数据挖掘第二次作业项目文档(b)

学号：1552635　　姓名：胡嘉鑫

1. 项目说明

1.1 项目介绍（同(a)）

1.2 算法介绍

在题目(a)中我们已经探索了如何通过构建 fp-tree 来解决频繁项集的挖掘问题。但在 fp-tree 中，每条 transaction 需要按照 item 出现的次数进行倒序排序后再插入 fp-tree 中，即忽略了 transaction 中每条项目的优先次序问题。对于挖掘有序集合的方法，我们称之为 sequential pattern 的挖掘。比较常见的 sequential pattern 算法有：

- GSP 算法
- SPADE 算法
- FreeSpan 算法
- PrefixSpan 算法

我所采用的是 PrefixSpan 算法，它通过不断找到每条 transaction 的合适的前缀来搜索频繁项。具体的算法如下：

输入：序列数据集 S 和支持度阈值 α

输出：所有满足支持度要求的频繁序列集

1）找出所有长度为 1 的前缀和对应的投影数据库

2）对长度为 1 的前缀进行计数，将支持度低于阈值 α 的前缀对应的项从数据集 S 删除，同时得到所有的频繁 1 项序列，i=1.

3）对于每个长度为 i 满足支持度要求的前缀进行递归挖掘：

a）找出前缀所对应的投影数据库。如果投影数据库为空，则递归返回。

b）统计对应投影数据库中各项的支持度计数。如果所有项的支持度计数都低于阈值 α，则递归返回。

c）将满足支持度计数的各个单项和当前的前缀进行合并，得到若干新的前缀。

d）令 i=i+1，前缀为合并单项后的各个前缀，分别递归执行第 3 步。

1.3 数据说明

我们在问题(a)中使用的 transaction 忽略了商品购买的先后次序。但我们直到，用户购买商品是有优先次序的，即一个用户（vipno）可以在不同时间购买不同的商品。因此，我根据每个用户的购买记录，将同一时间购买的商品存储在一个集合中，将一个用户所有集合按照时间的先后顺序存储成 list 作为 transaction。

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：prefix_span 包下的 prefixspan.py 是 PrefixSpan 的算法实现部分，其中 train()函数接受数据、阈值和接受的最长序列值作为输入，返回经过 PrefixSpan 查询生成的频繁项集 model。Model 的 freqSequences().collect()方法返回频繁项集。
- 测试部分：bi.py，对根据 uid 进行分组，同一时间下的订单内容放到一个 list 里，整个 transaction 的 list 顺序不可改变。Bii.py，对所有数据按照 vipno 进行分组，然后对上述数据进行相同的处理。
- 调用 bi 或者 bii 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 bi 代码截图

- 数据来源：trade.csv，阈值为 2，提取项 pluno

```
[['30380003'], ['23110009'], ['27000581']]: 2
[['30380003'], ['23110009'], ['25120016']]: 2
[['30380003'], ['23110009'], ['27240000']]: 2
[['30380003'], ['23110009'], ['22102014']]: 2
[['30380003'], ['23110009'], ['27410004']]: 2
consuming time: 2.3006186485290527s
```

- 数据来源：trade.csv，阈值为 4，提取项 bndno

```
[[nan], ['30248'], ['15052']]: 7
[[nan], ['30248'], ['15039']]: 8
[[nan], ['30248'], ['10086']]: 13
[[nan], ['30248'], ['15094']]: 13
[[nan], ['30248'], ['15094'], ['10060']]: 4
[[nan], ['30248'], ['15094'], ['14322']]: 4
consuming time: 1.701493501663208s
```

- 数据来源：trade.csv，阈值为 8，提取项 pluno

```
[['30380003'], ['15130027']]: 9
[['30380003'], ['22036000']]: 10
[['30380003'], ['22102014']]: 10
[['30380003'], ['25111048']]: 10
[['30380003'], ['22102005']]: 11
[['30380003'], ['23132068']]: 11
[['30380003'], ['23110009']]: 13
consuming time: 1.2037019729614258s
```

- 数据来源：trade.csv，阈值为 16，提取项 dptno

```
[['30380'], ['27300']]: 19
[['30380'], ['10141']]: 24
[['30380'], ['22102']]: 27
[['30380'], ['15110']]: 30
[['30380'], ['27410']]: 32
[['30380'], ['27000']]: 32
[['30380'], ['23110']]: 37
consuming time: 1.1585829257965088s
```

- 数据来源：new_trade.csv，阈值为 32，提取项 pluno

```
[['23110009']]: 110
[['30380002']]: 158
[['30380002'], ['30380003']]: 63
[['30380003']]: 230
[['30380003'], ['23110009']]: 36
[['30380003'], ['30380002']]: 42
consuming time: 2.8169896602630615s
```

- 数据来源：trade.csv，阈值为 64，提取项 pluno

```
[['30380002']]: 65
[['30380003']]: 92
consuming time: 1.1284995079040527s
```

2.3 bii 截图

- 数据来源：trade.csv，阈值为 2，提取项 pluno

```
[['30380003'], ['30380003'], ['30380003'], ['30380003'], ['22002239'], ['25120016']]: 2
[['30380003'], ['30380003'], ['30380003'], ['30380003'], ['22002239'], ['25120016'], ['25101044']]: 2
[['30380003'], ['30380003'], ['30380003'], ['30380003'], ['22002239'], ['25120016'], ['30380002']]: 2
[['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380002']]: 3
[['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380003']]: 4
[['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380003']]: 2
consuming time: 3.5635130405426025s
```

- 数据来源：trade.csv，阈值为 4，提取项 pluno

```
[['30380003'], ['30380003'], ['30380003']]: 12
[['30380003'], ['30380003'], ['30380003', '22102014']]: 4
[['30380003'], ['30380003'], ['30380003'], ['30380002']]: 4
[['30380003'], ['30380003'], ['30380003'], ['22102005']]: 4
[['30380003'], ['30380003'], ['30380003'], ['30380003']]: 8
[['30380003'], ['30380003'], ['30380003'], ['30380003'], ['30380003']]: 4
consuming time: 1.145045280456543s
```

- 数据来源：trade.csv，阈值为 6，提取项 pluno

```
[['30380003'], ['30380003', '22102014']]: 6
[['30380003'], ['30380003'], ['30380001']]: 6
[['30380003'], ['30380003'], ['23110009']]: 6
[['30380003'], ['30380003'], ['30380002']]: 7
[['30380003'], ['30380003'], ['30380003']]: 12
[['30380003'], ['30380003'], ['30380003'], ['30380003']]: 8
consuming time: 1.1555733680725098s
```

- 数据来源：trade.csv，阈值为 8，提取项 dptno

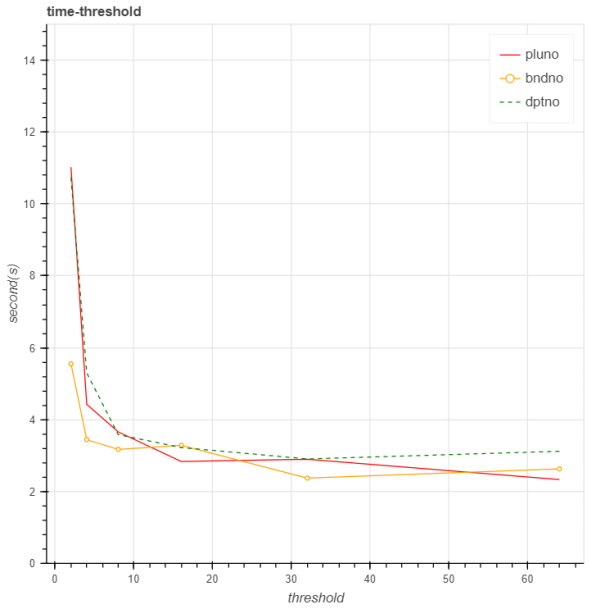
```
[['30380'], ['30380'], ['30380'], ['22102']]: 8
[['30380'], ['30380'], ['30380'], ['27410']]: 8
[['30380'], ['30380'], ['30380'], ['27300']]: 10
[['30380'], ['30380'], ['30380'], ['30380']]: 15
[['30380'], ['30380'], ['30380'], ['30380'], ['30380']]: 9
[['30380'], ['30380'], ['30380'], ['30380'], ['30380'], ['30380']]: 8
consuming time: 1.2443187236785889s
```

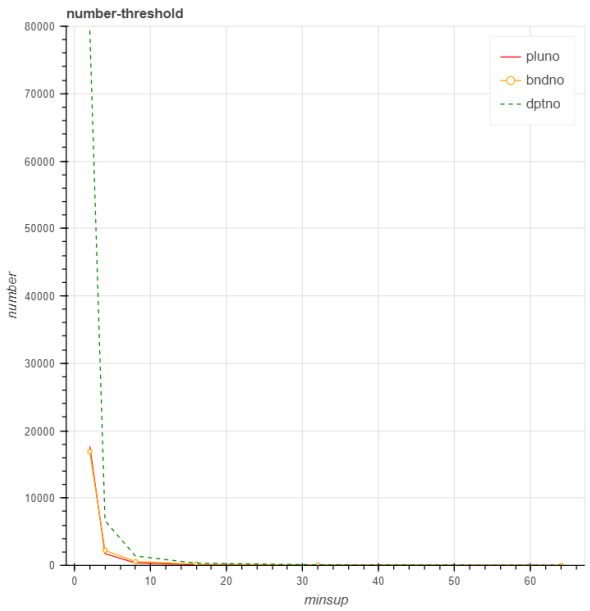
- 数据来源：trade.csv，
- 阈值为 10，提取项 pluno

```
[['30380003'], ['27410000']]: 10
[['30380003'], ['22102014']]: 10
[['30380003'], ['30380002']]: 12
[['30380003'], ['30380003']]: 31
[['30380003'], ['30380003'], ['30380003']]: 12
consuming time: 0.9555418491363525s
```

3. 性能比对

对 bi 的不同 property、min_support 来说，根据 uid 分组以后生成的频繁项集个数随着 min_support 的增粘而下降，计算时间也随着 min_support 的增长而降低，这是因为随着 min_support 的增长，筛选过后的 transaction 条数和每条内的 item 个数都会相应减少。横向比较的话，与 fp-tree 训练出的结果相比，时间会增多一些，这不仅是实现本身所带来的影响，因为随着 minsupport 达到 10 左右的时候，二者的处理速率是差不多的，主要还是因为算法本身所带来的效率问题。





同样，对 bii 问进行类似的比较操作：可以看出：根据 vipno 分组后频繁项集的个数增加了很懂，响应的处理时间也增加很多。