Department of computer science and technology

Introduction to Data Mining - Assignment *II* LaTeX

第*2*次作业

2019 年 4 月 20 日

南京大學

姓名: 张逸凯

学号: 171840708

年级: 大二

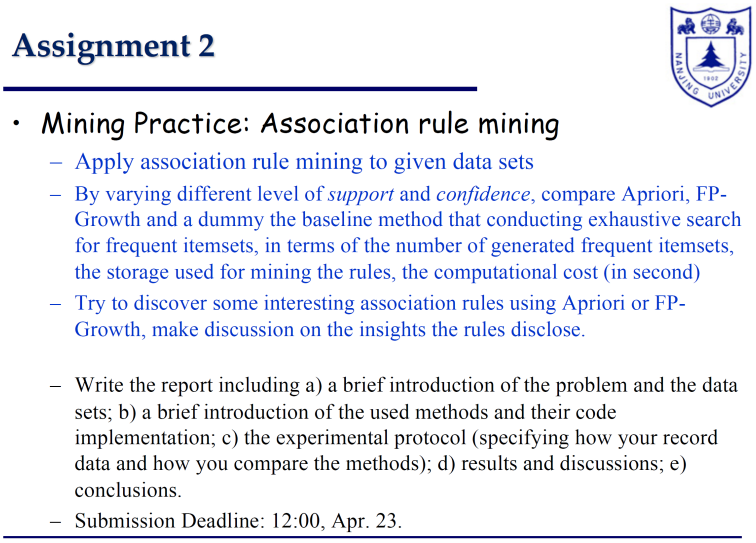
指导老师: 黎铭

邮箱: zykhelloha@gmail.com

联系电话: 18051988316

目录

1 问题重述



Assignment 2

- Mining Practice: Association rule mining
  - Apply association rule mining to given data sets
  - By varying different level of *support* and *confidence*, compare Apriori, FP-Growth and a dummy the baseline method that conducting exhaustive search for frequent itemsets, in terms of the number of generated frequent itemsets, the storage used for mining the rules, the computational cost (in second)
  - Try to discover some interesting association rules using Apriori or FP-Growth, make discussion on the insights the rules disclose.
  - Write the report including a) a brief introduction of the problem and the data sets; b) a brief introduction of the used methods and their code implementation; c) the experimental protocol (specifying how your record data and how you compare the methods); d) results and discussions; e) conclusions.
  - Submission Deadline: 12:00, Apr. 23.

注意分析Apriori和FP-growth的差别, 包括时间开销(算法进行时每层的开销), 空间开销, 注意最后分析数据, 发现那些有趣的规则背后的东西.

2 代码实现与Apriori, FP-growth开销及适用性对比

2.1 Apriori

2.1.1 时间开销

参考课本算法标识, 对于其中的apriori_gen()函数进行讨论, 因为连接步和剪枝步都在apriori_gen()函数里:

按照黎老师要求计算Apriori每一层连接扩展的时间:

表 support == 0.01 confidence == 0.5 时Apriori时间开销

k(C中项集个数)	扩展所耗时间(单位: 秒)
1	0.219560861588
2	4.34375596046
3	3.37926101685
4	0.128403902054

在源代码中插桩时间戳:

```
######时间戳!!!!!
startt = time.time()

itemSet, transactionList = getItemSetTransactionList(data_iter)
freqSet = defaultdict(int)
largeSet = dict()
#Global dictionary which stores (key=n-itemSets,value=support)
#which satisfy minSupport
assocRules = dict()
#Dictionary which stores Association Rules
oneCSet = returnItemsWithMinSupport(itemSet,
                                    transactionList,
                                    minSupport,
                                    freqSet)

######时间戳!!!!!
currentLSet = oneCSet
k = 2

endd = time.time()
print endd - startt
startt = endd

while(currentLSet != set([])):
    largeSet[k - 1] = currentLSet
    currentLSet = joinSet(currentLSet, k)
    currentCSet = returnItemsWithMinSupport(currentLSet,
                                            transactionList,
                                            minSupport,
                                            freqSet)
    currentLSet = currentCSet
    ######时间戳!!!!!
    endd = time.time()
    print endd - startt
    startt = endd
    k = k + 1
```

2.2 FP-growth

2.2.1 时间开销

以support == 0.01 confidence == 0.5的Groceries数据集为例:

将每插入树里面一个事项都记录时间, 有很多很多写到了Excel里面, 下面列出表的一部分, 这是非常快的.

建树总时间: 0.9857611656188965 (单位:秒)

1	['citrus[S]fruit', 'margarine', 's	7.15256E-07
2	['yogurt', 'tropical[S]fruit', 'co	4.76837E-07
3	['whole[S]milk']	2.38419E-07
4	['yogurt', 'pip[S]fruit', 'cream[	2.38419E-07
5	['whole[S]milk', 'other[S]vege	2.38419E-07
6	['whole[S]milk', 'yogurt', 'butt	2.38419E-07
7	['rolls/buns']	0
8	['other[S]vegetables', 'rolls/bu	0
9	['pot[S]plants']	2.38419E-07
10	['whole[S]milk']	2.38419E-07
11	['other[S]vegetables', 'bottlec	2.38419E-07
12	['whole[S]milk', 'yogurt', 'bott	4.76837E-07
13	['beef']	2.38419E-07
14	['rolls/buns', 'soda', 'frankfurt	2.38419E-07
15	['tropical[S]fruit', 'chicken']	2.38419E-07
16	['newspapers', 'fruit/vegetabl	2.38419E-07
17	['fruit/vegetable[S]juice']	2.38419E-07
18	['packaged[S]fruit/vegetables	2.38419E-07
19	['chocolate']	2.38419E-07
20	['specialty[S]bar']	2.38419E-07
21	['other[S]vegetables']	2.38419E-07
22	['pastry', 'butter[S]milk']	4.76837E-07
23	['whole[S]milk']	0
24	['tropical[S]fruit', 'newspapers	0
25	['other[S]vegetables', 'rolls/bu	0
26	['bottled[S]water', 'canned[S]	2.38419E-07
27	['yogurt']	4.76837E-07
28	['rolls/buns', 'soda', 'sausage'	2.38419E-07
29	['other[S]vegetables']	2.38419E-07
30	['soda', 'shopping[S]bags', 'ne	0
31	['yogurt', 'bottled[S]water', 'su	2.38419E-07

建树之后的搜索时间: 0.18672990798950195 (单位:秒)

如下在源代码中设置时间戳:

```
        #·perform·fp·growth
        self.frequent_patterns·=·dict()
        startt·=·time.time()
        self.fp_growth(None,·self.threshold,·self.header_table,
                       self.frequent_patterns)
        print(time.time()···startt)
```

2.3 两算法对比分析

Apriori采用广度优先的搜索方式，缩小搜索空间用到了先验性质：频繁项集的所有非空子集必然也是频繁的。比如 同时包含项AB的记录条数肯定比只包含A的记录少。这条性质也可以这么说：如果一个项集是非频繁的，那么它的超集必然也是非频繁的。

而fp-growth采用了一个叫fp-tree的数据结构去压缩存储数据集，放到内存里, fp-tree像是一种前缀树, 节点把项的次数记录下来, **用字母表的频率降序, 出现次数多的项作为共享前缀的概率也大，fp-tree的压缩率就高.** 注意这里需要把fp-tree中所有枝干、叶子上相同的项全串一起，这样项从一个起点开始，向树根遍历，就能得到包含这个项的子数据库了.

所以首先算法是压缩数据库，过滤非频繁项，得到一棵fp-tree 1号, 通过兄弟链，遍历树找出包含这棵树的子树(condition pattern tree), 然后把这个子树当做一个新的数据库2号，过滤2号子树非频繁项, 这样继续下去.

综上所述: apriori算法多次扫描交易数据库, 每次利用候选频繁集产生频繁集, 时间开销更差; 而FP-growth则利用树储存，**无需产生候选频繁集而是直接得到频繁集，大大减少扫描交易数据库的次数**，时间开销更出色, 但是空间开销上肯定加大了. 这种差别就是这两种算法形式不同原理不同造成的.

2.4 a dummy the baseline method

比如顾客买了32件商品(Groceries数据集最大值), 那么就有$\sum_{i=1}^{32} C_{32}^{i}$种排列方法, 这个所有构成的项集不加优化全部扫过去...可能要跑上一天!

不妨做一个模糊估计: Groceries数据集一共有9835条记录, 单次购买数量平均为: 4.409456024个物品, 不妨按4个算: 每个事项有组合 $\sum_{i=1}^{4} C_{4}^{i} = 15$ 种. 两两比较集合(不加优化) 折半顾客数之后也有15^{4000+}次比较, 这个量级不是我们想要的计算[1].

[1]**本次作业代码实现部分附注**: 我自己实现了一个有小Bug的Apriori, 就是只能从一个商品生成规则不能实现多个商品推出的规则, 大二外院系小朋友真的好菜呜呜呜, 然后我就找到了GitHub上面很多star的一个Apriori以及FP-growth项目, 很辛苦地看了源代码, 调整自己的思路做了分析, 这些程序包括我自己实现的都会在附件中提交, 希望老师谅解.

3 Groceries数据分析(有趣的规则!)

3.1 数据预处理

UNIX users 数据集有许多无用指令, 比如 < 1 >, / ∗ − − − − − − − − − − − − − − − − − −∗, 就是说有很多的无用数据点, 这一个数据集的数据处理占了很多时间.

因为Apriori程序对空格不敏感, 我就没有替换, 在FP-growth程序中, 我把所有空格替换成了[S], 以在后期更方便地处理.

FP-growth中过滤了每次shell开启后的重复操作, 以符合FP-growth算法初衷.

3.2 Grocery数据

首先对全局数据做一个分析:

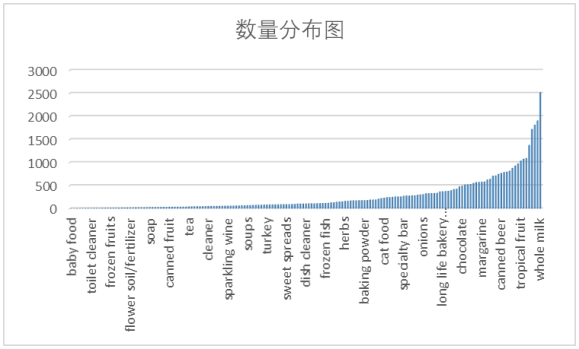
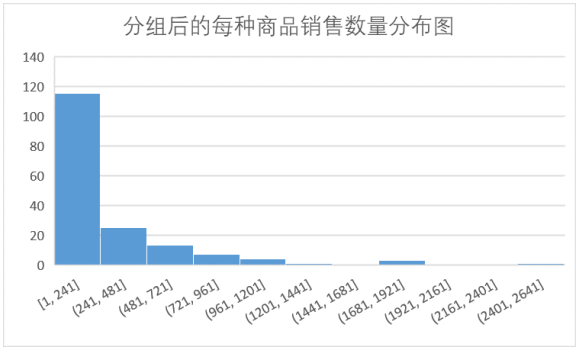


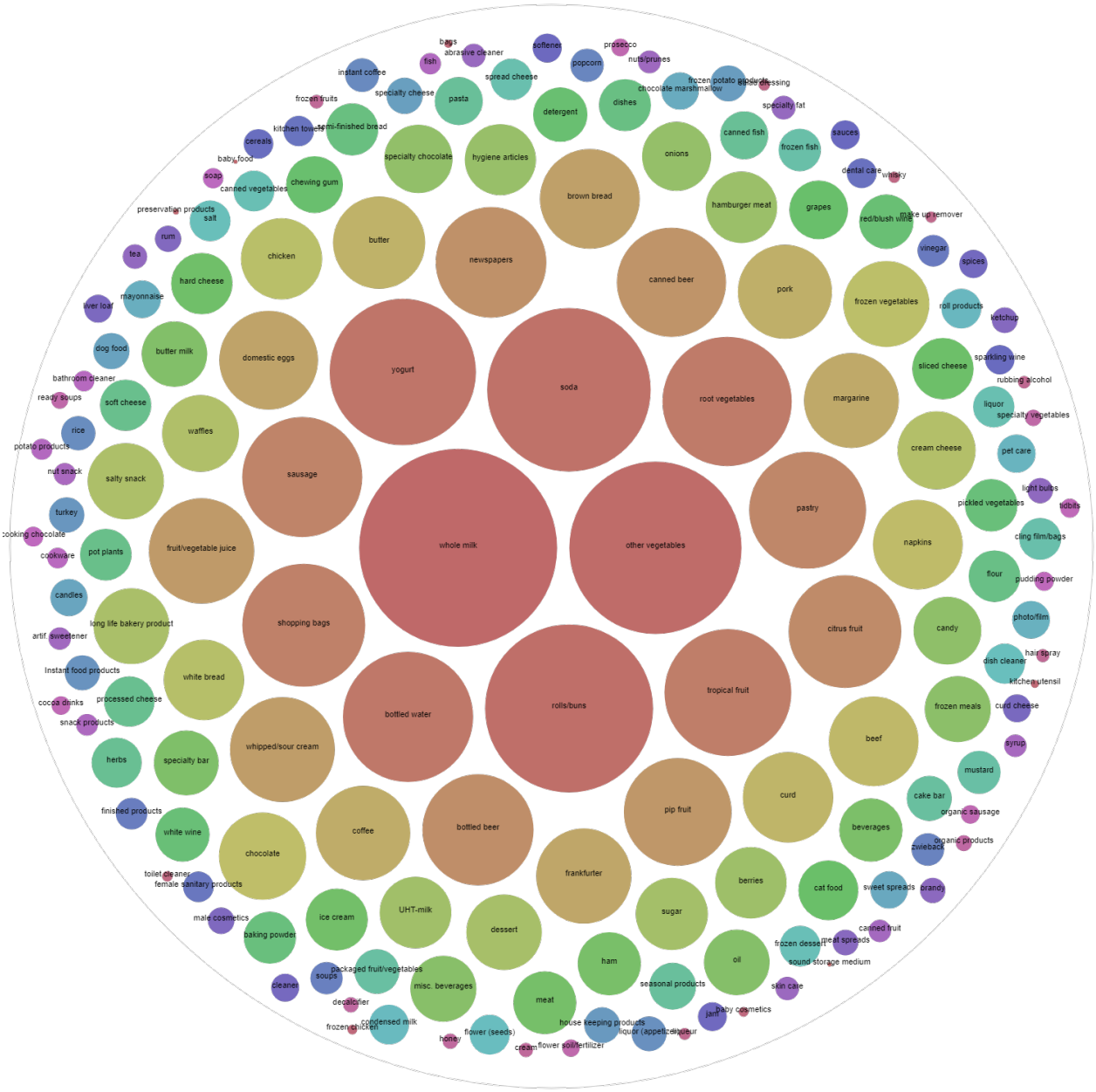
图 1: 每种商品销售数量分布图



一共有9835条记录, 单次购买数量平均数为: 4.409456024

下面绘制了使用circle packing的Hierarchy(weighted) 图像, (好吧确实有点那个), 但是从这个图中我们可以看出各个商品的大致分布, 还有整个商品售卖的密集程度, 种类丰富度:

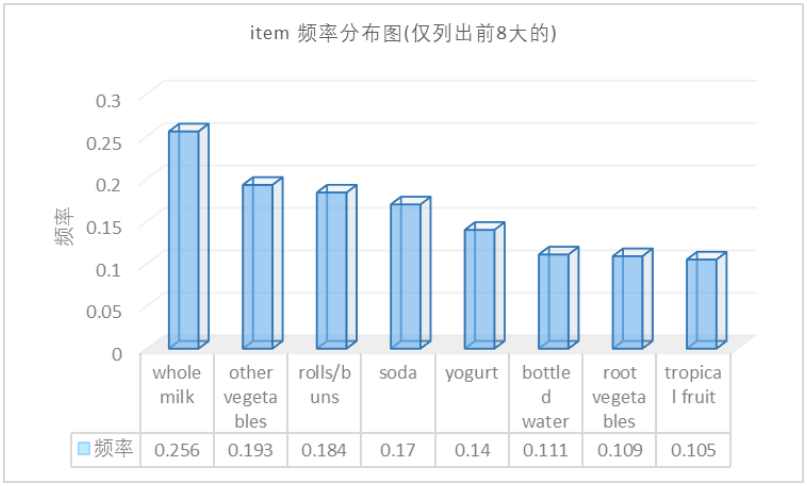
(这个图可能对您不太友好(过于密集), 但是它真的可以展现一下数据的呜呜呜...)



3.3 交易数据

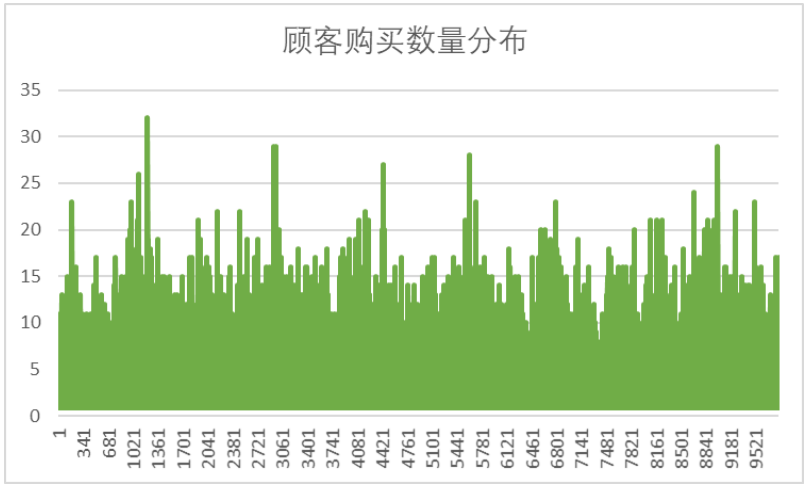
3.3.1 购买频率分布

下面绘制了所有item计算的独立的Absolute Item Frequency, 可以看出whole milk产品在这一个月的销售中卖得最好, 卖得比较好的前八个都是生鲜蔬菜, 或者即食类产品, 可以初步推断这是一个有售卖特色的Grocery.



3.3.2 单次购买数量分布

下面x轴是第几位顾客, y轴是购买的数量. (这个体量, 虚拟机里的Python竟然带不起来...



// 好像看不出什么, 不妨统计一下人数吧:

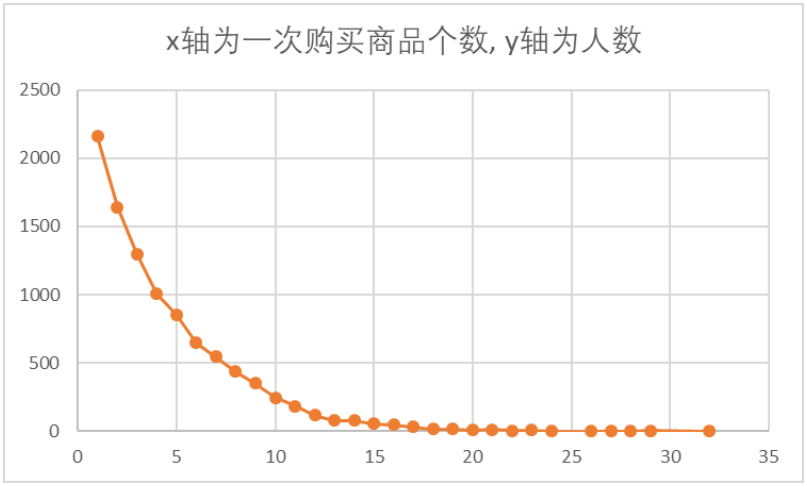


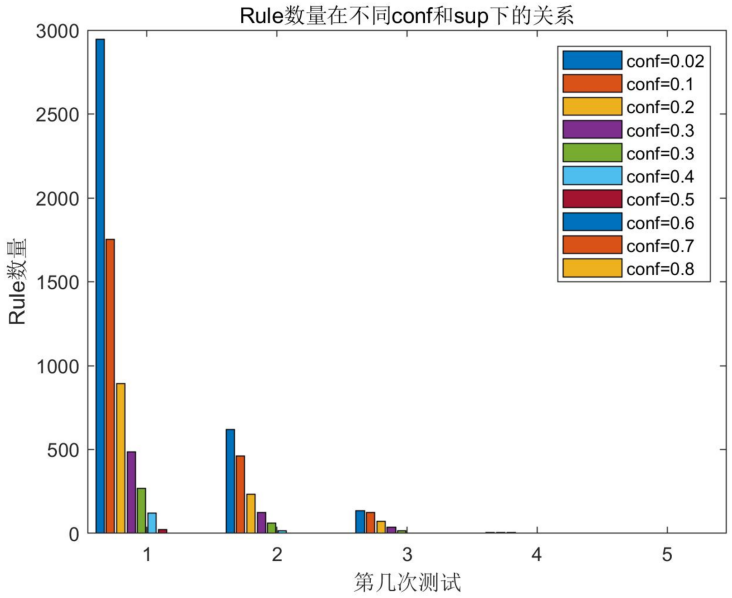
图 2: 单次购买量

3.3.3 总结

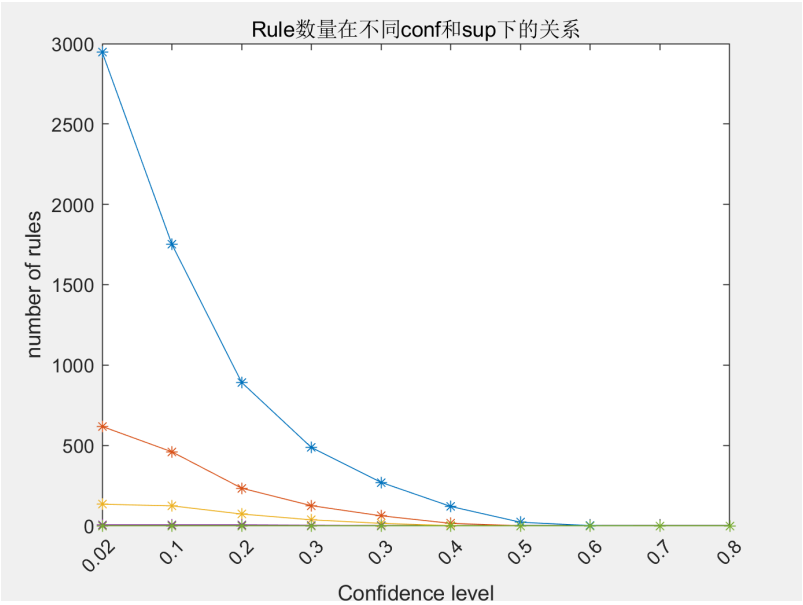
至此preprocessing已经做得差不多了, 我们对数据也有了大致的了解, 首先这个Grocery的商品很丰富, 有170种, 各商品销售量分布见(图1), 其次大部分购买者单次购买量分布在1 5之间, 最高一次购买了32件商品, 单次购买量分布见图2, 然后还有聚类可视化图, 购买数量分布图更好地展现了所给数据. 数据中噪点较少或者对接下来的分析无大的影响, 数据清理至此已经完成.

3.4 support and confidence 的选择

- 创建一组关联规则，第一步是确定support和confidence的optimal thresholds。如果这些值设置得过低，那么算法将花费很长的时间来执行，并且我们将得到许多规则(**其中大多数都没有用处**)。所以我尝试了不同的support和confidence，并以图形化的方式查看每个组合生成了多少规则。



上图中第i次测试中使用的support level为 $x_i = \{0.5\%, 1\%, 2\%, 5\%, 10\%\}$
用折线图, 并把他们画在一起会更加可视化!
同理, 下图support level从上到下为 $\{0.5\%, 1\%, 2\%, 5\%, 10\%\}$



分析一下结果:

- support level为10%, 5%。几乎没有确定任何规则, 意味着在我们的数据集中没有相对频繁的关联。我们不能选择这个值，结果规则不具有代表性, 或者说这样根本得不出结果
- support level5%。我们只确定几个至少有50% 置信度的规则。所以必须寻找低于5%的support level，才能有合理的confidence或者获得更多的规则。
- support level为1%。可以得到几十条规则，其中有一些至少有50%的confidence。
- support level 0.5%。太多的规则啦, 还是算了吧

综上所述，我们将使用1%的support level和50%的confidence。

3.5 关联规则分析

3.5.1 关联规则预处理

就像课本上说的那样, 有些规则是无趣的或者说是错误的, 比如shopping bag ==> sth. 因为这肯定是买了东西才买购物袋. 比如UNIX users 数据集中大量的exec.

3.5.2 可视化提取的规则

下面列举一下刚刚说的 > 50% 的规则:

例如 $support\,level == 0.01, confidence == 0.5$:

- Rule: ('yogurt', 'root vegetables') ==> ('other vegetables',) , 0.500
- Rule: ('rolls/buns', 'root vegetables') ==> ('other vegetables',) , 0.502
- Rule: ('other vegetables', 'whipped/sour cream') ==> ('whole milk',) , 0.507
- Rule: ('yogurt', 'other vegetables') ==> ('whole milk',) , 0.513
- Rule: ('tropical fruit', 'yogurt') ==> ('whole milk',) , 0.517
- Rule: ('pip fruit', 'other vegetables') ==> ('whole milk',) , 0.518
- Rule: ('rolls/buns', 'root vegetables') ==> ('whole milk',) , 0.523
- Rule: ('yogurt', 'whipped/sour cream') ==> ('whole milk',) , 0.525
- Rule: ('other vegetables', 'domestic eggs') ==> ('whole milk',) , 0.553
- Rule: ('yogurt', 'root vegetables') ==> ('whole milk',) , 0.563
- Rule: ('tropical fruit', 'root vegetables') ==> ('whole milk',) , 0.570
- Rule: ('butter', 'other vegetables') ==> ('whole milk',) , 0.574
- Rule: ('yogurt', 'curd') ==> ('whole milk',) , 0.582
- Rule: ('tropical fruit', 'root vegetables') ==> ('other vegetables',) , 0.585
- Rule: ('citrus fruit', 'root vegetables') ==> ('other vegetables',) , 0.586

这些规则是在讲述什么呢:

比如倒数第三个: 表示买了’yogurt’, ’curd’的顾客中有58.2%也买了whole milk, 接下来往上看:

买了’butter’, ’other vegetables’的顾客中有57.4%也买了whole milk

买了’tropical fruit’, ’root vegetables’的顾客中有57.0%也买了whole milk

买了’yogurt’, ’root vegetables’的顾客中有56.3%也买了whole milk

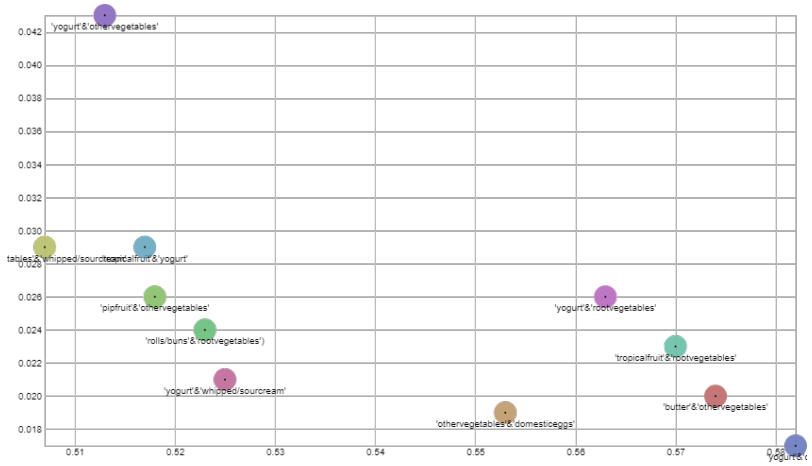
买了’other vegetables’, ’domestic eggs’的顾客中有55.3%也买了whole milk

.....

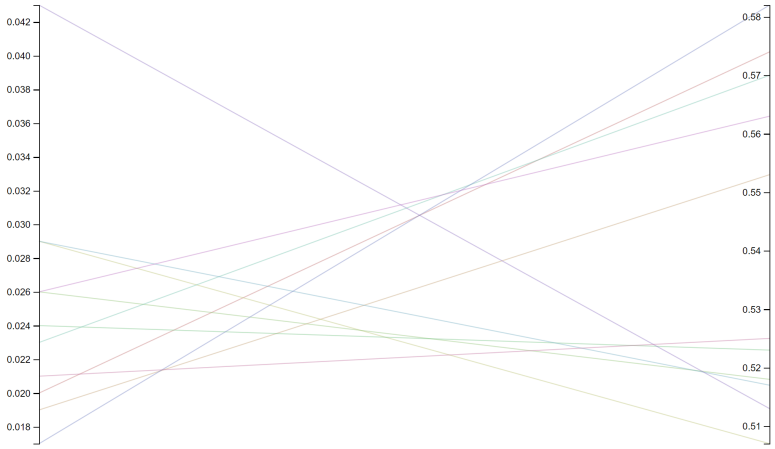
绘制如下表格

item	supp	conf
'othervegetables'&'whipped/sourcream'	0.507	0.029
'yogurt'&'othervegetables'	0.513	0.043
'tropicalfruit'&'yogurt'	0.517	0.029
'pipfruit'&'othervegetables'	0.518	0.026
'rolls/buns'&'rootvegetables')	0.523	0.024
'yogurt'&'whipped/sourcream'	0.525	0.021
'othervegetables'&'domesticeggs'	0.553	0.019
'yogurt'&'rootvegetables'	0.563	0.026
'tropicalfruit'&'rootvegetables'	0.57	0.023
'butter'&'othervegetables'	0.574	0.02
'yogurt'&'curd'	0.582	0.017

我们打一个丑陋但是很有用的散点图 (展示多维数据, 比较值集之间的关系)



或许下面的Parallel Coordinate能更好的展现不同confidence和support level之间的关系, 左边的维度是confidence, 右边是support level:



3.5.3 分析有趣的规则

分析后可以发现, confidence与support level并没有直接的正相关或者负相关关系, 也就是说, **上面的这些可以导出买了the whole milk 的商品(特别是那些强关联的商品), 可能是support并不高的**, 针对这一特点从下面这一条规则出发, 我有很细致思考:

Rule: ('yogurt', 'other vegetables') ==> ('whole milk',) , 0.513

首先来看定义:

support 指示项集在数据集中出现的频率:

$$supp(X \Rightarrow Y) = \frac{|X \cup Y|}{n}$$

对于一条规则$X \Rightarrow Y$，confidence level显示了购买X 后导出购买Y的百分比，它表明了规则被发现为正确的频率:

$$conf(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}$$

这里 Y 就是 the whole milk, 从定义出发我们就知道了: 'yogurt', 'other vegetables' 这样的组和, 可能很少人买这样的组和, 但是买了这样组和的顾客, 很大一部分都又去买 the whole milk 了, 或许是因为'yogurt', 'other vegetables' 搭配 the whole milk 是很棒的或者是这个地区人们喜欢的某种菜的做法, 这很有趣! 这时候就可以依靠数据大胆猜测了:

或许yogurt和other vegetable的商品配置出现了问题, 可以更好地引导人们买这个组合!

下面我就来说说我为什么这么猜:

首先调出所有关于 yogurt, other vegetables之类的规则:

1 Rule: ('yogurt',) ==> ('other vegetables',) , 0.311

2 Rule: ('whole milk', 'bottled water') ==> ('other vegetables',) , 0.314

3 Rule: ('whole milk', 'rolls/buns') ==> ('other vegetables',) , 0.316

4 Rule: ('whole milk', 'pastry') ==> ('other vegetables',) , 0.318

5 Rule: ('whole milk', 'sausage') ==> ('other vegetables',) , 0.340

6 Rule: ('tropical fruit', 'other vegetables') ==> ('yogurt',) , 0.343

7 Rule: ('whole milk', 'soda') ==> ('other vegetables',) , 0.348

8 Rule: ('other vegetables', 'whipped/sour cream') ==> ('yogurt',) , 0.352

9 Rule: ('sausage', 'other vegetables') ==> ('whole milk',) , 0.377

10 Rule: ('yogurt', 'whole milk') ==> ('other vegetables',) , 0.397

11 Rule: ('yogurt',) ==> ('whole milk',) , 0.402

12 Rule: ('tropical fruit', 'whole milk') ==> ('other vegetables',) , 0.404

13 Rule: ('whole milk', 'domestic eggs') ==> ('other vegetables',) , 0.410

14 Rule6: ('yogurt', 'tropical fruit') ==> ('other vegetables',) , 0.420

15 Rule: ('other vegetables', 'soda') ==> ('whole milk',) , 0.425

16 Rule: ('citrus fruit', 'whole milk') ==> ('other vegetables',) , 0.427

17 Rule: ('whole milk', 'pip fruit') ==> ('other vegetables',) , 0.449

18 Rule: ('citrus fruit', 'other vegetables') ==> ('whole milk',) , 0.451

19 Rule: ('whole milk', 'whipped/sour cream') ==> ('other vegetables',) , 0.454

20 Rule: ('whole milk', 'pork') ==> ('other vegetables',) , 0.459

21 Rule: ('other vegetables', 'pastry') ==> ('whole milk',) , 0.468

22 Rule: ('other vegetables', 'root vegetables') ==> ('whole milk',) , 0.489

23 Rule: ('yogurt', 'whipped/sour cream') ==> ('other vegetables',) , 0.490

24 Rule: ('other vegetables', 'fruit/vegetable juice') ==> ('whole milk',) , 0.498

25 Rule: ('yogurt', 'root vegetables') ==> ('other vegetables',) , 0.500

26 Rule: ('other vegetables', 'whipped/sour cream') ==> ('whole milk',) , 0.507

27 Rule: ('yogurt', 'other vegetables') ==> ('whole milk',) , 0.513

28 Rule: ('pip fruit', 'other vegetables') ==> ('whole milk',) , 0.518

29 Rule: ('yogurt', 'whipped/sour cream') ==> ('whole milk',) , 0.525

30 Rule: ('other vegetables', 'domestic eggs') ==> ('whole milk',) , 0.553

31 Rule: ('yogurt', 'root vegetables') ==> ('whole milk',) , 0.563

32 Rule: ('butter', 'other vegetables') ==> ('whole milk',) , 0.574

33 Rule: ('yogurt', 'curd') ==> ('whole milk',) , 0.582

注意: 上面是所有关于 'yogurt', 'other vegetables' 以及 'whole milk'的规则, 其中confidence >= 0.3, 分析这里的数据, 首先让我感到敏感的一点: 这里的相关于'yogurt', 'other vegetables'的confidence比例都在0.5以下! 这很重要似乎在透露着什么信息.

不妨用一个量化分析的思路[2]:

指标变量X_{conf}

- $X_{conf} = 1$,当$confidence \in [0.3, 0.4)$: 较为不好的关联
- $X_{conf} = 2$,当$confidence \in [0.4, 0.5)$: 中等的关联
- $X_{conf} = 3$,当$confidence \in [0.5, +\infty)$: 强关联

.

遍历评估$C_3^2 = 3$**种**'yogurt', 'whole milk', 'whole vegetable'组合

计算方法: 每种规则定义前件 $\Rightarrow$ 后件, 只要前件或者后件中出现'yogurt', 'whole milk', 'whole vegetable'中的一种, 就可以定义为以此种商品为前件/后件的规则:

例如上面的规则2 (*2 Rule: ('whole milk', 'bottled water') ==> ('other vegetables',) , 0.314*) 可以定义为 $vegetable \Longleftrightarrow milk$ 中指标变量$X_{conf} + 1$, (+ 1 是因为 $0.314 \in [0.3, 0.4)$.

[2](本节附注: 这里的分析是我综合Kaggle上以往数据分析的经验, 然后X_{conf}的分析之类是自己想出来的嘞, 也不知道靠不靠谱(笑哭))

Rule	$yogurt \Longleftrightarrow milk$	$yogurt \Longleftrightarrow vegetable$	$vegetable \Longleftrightarrow milk$
1	+0	+1	+0
2	+0	+0	+1
3	+0	+0	+1
4	+0	+0	+1
5	+0	+0	+1
6	+0	+1	+0
7	+0	+0	+1
8	+0	+1	+0
.	.	.	.
.	.	.	.
.	.	.	.
31	+3	+1	+0
32	+0	+0	+3
33	+3	+1	+0
权值和	21	12	32

把这些权值和算出来之后就很清晰了! yogurt, whole milk和vegetable的关联性高主要是由$yogurt \Longleftrightarrow milk$ 和 $vegetable \Longleftrightarrow milk$两个方面提供, 而$yogurt \Longleftrightarrow vegetable$这个方面的规则权值并没有达到预期!

又注意到以下强关联规则:

Rule: ('yogurt', 'root vegetables') ==> ('whole milk',) , 0.563, 也就是说, 更多的人在买了yogurt和root vegetables这个组合之后, 购买了whole milk, 然而root vegetables 和 other vegetable的相关性是很强的, 这里的差异就说明问题不是出在'yogurt', 'other vegetables'这个组合上.

3.5.4 总结

从以上三个方面可以说明: Grocery可以更好地引导顾客购买'yogurt', 'other vegetables'这个组合, 这就证明了之前的猜想是正确的.

4 本次Assignment总结

通过这次作业, 我更加深入地理解了Apriori和FP-growth算法, 大二孩子看源码很痛苦, 而且FP-growth程序是GitHub上面star最多的项目, 用类封装了, 很多复杂的优化很难看懂! 但是还是坚持了下来, 很开心有了能力的提升.

再者我挖掘了一个有趣的规则, 根据以前看Kaggle上的kernel和以往知识总结我构建了一套量化分析的方法(不懂对不对(笑哭)), 还是很高兴地很享受这个过程! 这次作业花了很多很多时间, 很辛苦但是也很欣慰!