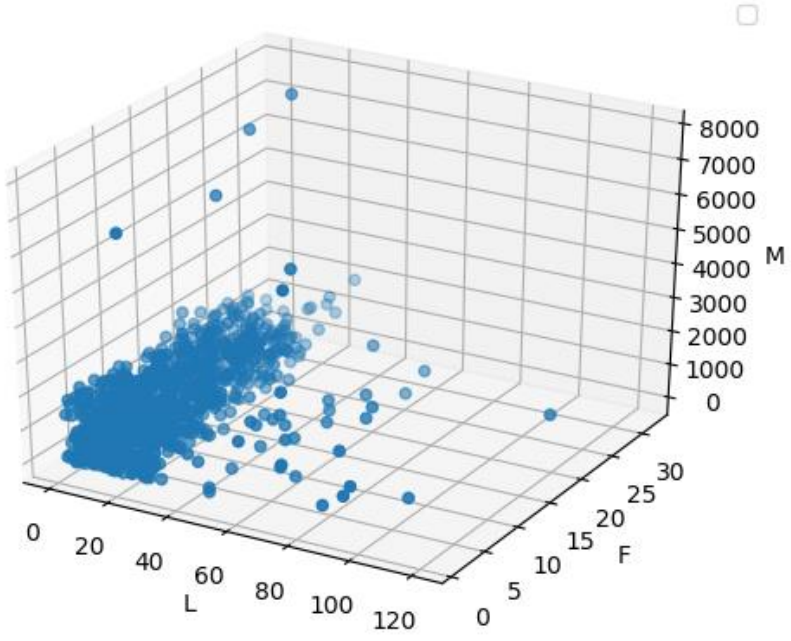
离群点检测

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]



2. 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

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} lrd_k(o)}{|N_k(p)| \times lrd_k(p)}$$

3. 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

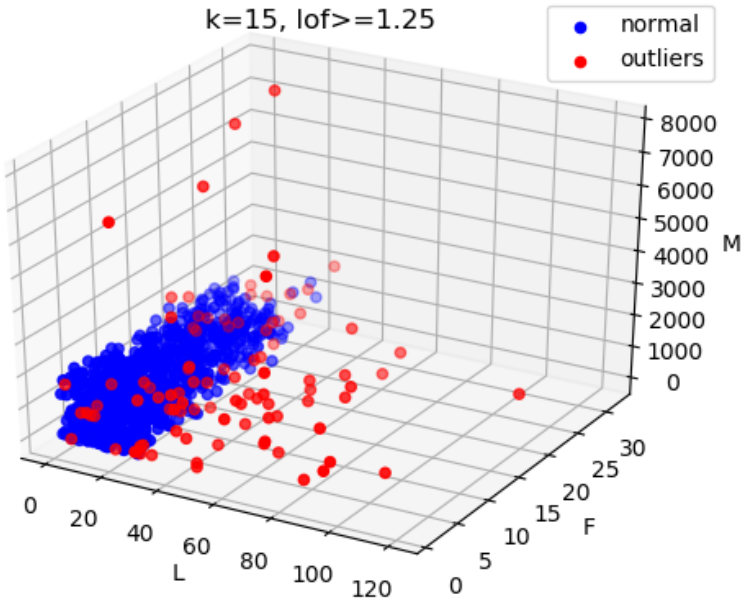
这个算法有两个参数 k、threshold

- 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**。

$$\mathrm{LOF(p)} \geq \mathrm{threshold} > 1$$

4. 离群点判断

参数设定	k	15
	threshold	1.25
outlier 结果	判断依据	$\mathrm{LOF(p)} \geq \mathrm{threshold} = 1.25$
	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

2. 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

输入： D：一个包含n个对象的数据集。ε：半径参数。M：邻域密度阈值。
输出： 基于密度的簇的集合。

```
1: 标记所有对象为unvisited;
2: while 有标记为unvisited的对象 do
3:     随机选择一个unvisited对象p;
4:     标记p即为visited;
5:     if p的ε-邻域至少有M个对象 then
6:         创建一个新簇C，并把p添加到C;
7:         令N为p的ε-邻域中的对象的集合;
8:         for N中每个点p' do
9:             if p'是unvisited then
10:                标记p'为visited;
11:                if p'的ε-邻域至少有M 个点 then
12:                    把这些点添加到N;
13:                end if
14:                if p'还不是任何簇的成员 then
15:                    把p' 添加到C;
16:                end if
17:            end if
18:        end for
19:        输出C;
20:    else标记p为噪声;
21:    end if
22: end while
```

3. 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

$$\begin{cases} mark(p) = -1, p \in outliers \\ mark(p) = i, i \geq 0, p \in cluster_i \end{cases}$$

这个算法有两个参数 eps、min_points

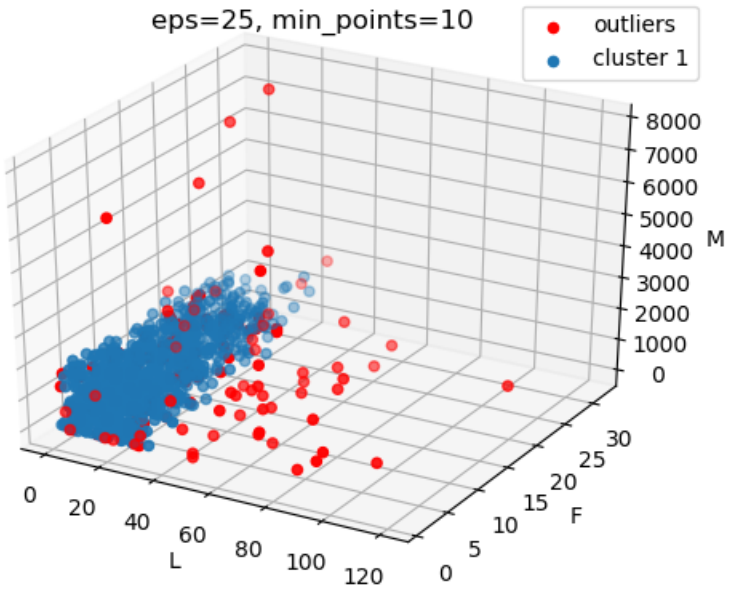
- eps 限制邻域的半径，若点 p 要成为核心点，则 p 的邻域内至少有 min_points 数量的点，两个数共同决定了密度可达的能力。若 eps 越大，min_point 越小，则对密度要求低，outlier 越少；若 eps 越小，min_points 越大，则对密度要求高，outlier 越多。因此 **eps 和 min_points 都要取适中的值**。

- 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

$$\text{size}(\text{cluster}_i) < \text{threshold}, \text{p} \in \text{outliers} \ \ \forall \text{p} \in \text{cluster}_i$$

4. 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	$\text{size}(\text{cluster}_i) < \text{min\_cluster\_size} = 2$
	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

$$\text{value} = \frac{\text{value} - \text{mean}(\text{value})}{std(value)}$$

2. 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**

3. 算法参数分析

算法结果分析

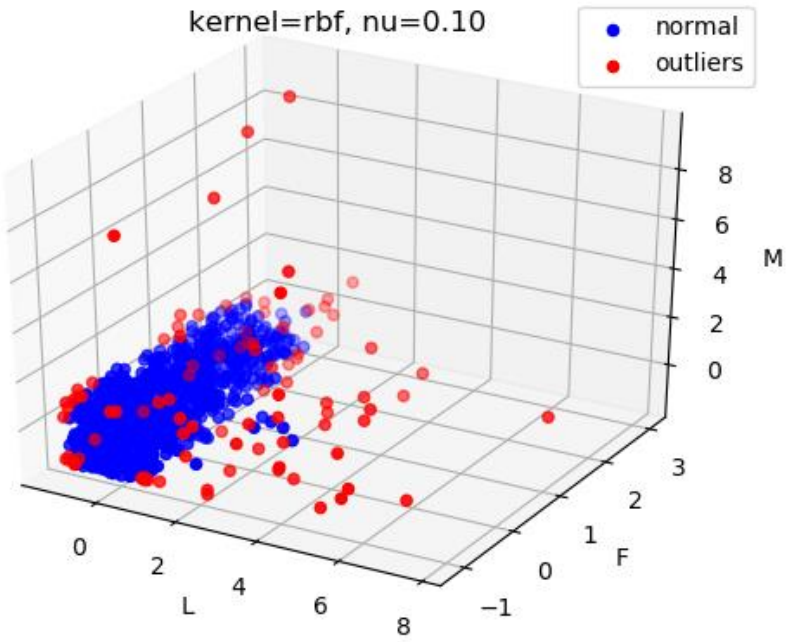
- 算法结果 pred 给每个点标记 1 和-1，其中标记为 1 的是正常点，标记为-1 的是 outlier

这个算法有两个参数 kernel、nu

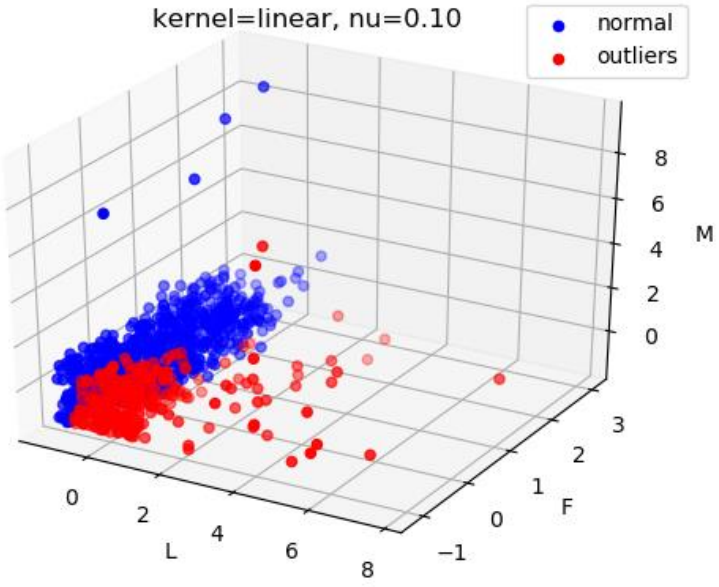
- kernel 是指 kernel 函数，不同的 kernel 函数会有不同的效果，**需要针对数据分布来取**。通过下文结果对比可以发现，对于这次数据分布，**用 Gaussian kernel 更合适**。
- nu 是硬度，nu 越大，对 outlier 的容忍度越低，outlier 越多，因此 **nu 要适中**。

4. 离群点判断

参数设定	kernel	'rbf' (Gaussian kernel)
	nu	0.1
outlier 结果	判断依据	$\mathrm{pred(p)} == -1$
	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	$\mathrm{pred(p)} == -1$
	数量	284
	运行时间	0.0026s


kernel=linear, nu=0.10
normal
outliers
M
L
F
0
2
4
6
8
−1
0
1
2
3