Q1 报告

1. 代码运行截图

a) 根据 csv 文件获取 vipno-pluno 的矩阵

```
22102005            0
22102006            0
14082037            0
22102009            0
14721018            0
11051003            0
22102012            0
14721021            0
14721022            0
14721023            0

[2635 rows x 298 columns]

Process finished with exit code 0
```

b) 任选一个 vipno，查看与此 vipno 在同一个桶中的 vipno 集合（hash_size 取总数的 0.01~0.5）

```
hash_size为2.98
输入的vipno是2900000258204
其桶中的vipno有:
2900000258204
1590142192491
hash_size为14.9
输入的vipno是2900000394193
其桶中的vipno有:
2900000394193
hash_size为29.8
输入的vipno是1591013442134
其桶中的vipno有:
1591013442134
hash_size为59.6
输入的vipno是1592015015395
其桶中的vipno有:
1592015015395
hash_size为89.39999999999999
输入的vipno是2900000869264
其桶中的vipno有:
2900000869264
hash_size为149.0
输入的vipno是1591015235529
其桶中的vipno有:
1591015235529

Process finished with exit code 0
```

2. 讨论分析部分

2.1 截图说明

截图 a）根据 trade.csv 文件的数据进行提取整合，生成了 2635 x 298 的矩阵。此处利用 Pandas 的 DataFrame 来管理矩阵。

截图 b)，根据 rand 取到任意的 vipno，输出了不同 hash_size 下给定一个 vipno 查询的相关情况。

2.2 相关说明

2.2.1 LSHash

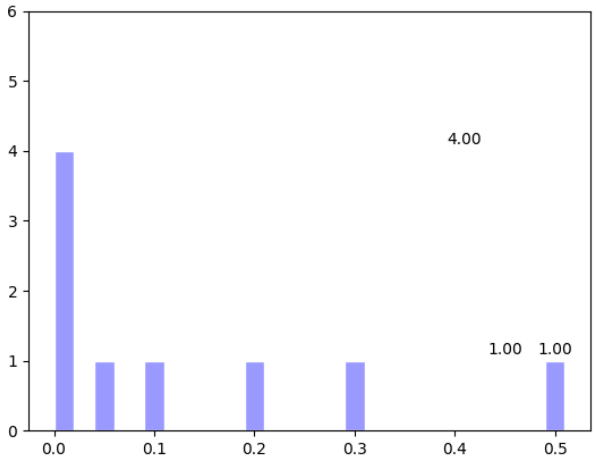
LSHash（局部敏感哈希）的基本思想是：将原始数据空间中的两个相邻数据点通过相同的映射或投影变换后，这两个数据点在新的数据空间中仍然相邻的概率很大，而不相邻的数据点被映射到同一个桶的概率很小。对原始数据集合中所有的数据都进行 hash 映射后，我们就得到了一个 hash table，这些原始数据集被分散到了 hash table 的桶内，每个桶会落入一些原始数据，属于同一个桶内的数据就有很大可能是相邻的，当然也存在不相邻的数据被 hash 到了同一个桶内。那么我们只需要将查询数据进行哈希映射得到其桶号，然后取出该桶号对应桶内的所有数据，再进行线性匹配即可查找到与查询数据相邻的数据。

在 LSHash 的函数中，input_dim 是每条数据的输入维度，hash_size 是分成桶的个数。测试时需要先放入数据，并通过 index()方法添加每条数据。这里 input_point 参数是输入的数据，extra_data 储存了每条数据所对应的标签值。最后通过 querry 可以获得某条 vipno 所在桶的信息。

2.3 效果评估

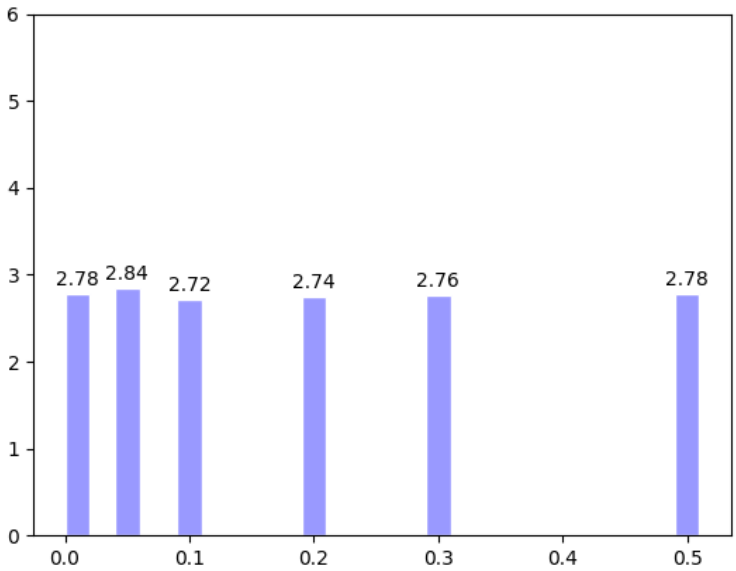
首先获得矩阵没有什么问题，唯一的问题是 pandas 处理数据的效率一般，获得矩阵的时间比较长。

然后，LSHash 中其实是含有 knn 的思想的。在任意一个桶中，任意一个 vipno 距离桶中其他数据的距离都是不定的。在根据某条 vipno 数据来 query 此桶中的其他 vipno 时，可以选取离它最近的 k 个值。通过测试，可以看出 hash_size 为 0.01 时，每个桶中的 vipno 可以达到 3-6 个；但 hash_size 为全体个数的 0.05 甚至更多的时候，一个桶中的数据个数基本为 1。



这是因为 LSHash 在根据 hash_size 定桶的数量的时候，是按照 n = 2 ^ hash_size 来取的；当 hash_size 增大，桶的数量会成指数级的增长方式增长；桶的数量变多了，每个桶中的数据条数就减少了。

3 性能图表



可以看出，hash_size 的选取对 LSHash 的时间复杂度影响不大