通过第k领域来计算，即该点的旁边局部点而得出，而不是全局计算，所以称之为“局部”异常因子；所以即使数据都是紧凑的那种，还是散漫的那种；我们通过判断密度来判断，二者都可以被认为是正常的样本数据；

2、one-hot编码

2.1 概念

one-hot编码又被称为有效编码，主要是采用N位寄存器来对N个状态进行编码，每个状态都有独特的寄存器位且在任意时候有且只有一位有效；

在实际的应用任务中，有些样本的特征有时候并不是总是连续值，而是一种分类值，如二手车的样本值中，引擎缸数、档次等信息，就是一种分类值，这个时候我们就需要对其特征进行数字化；拿引擎缸数和档次为例子如下：

* 引擎缸数（engine）：[4 Cyl；4杠,6Cyl：6杠]；
* 档次（trim）：[exl：高配款且带皮革内饰，ex：高配款，lx：低配款]

我们就需要对这种分类值的特征进行one-hot编码数字化

2.2 处理方式

原始数据如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Price | Mileage | Year | Trim | Engine | transmission |
| 0 | 14995 | 67692 | 2006 | ex | 4Cyl | Manual |
| 1 | 11988 | 72728 | 2006 | ex | 4Cyl | Manual |
| 2 | 11999 | 80313 | 2006 | lx | 4Cyl | Automatic |
| 3 | 12995 | 86096 | 2006 | lx | 4Cyl | Automatic |
| 4 | 11333 | 79607 | 2006 | lx | 4Cyl | Automatic |
| 5 | 10067 | 96966 | 2006 | lx | 4Cyl | Automatic |

Java one-hot编码代码：

**private** List<OneHotSample> readFile2(String path) **throws** Exception {  
 InputStream is = **new** FileInputStream(path);  
 List<OneHotSample> list = **new** ArrayList<>();  
 **try**{  
 *//换成你的文件名* BufferedReader reader = **new** BufferedReader(**new** FileReader(path));  
 reader.readLine();*//第一行信息，为标题信息，不用* String line = **null**;  
 **while**((line=reader.readLine())!=**null**){  
 *//CSV格式文件为逗号分隔符文件，这里根据逗号切分* String item[] = line.split(**","**);  
 OneHotSample sample = **new** OneHotSample();  
 sample.setPrice(Integer.*valueOf*(item[0]));  
 sample.setMileage(Integer.*valueOf*(item[1]));  
 sample.setYear(Integer.*valueOf*(item[2]));  
 **switch** (item[3]) {  
 **case "ex"** :  
 sample.setTrimEx(1);  
 **break**;  
 **case "lx"**:  
 sample.setTrimLx(1);  
 **break**;  
 **default**:  
 sample.setTrimExl(1);  
 }  
 **switch** (item[4]) {  
 **case "4 Cyl"** :  
 sample.setEngine4Cyl(1);  
 **break**;  
 **default**:  
 sample.setEngine6Cyl(1);  
 }  
 **switch** (item[5]) {  
 **case "Automatic"** :  
 sample.setTransmissionAutomatic(1);  
 **break**;  
 **default**:  
 sample.setTransmissionManual(1);  
 }  
 list.add(sample);  
 }  
 } **catch** (Exception e){  
 System.***out***.println(**"文件转换出错:"** + e);  
 } **finally** {  
 **if** (is != **null**) {  
 **try** {  
 is.close();  
 } **catch** (Exception e){  
 System.***out***.println(**"输入流关闭异常"**);  
 }  
 }  
 }  
 **return** list;  
}

经过one-hot编码后（由于空间问题，价格price列未展示）：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Mileage | Year | Trim-ex | Trim-lx | Trim-exl | Engine-4Cyl | Engine-6Cyl | Tr-manu | Tr-auto |
| 0 | 67692 | 2006 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 72728 | 2006 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 2 | 80313 | 2006 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 3 | 86096 | 2006 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 4 | 79607 | 2006 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 5 | 96966 | 2006 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

从编码结果来说，对于如trim这种非数字化的特征，我们将其one-hot编码后，就变成了数字化，且在trim的属性中，有且只有一位是有效的，如 1 0 0，0 1 0；这个时候，我们就将影响因素数值化了，后续就可以对其进行构建线性回归模型；

3、线性回归模型

3.1一维线性回归方程

对于表面上的线性，我们给定一组输入值x和输出值y；（x,y），我们假设他们的关系是线性的关系，这时就可以得到关系 y = kx + b，只要我们得到k和b的值，就可以得到关于这个关系的方程，即得到我们的线性。

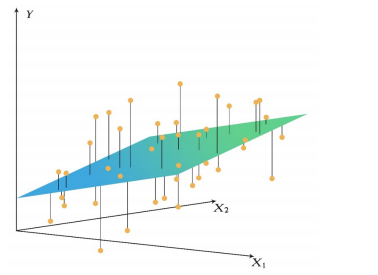


我们所有的（x，y），都是围绕这条线分布的，这时，我们通过输入的自变量x，可以一个值y，将该得到的y和实际值y比较，当二者的差距在我们置信水平范围内（误差范围），我们就有理由的认为该组数据是正常值，否则认为是异常值。

3.2多维线性回归方程

上述一维的线性回归方程，其实是相当于多维线性回归模型中特殊的一种情况，即自变量只有一种的情况；线性回归模型真正的含义是：利用线性函数对一个或多个自变量（x或者x0、x1、x2、x3...xn）和因变量（y）之间的关系进行拟合的模型，即线性回归模型构建成功后，这个模型表现线性函数的形式，所以说一维的，就是特殊的多维线性回归方程；

* 自变量只有一个：y = f(x) = a + bx;
* 自变量多个：y = f(x1,x2,,x3,...,xn) = a + b1x1 + b2x2 + b3x3 + ...+ bnxn;



如果特征属性是一维的，那么他就是一条直线，如果是二维的就是一个平面，三位的就是一个体，以此类推下去。

上述多元线性回归方程中，这些x和y的关系整体的图像还并不可以完全的满足我们任意两点之间的关系的，但是这条直线又是我们综合所有的点里面最合适的用来描述他们共同特征的直线了，毕竟它到我们所有点的距离的和是最小的即是误差最小的所以我们的多元线性回归模型方程的表达式又可以写成：

y = w0x0 + w1x1 + w2x2 + ... + wnxn

由于我们无法确定多元线性回归模型函数中预测到的回归函数是否经过原点，于是我们在多元线性回归模型函数中，就需要保留一项常数项来作为我们的截距，于是就有了新的多元线性回归模型函数：

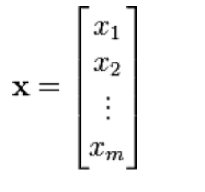
y = w0 + w1x1 + w2x2 + ... + wnxn

当我们没有w0这一项的时候，我们的多元线性回归模型函数就是一条完全由n+1个自变量构成且经过坐标原点的图像的函数，这将导致我们一直用一个经过原点的图像来概括或者说描述一些散列点的分布情况，这显然是不可理的，增大了局限性，这将造成我们预测出的结果的函数准确率将会大幅度的下降。在这里我们默认x0 = 1而不是x2或者其他的x，是因为我们的目标就是为了让函数y = w0 + w1x1 + w2x2 + ... + wnxn是一个包含常数项的线性回归函数模型，无论如何的去选取这个x都是可以的，放在第一个位置，这样便于理解。

4 多元线性回归的推导过程以及求解

4.1向量表示形式

向量就是一个数组，如[1,2,3,4,5]这是一个有五个元素的向量，向量有行向量和列向量之分，行向量就是前面的那种，数字横向排列的：X = [1,2,3,4,5]，而列向量的话，就是数字是竖着排列的，如图：



现在，我们假设有这么一个向量W = [w1,w2,w3,...,wn]为行向量，另一个列向量

X = [x1,x2,x3,...,xn]，行向量和列向量相乘的法则就是对位相乘且相加，结果就会得到一个实数，这就符合我们的预测结果等于y了，所以我们就可以将表达式写成 y = W \* X了。

在这种情况下，多元线性回归模型函数 y = w0 + w1x1 + w2x2 + ... + wnxn，方程就化简成为y = W \* X了，但是如果两个向量中，一个是行向量，一个是列向量的话，很容易就混淆了，所以我们就直接规定我们的W和X都是列向量，这时表达式就可以改写成WT（列向量的转置，即WT，所以WT是行向量）与列向量相乘，最后的最终结果的线性回归模型方程的表达式就变为：

y = WT \* X；但是WT也经常用θT表示

如果W和X都行向量的话， y = W \* XT也是和前面一样的，只是单纯的为了统一表达式，而选择的一种形式而已，二者并没有差别。

4.2最大似然估计

最大似然估计【13】就是最大可能性估计的意思，主要内容为：如果事件A和事件B相互独立，于是我们可以说事件A和事件B同时发生的概率会满足一个公式

P(A,B) = P(A) \* P(B)

P(X)就表示事件X发生的概率；

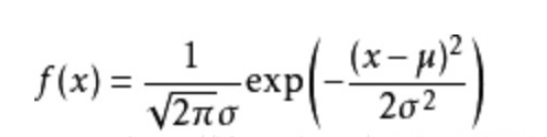
所谓两个独立的事件的意思就是：这两个事件不相关，即A事件发生与否，对B事件的发生与否完全没有影响，即两事件完全不存在关系；这个时候我们就可以称这两个事件是相互独立的；我们使用多元线性回归模型函数的目的就是为了总结一些不相关因素的规律，总结相互独立事件发生的概率也就可以认为是在总结所有相互独立事件同时发生的概率，当所有的事情发生的概率越大时，那么我们预测到的规律就会变得更加的准确。当影响的因素或者说观察到的维度更多的话，则对预测到的维度发生的机率就会越大，也就是说我们的结果更准确，根据最大似然估计表达式：

P(y) = P(x1,x2,x3,..xn.) = P(x1) \* P(x2) \*P(x3) ...P(xn)

当我们所有的事件发生的概率最大时，就可以认为所得函数最符合这些独立事件发生的实际规律，于是我们就把所有样本点的基本分布规律转变成了一个求最大似然估计P(y)的概率,即P(x1,x2,x3...xn) = P(x1) \* P(x2) \* P(x3)...\* P(xn)发生的最大概率。

4.3 概率密度函数

虽然我得到了分布规律的最大似然值，但是在数学当中，其实并没有某种方法来让我们直接求得在何种情况下几个独立事件同时发生的最大的概率，于是我们又引入了概率密度函数【14】；但是在此之前还有一个重要的概念就是：一个随机的变量发生的概率一定会符合正态分布或者说高斯分布，而高斯分布的概率分布密度还是高斯分布，公式：



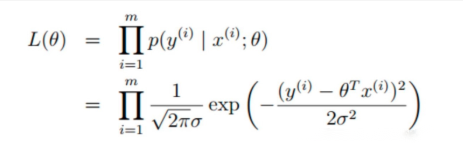
在这个公式中，x就是我们的实际值，μ就是我们的预测值，所以在多元线性回归模型函数中，x就是实际的y，而μ就是我们的θT \* X；前面说到我们需要总结的事件是相互独立的事件，于是这里的每个事件就都是一个随机的事件，或者称他们为随机变量，所以我们最后总结的每个事件的发生概率都将会符合我们的高斯分布。

概率密度函数，就是某个事件发生的概率有多大，当将事件x带入到上述的公式中得到的值越大时，就证明该事件发生的概率越大，但是又值得注意的是，得到的并不是该事件发生的概率，而只是让我们知道该高斯分布公式的值同该事件发生的概率是个正相关而已。

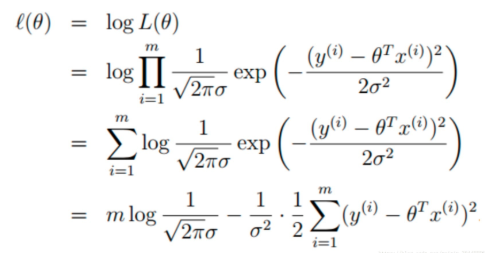
如果将y = θT \* X中的每个自变量x带入到这个公式，将得到如下函数



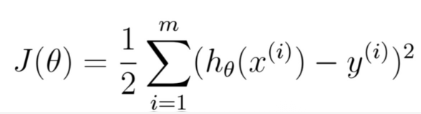
求得所得的独立事件的发生概率最大就是求得左右独立事件概率密度函数结果的乘机的最大值，于是可以得到如下公式：



求L(θ)最大的时候，W的值，则能总结出所有的独立事件符合的规律，需要对上述公式进行求解，因为需要求得的是该函数在什么情况下函数所得到的值最大，而不是求该函数的所有解：



将函数进行一个Log计算，这样将连乘化解成为了连加,这样便于后续的计算，在该公式中，m就是样本的个数，π和σ都是常数，不会影响上述表达式的大小，于是去掉所有的常数项后得到的新的公式如下：



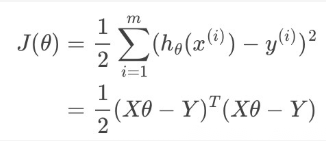
上述得到的公式实质上是一个常数减去减去该公式，所以求概率密度函数的最大值就相当于求这个公式的最大值，而这个公式又是一个数的平方，在数学上，我们称其为最小二乘公式【15】，所以，我们的多元线性模型回归函数的本质就是一个最小二乘。

4.4 多元线性模型回归函数解析解表达

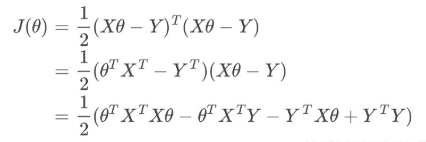
在多线线性回归模型函数中有种解析解求解法，即我们可以通过一个指定的公式得到方程的解，当我们把函数方程的参数带入到公式中，计算公式得到的结果就可以得到方程的解，而不需要一步一步的去化简从而求解；最简单且最熟悉的莫过于学过的一元二次方程的解析解了我们只需要将上述的a、b、代入就可以得到方程的解；所以我们需要得到最小二乘函数的解析解表达方式。

求某个函数方程在它某一点上面的导数，就相当于在该方程函数上，过该点多做切线得到的斜率，而该点的导函数就是切线的函数，如果能够找到函数图像上面切线是零的点的话，就可以得到函数的解，而函数上某点求导就相当于通过该点在图像上做切线，做出来得到的切线就是我们所求导得到的导函数的图像，即切线的函数就是对函数方程求导得到的一个导函数，则我们只要获取到该导函数为0的点，就可以得到该图像的解；于是我们对最小二乘函数求导，然后让导函数的结果等于0；最后得到的结果就是最小二乘函数的解；过程如下：

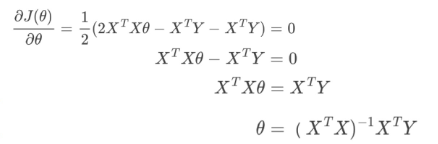
首先对最小二乘做一个变形，变成一个矩阵的表达形式：



展开矩阵函数：



对展开的矩阵J(θ）求导，然后让导数等于0



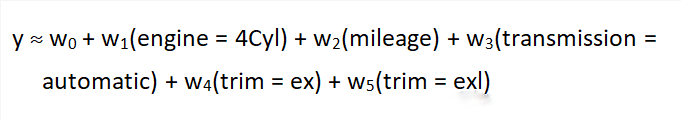
最后我们就求得了解析解的表达式为：



其中θ就是我们线性回归模型方程的系数向量，而X,Y则是样本的影响因素值的向量和样本结果值的向量，即二手车中，X是非价格列，而Y就是样本的价格，当将所有的样本值代入后，就可以得到向量θ的值，即线性回归模型方程的系数向量。

4.5 多元线性函数系数求解

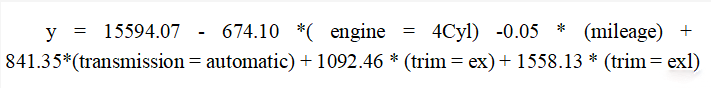
得到多线线性模型函数的解析解的表达式，后续我们只需要将one-hot编码后的样本数据带入即可，但是在样本数据当中，某些特殊因素会影响求解过程导致无法求解，如年份一列，样本值中年份列的所有值都是一样的，这样导致由样本组成矩阵会有线性相关，从而导致该矩阵是一个奇异矩阵而无法求逆，最后无法得到系数的解，于是我们需要删除这些特殊的列，做一个线性回归自变量的选择【16】，结果最后筛选我们得出线性回归模型的函数表达式如下



经过一系列函数求解后得（结果都是保留两位小数）：

* W0 = 15594.07
* W1 = -674.10
* W2 = -0.05
* W3 = 841.35
* W4 = 1092.46
* W5 = 1558.13

于是我们的关于该样本的线性回归函数的表达式为：



代码如下  
**import** org.ujmp.core.DenseMatrix;  
**import** org.ujmp.core.Matrix;  
**import** java.util.List;  
  
**public class** LinearRegression {  
  
 Matrix **denseX**;  
 Matrix **densext**;  
 Matrix **denseY**;  
 **private** List<OneHotSample> **oneHotSamples**;  
  
 **public** LinearRegression() **throws** Exception {  
 UsedCarNnalysis usedCarNnalysis = **new** UsedCarNnalysis();  
 **oneHotSamples** = usedCarNnalysis.readFile2(**"汽车价格离群值检测/dataset/accord\_sedan\_training.csv"**);  
 *//获取输入训练数据文本的 行数* **int** rowoffile=**oneHotSamples**.size();  
 *//获取输入训练数据文本的 列数* **int** columnoffile = OneHotSample.**class**.getDeclaredFields().**length**;  
 *//去掉某些会造成线性相关的列* **denseX** = DenseMatrix.***Factory***.zeros(rowoffile, columnoffile - 4);  
 **denseY** = DenseMatrix.***Factory***.zeros(rowoffile, 1);  
 initMatrix();  
 }  
 **private void** initMatrix(){  
 **for** (**int** i = 0; i < **oneHotSamples**.size();i++) {  
 **denseX**.setAsDouble(1.0, i, 0);  
 **denseX**.setAsDouble(**oneHotSamples**.get(i).getEngine4Cyl(), i, 1);  
 **denseX**.setAsDouble(**oneHotSamples**.get(i).getMileage(), i, 2);  
 **denseX**.setAsDouble(**oneHotSamples**.get(i).getTransmissionAutomatic(), i, 3);  
 **denseX**.setAsDouble(**oneHotSamples**.get(i).getTrimEx(), i, 4);  
 **denseX**.setAsDouble(**oneHotSamples**.get(i).getTrimExl(), i, 5);  
 **denseY**.setAsDouble(**oneHotSamples**.get(i).getPrice(), i, 0);  
 }  
 **densext** = **denseX**.transpose();  
 }  
  
 **public void** result(){  
 Matrix denseXtX = **densext**.mtimes(**denseX**);  
 Matrix denseXtXInv = denseXtX.inv();  
 Matrix denseXtXInvXt = denseXtXInv.mtimes(**densext**);  
 Matrix denseXtXInvXtY = denseXtXInvXt.mtimes(**denseY**);  
 System.***out***.println(**"常数项w0："**+ denseXtXInvXtY.getAsDouble(0, 0));  
 System.***out***.println(**"Engine4Cyl w1："**+ denseXtXInvXtY.getAsDouble(1, 0));  
 System.***out***.println(**"Mileage w2："**+ denseXtXInvXtY.getAsDouble(2, 0));  
 System.***out***.println(**"TransmissionAutomatic w3："**+ denseXtXInvXtY.getAsDouble(3, 0));  
 System.***out***.println(**"TrimEx w4："**+ denseXtXInvXtY.getAsDouble(4, 0));  
 System.***out***.println(**"TrimExl w5："**+ denseXtXInvXtY.getAsDouble(5, 0));  
 }  
  
 **public static void** main(String[] args) **throws** Exception {  
 LinearRegression m = **new** LinearRegression();  
 m.result();  
 }  
}

4.6 线性回归模型离群值检测

在本样本案例中，我们选择置信水平为0.95，即认为只要超过了95%百分位数的样本数据，都被认为是异常数据。

5 局部异常因子（LOF）算法

5.1 LOF算法简介

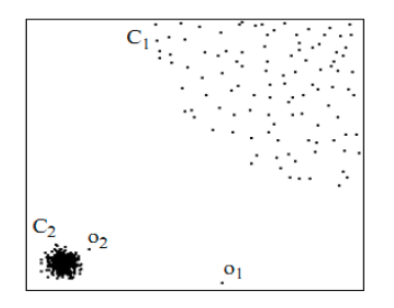
异常值检测在实质上就是寻找观测值与参照值之间有意义的偏差，样本库中的数据由于各种原因经常有包含一些异常的样本记录，我们对这些异常样本记录的检测和解释也有着重要的意义。而离群点检测是异常值检测方法当初最常用的方法之一了，离群点检测主要的目的就是为了检测出与那些正常数据样本或者特征属性差别有着较大的异常样本数据，与离群点检测相关的还有大量关于类聚算法的工作，从类聚算法的角度来看，离群点不属于数据中的对象，这是数据和行为又被称作孤立点、噪音、异常点或离群点，但还是离群点的叫法最为普遍。

离群检测算法分类：

* 基于统计的离群检测方法
* 基于聚类的离群检测方法
* 基于分类的离群检测方法
* 基于距离的离群检测方法
* 基于密度的离群检测方法
* 基于信息熵的离群检测算法

而局部离群检测算法（Local Outlier Factor）就是基于密度的离群检测算法。

LOF算法全程又叫做局部一场因子，一种基于密度的异常点检测算法，在视觉感官上的效果如下图：



对于C1的集合点群，在整体上的距离、密度、分散情况都比较的均匀一致，我们可以认为他是同一簇；而对于C2的集合点群，我们也可以认为是统一簇，但是点o1、o2相对于集合簇就相对孤立，所以可以判定这两个点就是异常点或者称之为离散点。于是我们需要一个通用的算法，实现既可以从满足C1这种密度较分散而C2密度又较紧凑的集合点中得出异常样本点，而LOF算法就可以实现这种算法。

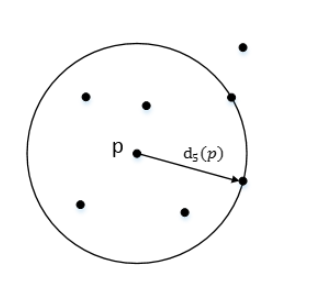
5.2 LOF算法实现过程

在LOF算法的实现过程中，有一些比较重要的定义概念：

* d(p,o)：点p与o之间的距离
* k-distance：第k距离，对于某点p的第k距离dk(p)有如下定义规则

1. dk(p) = d(p，o)
2. 在集合当中至少会有不包含点p在内的k个点o’∈ C{x≠p}且d(p,o’)≤d(p，o)
3. 在集合当中至多会有不包含点p在内的k - 1个点o’∈ C {x≠p}且d(p,o’)＜d(p，o)

如下图就可以表示为点p的第5距离，即离p第5远的点的距离：



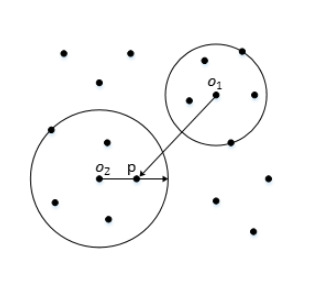
* k-distance neighborhood of p：第k距离领域；用符号Nk(p)来表示点p的第k距离领域，即点p的第k距离内的所有的点，也包含我们的k点，所以p的第k距离领域内的点的个数|Nk(p)| ≥ k
* reach distance：可达距离；点o到点p的第k可达距离表示为

reach-distancek(p,o) = max{k-distance(o)，d(p,o)}

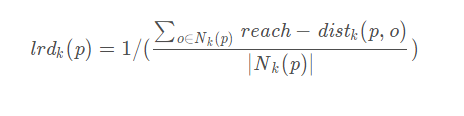
即点o到点p的第k可达距离，它至少是点o的第k距离，或者是o、p两点之间的真实的距离；这种定义也将意味着，离点o最近的k个点，他们与o的可达距离将被认为是相等的，都会是等于dk(o)。如下图：o1到点p的第5可达距离是d(p，o1),o2到p的第5可达距离为d5(o2)。

reach-distk(p，o1) = d(p,o1)

reach-distk(p，o2) = d(p,o2)



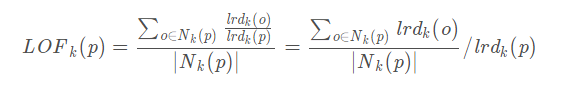
* Local reachability density：局部可达密度；该密度主要是利用上述所得到值进一步得出的，表达式如下



该表达式表示点p的第k距离领域内的有所有的点到p的平均可达距离的倒数；

通过局部可达密度，我们理解为，密度越高，就认为越可能是属于同一簇的样本，密度越低，就越可能是离群点；如果p和他周围的领域点是同一簇的话，那可达距离越可能是比较小的dk(o)，这将导致可达距离的和越小，从而使密度更高；如果p和周围的领域点比较远，则可达距离也将变得更加的大，从而导致密度变小，越是离群点

* Local outlier factor：局部离群因子；表达式如下：



表示点p的领域内的点Nk(p)的局部可达密度与点p的局部可达密度之比的平均值；我们规定如果这个值接近1，说明p与它领域内点的密度差不多，p就可能和领域是同一簇；如果这个值越小于1，说明p的密度高于它的领域内的点的密度，p为密集点；如果这个值越大于1，就说明p的密度小于他的领域内的点的密度，该点就越可能是离群点。但是对于异常点的判断没有标准，需要根据数据的实际情况出发， 在这四百多个样本数据中，以1.5为标准，超过的我们就认定它是样本样本点

5.3 LOF算法的实验结果的分析

lof算法对数据分析代码如下：

package com.hbu.aile;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
*/\*\*  
 \* @author aile  
 \* @date 2020/4/26 19:10  
 \*/*public class MyLOF {  
  
 class Point {  
 double price;  
 double mileage;  
 *//p点的第K距离* double dk\_p = 0;  
 *//p点第k距离领域内的点的个数* int Nk\_p = 0;  
 *//p的第k距离领域内素有点的可达距离之和* double reach\_dist;  
 *//记录在第k距离内领域的点的下标* List<Integer> NkIndexs = new ArrayList<>();  
 *//局部可达密度* double lrdk\_p;  
 *//局部离群因子* double lof;  
 }  
  
 public MyLOF() throws Exception {  
 UsedCarNnalysis usedCarNnalysis = new UsedCarNnalysis();  
 List<OneHotSample> oneHotSamples = usedCarNnalysis.readFile2("汽车价格离群值检测/dataset/accord\_sedan\_training.csv");  
 sample = new ArrayList<>();  
 for (OneHotSample oneHotSample : oneHotSamples) {  
 sample.add(new double[]{oneHotSample.getPrice(), oneHotSample.getMileage()});  
 }  
 distance = new double[oneHotSamples.size()][oneHotSamples.size()];  
 distance();  
 k\_distance();  
 reach\_distance();  
 localReachabilityDensity();  
 localOutlierFactor();  
 int count = 0;  
 int flag = 1;  
 for (Point point : points) {  
 if (point.lof < 0.95) {  
 System.*out*.print("[" + point.price + "," + point.mileage +"]\t" + point.lof + "\t");  
 count++;  
 if (flag++ % 3 == 0) {  
 System.*out*.println();  
 }  
 }  
 }  
 System.*out*.println("异常点个数：" + count);  
 }  
  
 private List<double[]> sample;  
 *//记录每个点之间的距离* private double[][] distance;  
 private List<Point> points = new ArrayList<>();  
  
 */\*\*  
 \* 两点之间的距离  
 \*/* private void distance() {  
 for (int i = 0;i < sample.size();i++) {  
 for (int j = 0;j < sample.size();j++) {  
 if (i == j) {  
 continue;  
 }  
 double temp = Math.*pow*(sample.get(i)[0] - sample.get(j)[0], 2) + Math.*pow*(sample.get(i)[1] - sample.get(j)[1], 2);  
 *//点i和点J之间的距离,保留两位小数  
// DecimalFormat df = new DecimalFormat("#.00");  
// temp = Math.sqrt(temp);  
// distance[i][j] = Double.parseDouble(temp);* distance[i][j] = Math.*sqrt*(temp);  
 }  
 }  
 }  
  
 */\*\*  
 \* 计算每个点的第K距离以及第k距离领域内的点的个数  
 \*/* private void k\_distance() {  
 *//计算点i的第k距离* for (int i = 0;i < sample.size();i++) {  
 double[] p\_distance = distance[i].clone();  
 Arrays.*sort*(p\_distance);  
 *//第k距离k选9* double dk\_p = p\_distance[9];  
 Point point = new Point();  
 point.dk\_p = dk\_p;  
 for (int k = 0;k < distance[i].length;k++) {  
 if (i == k) {  
 continue;  
 }  
 if (distance[i][k] <= dk\_p) {  
 point.NkIndexs.add(k);  
 }  
 }  
 point.price = sample.get(i)[0];  
 point.mileage = sample.get(i)[1];  
 point.Nk\_p = point.NkIndexs.size();  
 points.add(point);  
 }  
 }  
  
 */\*\*  
 \* 可达距离之和  
 \*/* private void reach\_distance() {  
 for (int i = 0;i < points.size();i++) {  
 double reach\_distance = 0;  
 List<Integer> tempNkIndexes = points.get(i).NkIndexs;  
 for (int j = 0;j < tempNkIndexes.size();j++) {  
 reach\_distance += Math.*max*(distance[tempNkIndexes.get(j)][i], points.get(tempNkIndexes.get(j)).dk\_p);  
 }  
 points.get(i).reach\_dist = reach\_distance;  
 }  
 }  
  
 */\*\*  
 \* 局部可达密度  
 \*/* private void localReachabilityDensity() {  
 for (int i = 0;i < points.size();i++) {  
 points.get(i).lrdk\_p = 1 / (points.get(i).reach\_dist / points.get(i).Nk\_p);  
 }  
 }  
  
 */\*\*  
 \* 局部离群因子LOF  
 \*/* private void localOutlierFactor() {  
 for (int i = 0;i < points.size();i++) {  
 double lrdko\_p = 0;  
 for (Integer nkIndex : points.get(i).NkIndexs) {  
 lrdko\_p += points.get(nkIndex).lrdk\_p;  
 }  
 points.get(i).lof = lrdko\_p / points.get(i).Nk\_p / points.get(i).lrdk\_p;  
 }  
 }  
 public static void main(String[] args) throws Exception {  
 MyLOF myLOF = new MyLOF();  
 }  
}

下面是样本中通过价格和里程获得的lof值的数据：

|  |  |  |
| --- | --- | --- |
| 异常样本点（lof值以1.5为标准） | | |
| Price | Mileage | Lof值 |
| 14999 | 101600 | 1.5178026781744869 |
| 16595 | 80390 | 2.17223641058846 |
| 15995 | 114000 | 2.059677561143481 |
| 16900 | 129000 | 2.1049160143400587 |
| 9870 | 86800 | 1.613444434361677 |
| ... | ... | ... |

|  |  |  |
| --- | --- | --- |
| 正常样本点（lof值以1.5为标准） | | |
| Price | Mileage | Lof值 |
| 14995 | 67697 | 1.3789318169159108 |
| 11988 | 73738 | 1.0255446532058652 |
| 11999 | 80313 | 0.9983136796581344 |
| 8999 | 126150 | 1.1111500822307063 |
| 11333 | 79607 | 1.0033461628826805 |
| ... | ... | ... |

分析总结：

在417个样本数据中，有14个异常样本数据，即3.4%，说明样本数据的整体可信度还是较高的；从正常的结果集合可以看到，二手车的价格，基本上和已行驶里程是反相关，即当里程越大时，二手车的价格就会相应的较低；而异常的样本数据集合中，里程大的，价格也很高，所以其被检测出为异常样本点。

参考文献

[1] 蔡素丽. 多元线性回归模型应用实证分析[J]. 廊坊师范学院学报（自然科学版）. 2017.

[2] Jing Cheng,MingyaoAi. Optimal designs for panel data linear regressions[J]. Statistics and Probability Letters. 2020.

[3] 魏伟. 带混合测量误差的线性回归模型的参数估计[J]. 河北师范大学学报（自然科学版）. 2019.

[4] 张戈. One-Hot编码在学生选课数据分析中的应用研究[J]. 网络安全技术与应用. 2019

[5] Shunntaro Okada,Masayuki Ohzeki,Shinichiro Taguchi. Efficient partition of integer optimiaztion problems with one-hot encoding[J]. Scientific Reports. 2019

[6] 张娟，李梅. 多位二进制向量矩阵乘法的研究和实现[J]. 内燃机与配件. 2019.

[7] 马晓娟. 离群点检测算法的研究与应用[J]. 沈阳工业大学. 2017.

[8] 范敏，李泽明，石欣. 一种基于区域中心点的聚类算法[J]. 计算器工程与科学. 2014.

[9] 杨红，李丹宁，王雅洁. 基于离群点检测的K-means算法[J]. 2019.

[10] 李航. 基于LOF的快速密度峰值聚类的电力数据异常值检测方法研究[J]. 通信技术. 2019.

[11] 张魏，麦志深. 核模糊谱聚类LOF降噪方法研究[J]. 广东工业大学学报. 2018.

[12] Juozas Auskainis，Nerijus Paulauskas，Algirdas Baskys. Application of Local Outlier Factor Algorithm to Detect Anomalies in Computer Network[J]. Elkektronika ir Elektrotechnika. 2018

[13] 陈永娟. 极大似然估计概念的微课程教学设计[J]. 安阳师范学院学报 . 2019

[14] 叶利娟. 概率密度函数的引入及概率表示[J]. 湖南工程学院学报（自然科学版）. 2019

[15] 戚龙. 用最小二乘法求解线性回归方程的算法研究[J]. 计算机产品与流通. 2019

[16] 刘立祥. 线性回归模型中自变量的选择与逐步回归方法[J]. 统计与决策. 2015

致 谢

论文的完成，少不了他人的关心与帮助，特此向那些在论文撰写过程中给予我帮助、指导、关心的老师、同学朋友们表示最衷心的感谢和最诚挚的谢意。

我首先需要特比的感谢我的论文指导老师河北大学数学与信息科学学院张辉老师。张辉老师对我的论文的选题、研究方向论文定稿等各个环节都给予了大力的支持和帮助，做出了核心的指导性意见和推荐，并且在论文的写作过程中张辉老师对于我遇到的问题和困难都给予悉心的指导，倾注了大量心血，提出许多非常有益的改善性意见。同时也要感谢河北大学数学与信息科学学院软件工程老师们，在这四年里面授予我专业的技能知识，让我能够有充分的专业技能完成本次设计，再次向老师们表示最衷心的感谢和最诚挚的谢意。

另外，还需要感谢朋友还有同学们，以及在论文撰写过程中给予的大力的支持和帮助，以及了论文中参考文献的作者们，通过他们的文章，让我对设计的研究有了充足的参考文档，能够更好的解决在论文的完成过程中遇到的问题。

　　最后，谢谢论文评阅老师们的辛苦工作。衷心感谢我的家人、朋友，以及同学们，真是在他们的鼓励和支持下我才得以顺利完成此论文。