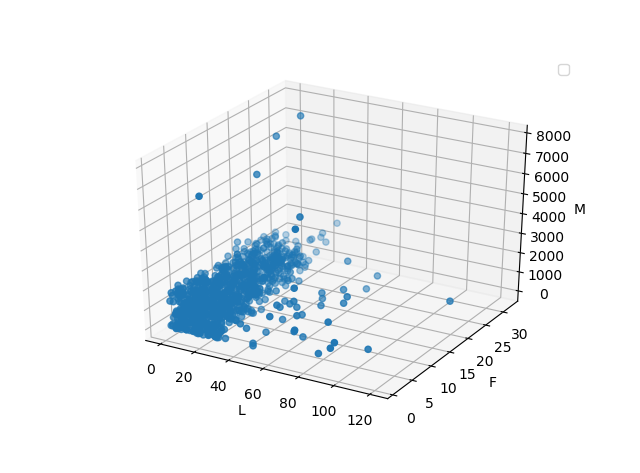
**离群点检测**

**Part I: LOF (density based)**

1. **数据处理**

* 取出数据的L、F、M列，每个数据点定义为(L, F, M)
* 对数据进行numpy序列化
* 数据参数

|  |  |  |
| --- | --- | --- |
| 数据量 | 数据点维度 | 数据点样例 |
| 940 | 3 | [ 27. 6. 232.61] |



1. **算法思路**

**（代码请看lof.py文件）**

* 定义距离为三维的欧几里得距离
* 求每个点的k邻居以及距离：首先求出每一个点p和其他所有点的距离，并从小到大排序，并获得下标顺序。而这个下标顺序就是p的邻居标号按照距离从小到大排序，去掉下标序列第一个点（p本身），从第二个点开始，到第k+1位置以后距离仍和第k+1个点的距离一致的点，就是p点的k邻居序列，记录这个序列以及对应的距离
* 求每个点的lrd：上述每个点p的k邻居距离序列的最后一个值就是点p的k distance，而距离序列中也记录了p到每个k邻居点o的距离。求解点p的lrd时，对p的每一个邻居o，取o的k distance和p与o的距离中的最小值作为p与o的reach distance，对reach distance求和，用p的k邻居数量除以reach distance之和，就可以得到每个点p的lrd
* 求每个点的LOF：由于每个点的lrd已经获得了，按照以下公式就可以求出每个点p的LOF值

1. **算法参数分析**

LOF参数分析

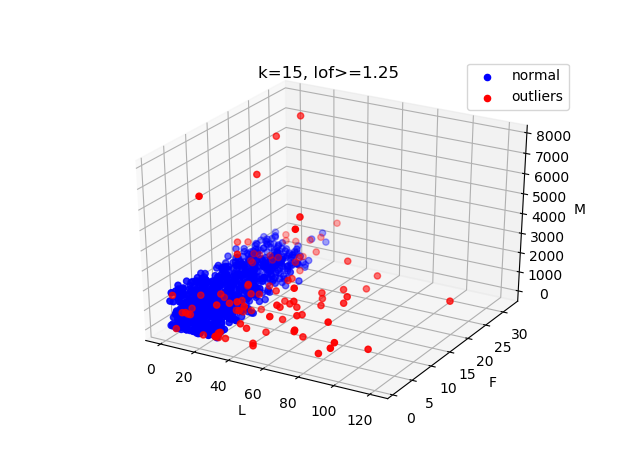
* 计算出p点的LOF如果接近1，则说明p的密度和p的邻域点密度差不多，那么p可能和邻域属于同一簇
* 如果LOF越小于1，说明p的密度高于p的邻域点的密度，那么p为密集点
* 如果LOF越大于1，则p的密度越低于p的邻域点的密度，那么p越可能为outlier

这个算法有两个参数k、threshold

* k是k邻居的范围限制，每个点p的第k远邻居以及其他和第k远邻居一样远的点，定义为点p的k邻居。如果k很小，则考虑的邻域范围小，则LOF是p的密度和一个很小的邻域内点密度的比，这样子可以找出局部的outlier，对于全局来说，会发现outlier零散地出现在大的cluster内部，因为它能检测出局部的离群点。如果k很大，同理，找到的outlier会在大的cluster外部。因此**k的取值要根据数据规模取一个适中的值**。
* threshold是LOF划分outlier的阈值，由上述对LOF数值的分析可得，满足下列公式的点p为outlier，而这个阈值通常取大于1的值。如果这个值取得越大，则outlier的数量也就少；阈值越大，则outlier数量越多。**threshold的取值大小要依据outlier判定标准来定**。

1. **离群点判断**

|  |  |  |
| --- | --- | --- |
| 参数设定 | k | 15 |
| threshold | 1.25 |
| outlier结果 | 判断依据 |  |
| 数量 | 97 |
| 运行时间 | 0.0930s |



**Part II: DBSCAN (cluster based)**

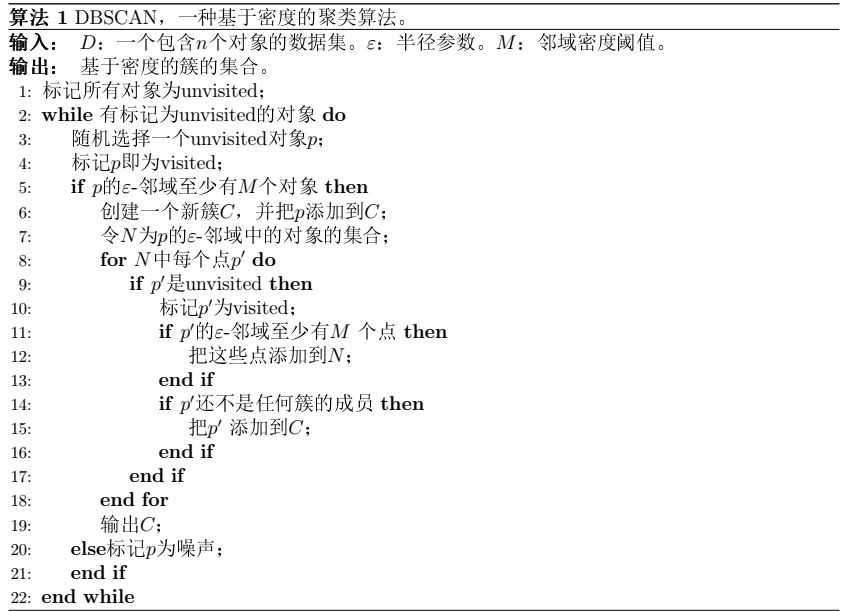
1. **数据处理**

* 和LOF算法的数据处理一样

1. **算法思路**

**（代码请看dbscan.py文件）**

* **算法关键：把点划分为核心点（稠密区域内部点）、边界点（稠密区域边界点）、噪声点，根据密度可达原理来进行cluster划分**
* 距离同样定义为三维的欧几里得距离
* unvisited集合用set实现
* 把p’的eps邻域的点添加到N时，直接数组拼接



1. **算法参数分析**

算法结果分析

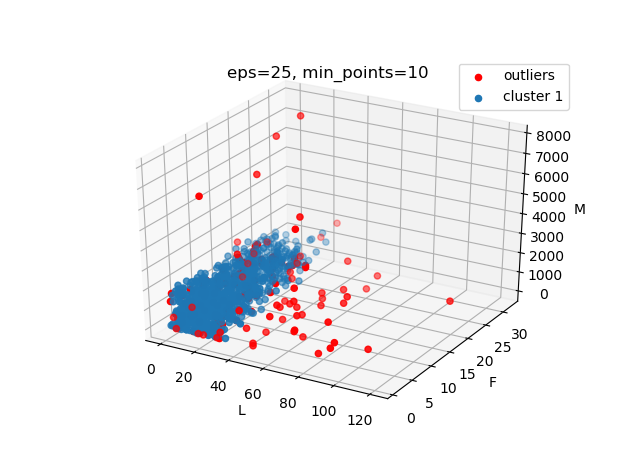
* 算法结果为对每个点的cluster标记mark

这个算法有两个参数eps、min\_points

* eps限制邻域的半径，若点p要成为核心点，则p的邻域内至少有min\_points数量的点，两个数共同决定了密度可达的能力。若eps越大，min\_point越小，则对密度要求低，outlier越少；若eps越小，min\_points越大，则对密度要求高，outlier越多。因此**eps和min\_points都要取适中的值**。
* outlier判断依据分析：该算法最终是把数据划分成若干个clusters，划分outlier时，可以限制cluster的大小，若cluster的数据量小于某个值，则把整个cluster定为outlier

1. **离群点判断**

|  |  |  |
| --- | --- | --- |
| 参数设定 | eps | 25 |
| min\_points | 10 |
| min\_cluster\_size | 2 |
| outlier结果 | 判断依据 |  |
| 数量 | 153 |
| 运行时间 | 0.0288s |



**Part III: SVDD (classification based)**

1. **数据处理**

* 首先按照上文的处理方法处理数据
* 分别对数据的每一个维度进行归一化

1. **算法思路**

**（代码请看svdd.py文件）**

* **算法关键：调用sklearn的svm.OneClassSVM，对数据进行拟合，然后预测数据**

1. **算法参数分析**

算法结果分析

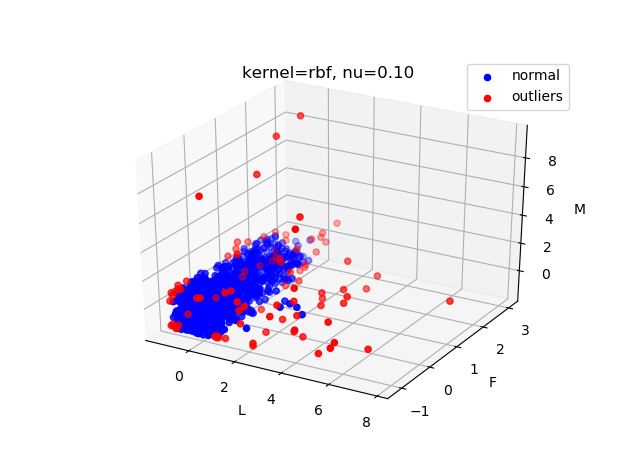
* 算法结果pred给每个点标记1和-1，其中标记为1的是正常点，标记为-1的是outlier

这个算法有两个参数kernel、nu

* kernel是指kernel函数，不同的kernel函数会有不同的效果，**需要针对数据分布来取**。通过下文结果对比可以发现，对于这次数据分布，**用Gaussian kernel更合适**。
* nu是硬度，nu越大，对outlier的容忍度越低，outlier越多，因此**nu要适中**。

1. **离群点判断**

|  |  |  |
| --- | --- | --- |
| 参数设定 | kernel | ‘rbf’ (Gaussian kernel) |
| nu | 0.1 |
| outlier结果 | 判断依据 |  |
| 数量 | 95 |
| 运行时间 | 0.0075s |



|  |  |  |
| --- | --- | --- |
| 参数设定 | kernel | ‘linear’ (linear kernel) |
| nu | 0.1 |
| outlier结果 | 判断依据 |  |
| 数量 | 284 |
| 运行时间 | 0.0026s |

