数据挖掘第二次作业项目文档(a)

学号：1552635　　姓名：胡嘉鑫

1. 项目说明

1.1 项目简介

本次项目所讨论的是频繁项集的挖掘，即探索数据记录之间的相关性。在生活中，如果将我们所做的一切事情都储存成数据记录的话，我们往往会发现这些记录往往具有内在的联系，这个理念应用最多的就是商品相关性的分析上。经典的购物车例子就是阐述了这个这个概念：给孩子购买尿布的成年男性，往往都有购买啤酒的倾向。那么如果我们找到了数据之间的相关性后，我们就可以分析用户购买倾向、分析商品之间潜在的联系等等。

本次项目利用已有的 10000 + 的用户商品购买记录作为挖掘频繁项集的 database，对应每个用户生成购买记录作为 transaction，探索 transaction 之间的频繁项。数据来源为 trade.csv 与 new_trade.csv，并根据两份表格中的数据分别进行挖掘探索。另外，为了考察根据交易记录生成的频繁项集是否可用，我将数据中所有用户购买记录按照时间排序，选出前 60%的数据作为训练集找出频繁项集，在根据后 40%的数据作为检验生成频繁项是否可靠的依据。

1.2 算法介绍

本次作业（a）小问要求利用 fp-growth 算法进行频繁集的挖掘。寻找到的数据集间的关系叫做关联规则。最早探索关联规则的算法叫做 Apriori，它从长度为 1 的频繁项开始算起，逐步找出长度为 2，3……的频繁项集。由于每次迭代生成长度不同的候选集额时候，会产生很多非频繁的候选集，这样会极大的占据内存空间，因此诸多频繁项集挖掘的改进算法不断被提出：

- 广度优先查询算法：Apriori
- 垂直深度优先查询算法：Eclat
- Pattern-growth 算法：FP-Growth，H-Mine，LCM

其中 FP-Growth 较前两者具有性能上的大幅提高。其算法如下：

1. 根据 transactions 筛选出其中满足阈值条件的项并构建成 FP-tree

2. 对于每个频繁项，构造它的 conditional FP-tree。

3. 对每个新构建的 FP-tree 重复这个过程，直到构造的新 FP-tree 为空，或者只包含一条路径。

4. 当构造的 FP-tree 为空时，其前缀即为频繁模式；当只包含一条路径时，通过枚举所有可能组合并与此树的前缀连接即可得到频繁模式。

这个算法通过构建树，让每个节点都存储了 database 的记录，避免了多次扫描数据库和构建非频繁项以提升效率。

1.3 数据说明

提取的数据包含五个值：

- uid：订单编号
- sldat：购买时间
- pluno：商品编号
- dptno：商品类型编号
- bndno：品牌编号

2. 代码部分

2.1 代码说明

- 算法部分：fpgrowth 文件夹下 三份文件：fp_node.py 是 fp-tree 的 node 类的定义部分，fp_tree.py 是 fp-tree 的定义部分，fp_growth.py 利用 fp-node 和 fp-tree 进行 fp-growth 的算法操作。
- 测试部分：ai.py，按照 uid 进行 transaction 的分组，读取 trade.csv 或者 new_trade.csv 数据进行测试；aii.py，按照 vipno 进行分组，读取 trade.csv 或者 new_trade.csv 数据进行测试
- 调用 ai.py 或者 aii.py 的 run_algorithm 方法，参数为：
    - Property：选择’pluno’,’dptno’或者’bndno’进行特定项的挖掘
    - Support：阈值，ai 可从 2，4，8，16，32，64 中选，aii 可从 2，4，6，8，10 中选
    - File：数据获取源，1 代表 trade.csv，2 代表 new_trade.csv

2.2 ai 代码截图

- 数据来源：trade.csv，阈值为 2，提取 pluno 项：

```
['10000015'] : 2
['15114013'] : 9
['15110071', '15114013'] : 2
['30380002', '15114013'] : 3
['14863009'] : 2
['14000005'] : 2
consuming time: 1.6488819122314453s
```

- 数据来源：trade.csv，阈值为 4，提取 pluno 项：

```
['20121061'] : 9
['15130009'] : 20
['30380002', '15130009'] : 5
['30380003', '15130009'] : 4
['15113000'] : 6
['22020002'] : 7
consuming time: 1.3421690464019775s
```

- 数据来源：trade.csv，阈值为 8，提取 pluno 项：

```
['22170001'] : 8
['25101002'] : 11
['23131002'] : 24
['22020006'] : 12
['15114033'] : 8
['15114000'] : 17
consuming time: 1.2257585525512695s
```

- 数据来源：trade.csv，阈值为 16，提取 bndno 项：

```
['14622'] : 23
[nan, '14622'] : 16
['15038'] : 21
[nan, '15038'] : 19
['15092'] : 39
[nan, '15092'] : 22
['14177'] : 23
consuming time: 1.12298583984375s
```

- 数据来源：trade.csv，阈值为 32，提取 pluno 项：

```
['22102005'] : 45
['22102014'] : 50
['27000581'] : 37
['27000582'] : 112
['27200924'] : 75
['25101044'] : 78
['22035000'] : 32
['27410003'] : 33
consuming time: 1.2297697067260742s
```

- 数据来源：trade.csv，阈值为 64，提取 pluno 项：

```
['25101044'] : 78
['23110009'] : 77
['27410000'] : 73
['27000582'] : 112
['27200924'] : 75
['30380003'] : 149
['30380002'] : 95
consuming time: 1.2723908424377441s
```

2.3 aii 代码截图

- 测试文件：trade_new.csv，阈值：2，提取 bndno 项：

```
[nan, '14177.0', '14333.0', '11052.0'] : 2
['30248.0', '14177.0', '14333.0', '11052.0'] : 2
[nan, '30248.0', '14177.0', '14333.0', '11052.0'] : 2
['30248.0', '14333.0', '11052.0'] : 2
[nan, '30248.0', '14333.0', '11052.0'] : 2
['30248.0', '11052.0'] : 2
[nan, '30248.0', '11052.0'] : 2
consuming time: 10.45891523361206s
```

- 测试文件：trade_new.csv，阈值：4，提取项：pluno

```
['22036000', '22100000'] : 4
['21021043', '22100000'] : 4
['22102005', '22100000'] : 4
['25101002'] : 7
['14722022'] : 4
['23113026'] : 7
['30380002', '23113026'] : 5
['30380003', '23113026'] : 4
['15116001'] : 6
consuming time: 3.6041226387023926s
```

- 测试文件：trade_new.csv，阈值：6，提取项：dptno

```
['23110', '14101'] : 8
['30380', '23110', '14101'] : 6
['14112'] : 13
['30380', '14112'] : 11
['15110', '14112'] : 6
['22111', '14112'] : 6
['30380', '22111', '14112'] : 6
consuming time: 5.895693063735962s
```

- 测试文件：trade_new.csv，阈值：8，提取项：pluno

```
['23110175'] : 14
['30380003', '23110175'] : 10
['14860025'] : 8
['15130035'] : 10
['27002174'] : 9
['22103011'] : 14
['30380003', '22103011'] : 9
consuming time: 3.058631658554077s
```

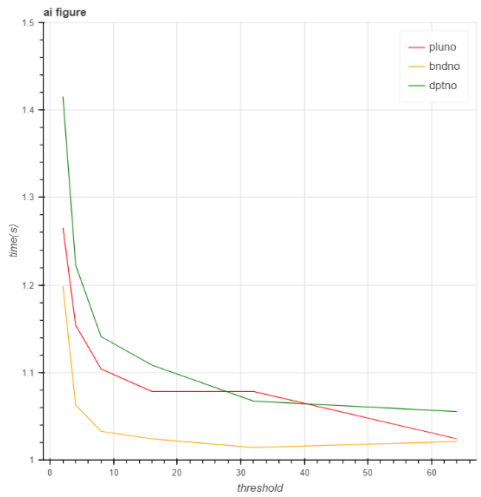
- 测试文件：trade_new.csv，阈值：10，提取项：bndno

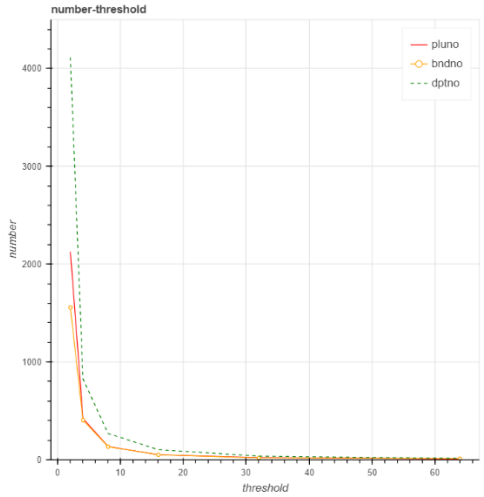
```
['15631.0', '15066.0'] : 10
[nan, '15631.0', '15066.0'] : 10
['15052.0', '15066.0'] : 15
[nan, '15052.0', '15066.0'] : 15
['30248.0', '15052.0', '15066.0'] : 11
[nan, '30248.0', '15052.0', '15066.0'] : 11
['30248.0', '15066.0'] : 33
[nan, '30248.0', '15066.0'] : 33
['15039.0', '15066.0'] : 13
[nan, '15039.0', '15066.0'] : 13
['30248.0', '15039.0', '15066.0'] : 11
[nan, '30248.0', '15039.0', '15066.0'] : 11
consuming time: 2.693781852722168s
```

3. 性能比较

对 ai 问 trade.csv 进行挖掘后的性能比较：

由图可以看出，对于 pluno，bndno 和 dptno 三个属性值来说，其运算时间随着阈值的增大逐渐减小。这是因为，随着阈值的增大，在构建 FP-tree 前的 transaction 预处理中会删除掉更多的数量小于阈值的 item，这样构建的 FP-tree 会更简单，遍历的时间也会减少。同样，由于 transaction 的 item 的减少，寻找到的频繁集的个数也会减少。





同样，可对 aii 进行相同的比较处理：

可以看出来，虽然有波动，但总体而言效率会提升。而根据挖掘出的频繁集个数可以看出，由于根据 vipno 分组的话 transaction 内元素个数增多，挖掘的频繁集个数也会增多。