Mining Practice: Association rule mining

171860607 白晋斌
810594956@qq.com

Table of Contents

You may choose to:

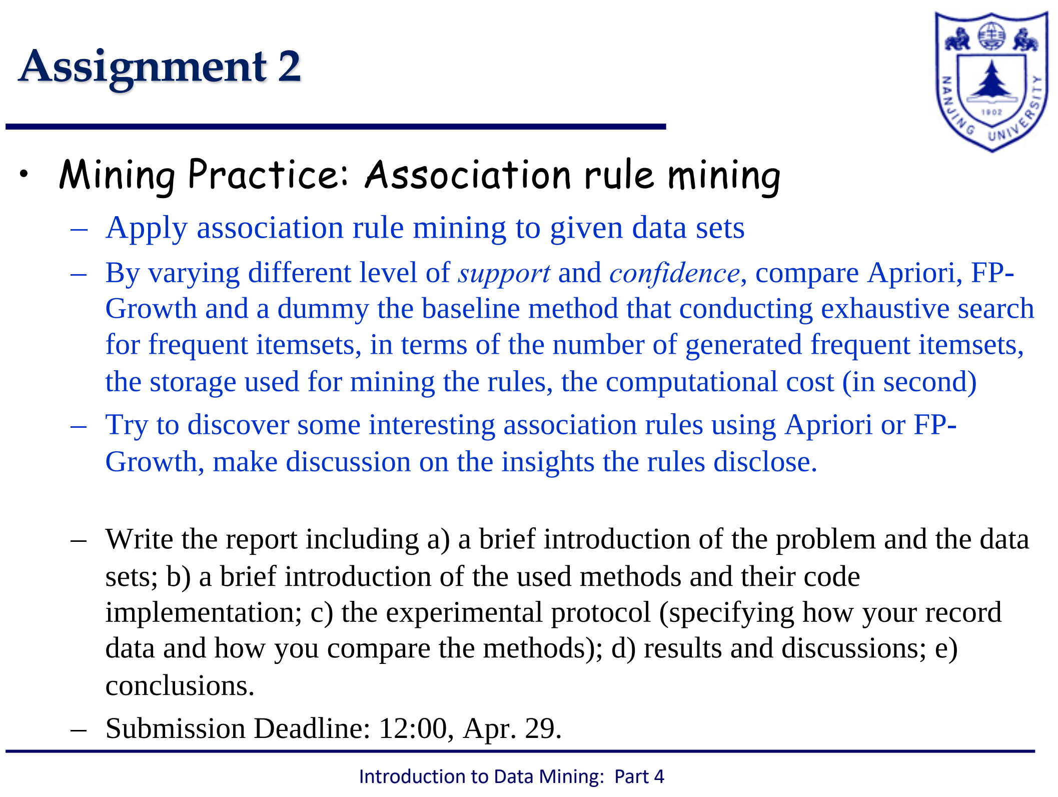
1) implement the dummy baseline, Apriori, FP-Growth by your own (encouraged)
2) use existing codes from packages on the web (please indicate this in your report), in this case. There are a number online resources that implement Apriori and FP-Growth. E.g., the WEKA package introduced in our reference book.

* I. H. Witten, E. Frank. Data Mining: Practical Machine Learning Tools and Techniques, 2nd edition. Morgan Kaufmann Publishers, 2005

I choose (1) which is to implement the dummy baseline, Apriori, FP-Growth by my own.

introduction of the problem and the data sets

1. introduction of the problem



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. 29.

Introduction to Data Mining: Part 4

2. the data sets

2.1 GroceryStore

This data set contains transaction records of a grocery store in a month. Each line is a transaction, where the purchased items line in {}, separated by "," (the space is replaced by "/"). In total, there are 9835 transactions of 169 items.

2.2 UNIX_usage

The data set contains 9 sets of sanitized user data drawn from the command histories of 8 UNIX computer users at Purdue over the course of up to 2 years (USER0 and USER1 were generated by the same person, working on different platforms and different projects). The data is drawn from tcsh(1) history files and has been parsed and sanitized to remove filenames, user names, directory structures, web addresses, host names, and other possibly identifying items. Command names, flags, and shell metacharacters have been preserved. Additionally, **SOF** and **EOF** tokens have been inserted at the start and end of shell sessions, respectively. Sessions are concatenated by date order and tokens appear in the order issued within the shell session, but no timestamps are included in this data.

For example, the two sessions:

```
# Start session 1
cd ~/private/docs
ls -laF | more
cat foo.txt bar.txt zorch.txt > somewhere
```

```
exit
# End session 1

# Start session 2
cd ~/games/
xquake &
fg
vi scores.txt
mailx john_doe@somewhere.com
exit
# End session 2
```

would be represented by the token stream

```
**SOF**
cd
<1>                     # one "file name" argument
ls
-laF
|
more
cat
<3>                     # three "file" arguments
>
<1>
exit
**EOF**
**SOF**
cd
<1>
xquake
&
fg
vi
<1>
mailx
<1>
exit
**EOF**
```

introduction of the used methods and code implementation

0. data structure

每一个 item 为一个元素,由多个 item 组成的每一事务为一个 set,由多个事务组成的数据集为一个 list.

1. methods

1.1 baseline

实现的是一个未经剪枝的 Apriori 算法,基本思想是:

[1] 根据给定的置信度计算最低阈值

[2] 遍历输入数据中的每一事务,得到所有 item 的出现次数,保留出现次数大于等于最低阈值的 item,得到 1-频繁项集

[3] 对 1-频繁项集生成 2-候选频繁项集

[3] 遍历输入数据得到 2-候选频繁项集的频数,保留出现次数大于等于最低阈值的 item,得到 2-频繁项集

[5] 重复第三、四步操作,依次得到 k-频繁项集,直到 k-频繁项集为空或达到预定 k 值.

[6] 基于每轮得到的 k-频繁项集,与其父辈 k-1-频繁项集作差构造关联规则,筛选出大于等于置信度且提升度大于 1.0 的关联规则作为挖掘出的强关联规则.

1.2 apriori

实现的是一个经过剪枝的 Apriori 算法,基本思想是:

[1] 根据给定的置信度计算最低阈值

[2] 遍历输入数据中的每一事务,得到所有 item 的出现次数,保留出现次数大于等于最低阈值的 item,得到 1-频繁项集

[3] 对 1-频繁项集生成 2-候选频繁项集

[4] 剪枝

[5] 遍历输入数据得到 2-候选频繁项集的频数,保留出现次数大于等于最低阈值的 item,得到 2-频繁项集

[6] 重复第三、四、五步操作,依次得到 k-频繁项集,直到 k-频繁项集为空或达到预定 k 值.

[7] 基于每轮得到的 k-频繁项集,与其父辈 k-1-频繁项集作差构造关联规则,筛选出大于等于置信度且提升度大于 1.0 的关联规则作为挖掘出的强关联规则.

其中剪枝的技巧包括:

[1] 减少事务表规模

即首先直接删去输入数据中 item 个数小于 k(第一次执行 k=2)的事务;并且在每轮遍历事务表时统计每个事务被匹配的次数,若低于 k(第一次执行 k=2)次,删除该事务.

[2] 减小生成候选频繁项集的规模

即对 k-候选频繁项集的任意项,遍历其 k-1 子项,判断该 k-1 子项是否为频繁项集,若不是,直接在 k-候选频繁项集中删去该项.

1.3 fpgrowth

[1] 遍历输入数据,找到 1-频繁项集并按频数降序排序得到 F-list,根据 F-list 再次遍历输入数据,得到 FP-tree.

[2] 对于 F-list 上的每个频繁项集, 按照倒序构造其 conditional pattern-base 和 conditional FP-tree

[3] 根据 conditional FP-tree 生成频繁项集.

[4] 基于得到的 k-频繁项集,与其父辈 k-1-频繁项集作差构造关联规则,筛选出大于等于置信度且提升度大于 1.0 的关联规则作为挖掘出的强关联规则.

2. code

2.1 baseline

```
Procedure baseline (input:database of transactions DB; min_sup; min_conf)
     // 找出频繁 1 项集
     L1 =find_frequent_1-itemsets(DB);
     for(k=1;L_k!=null;k++){
        // 产生候选，并剪枝
        C_{k+1} =apriori_gen(L_k,min_sup);
        // 扫描 D 进行候选计数
        for each transaction t  in DB{
            C_t =subset(C_{k+1},t); // 得到 t 的子集
            for each candidate c in C_t
                c.count++;
        }
      //返回候选项集中不小于最小支持度的项集
        L_{k+1} ={c in C_{k+1} | c.count>=min_sup}
      //生成关联规则
        rules.append(rules_gen(C_{k+1},C_k,min_conf)
}
return L= Union(L_k)；rules

Procedure apriori_gen (L_k :frequent k-itemsets;min_sup)
      for each itemset l_j in L_k
         for each itemset l_j in L_k
            if((I_i[1]=I_j[1])&&( I_i[2]=I_j[2])&& ……&& (I_i[k-1]=I_j[k-1])&&(l_i[k]<l_j[k]))
            then{
                    c = I_i join I_j    // 连接步：产生候选
                add c to C_{k+1};
                   }
          return C_{k+1};

Procedure rules_gen(C_{k+1},C_k ,min_conf)
    for(c_i in C_{k+1}){
         for(c_j in C_{k}){
              //构造关联规则,筛选出大于置信度且提升度大于 1 的强关联规则
               if (c_j in c_i and freq(c_j and c_i)/freq(c_i)>=min_conf and freq(c_j and c_i)/freq(c_i)>1.0*freq(c_i)/len(DB))
                    rules.append(c_j ->c_i,freq(c_j and c_i)/freq(c_i))
        }}
return rules

```

2.2 apriori

```
Procedure apriori (input:database of transactions DB; min_sup; min_conf)
     // 找出频繁 1 项集
     L1 =find_frequent_1-itemsets(DB);
     for(k=1;L_k  !=null;k++){
        // 产生候选，并剪枝
        C_{k+1} =apriori_gen(L_k,min_sup);
        // 扫描 D 进行候选计数
        for each transaction t  in DB{
            C_t =subset(C_{k+1},t); // 得到 t 的子集
            for each candidate c in C_t
                c.count++;
        }
    //返回候选项集中不小于最小支持度的项集
        L_{k+1} ={c in C_{k+1} | c.count>=min_sup}
    //生成关联规则
        rules.append(rules_gen(C_{k+1},C_k,min_conf)
}
return L= Union(L_k) ; rules

Procedure apriori_gen (L_k :frequent k-itemsets;min_sup)
      for each itemset l_j in L_k
         for each itemset l_j in L_k
            if( (I_i[1]=I_j[1])&&( I_i[2]=I_j[2])&& ……&& (I_i[k-1]=I_j[k-1])&&(l_i[k]<l_j[k]))
            then{
                    c = I_i join I_j     // 连接步：产生候选
                 //若 k-1 项集中已经存在子集 c 则进行剪枝
                    if has_infrequent_subset(c, L_k ) then
                       delete c; // 剪枝步：删除非频繁候选
                    else add c to C_{k+1};
                   }
          return C_{k+1};

Procedure has_infrequent_subset(c:candidate (k+1)-itemset; L_k:frequent k-itemsets)
         for each k-subset s of c
            if s not in L_k then
               return true;
        return false;

Procedure rules_gen(C_{k+1},C_k ,min_conf)
    for(c_i in C_{k+1}){
         for(c_j in C_{k}){
              //构造关联规则,筛选出大于置信度且提升度大于 1 的强关联规则
               if (c_j in c_i and freq(c_j and c_i)/freq(c_i)>=min_conf and freq(c_j and c_i)/freq(c_i)>1.0*freq(c_i)/len(DB))
                    rules.append(c_j ->c_i,freq(c_j and c_i)/freq(c_i))
        }}
return rules

```

2.3 fpgrowth

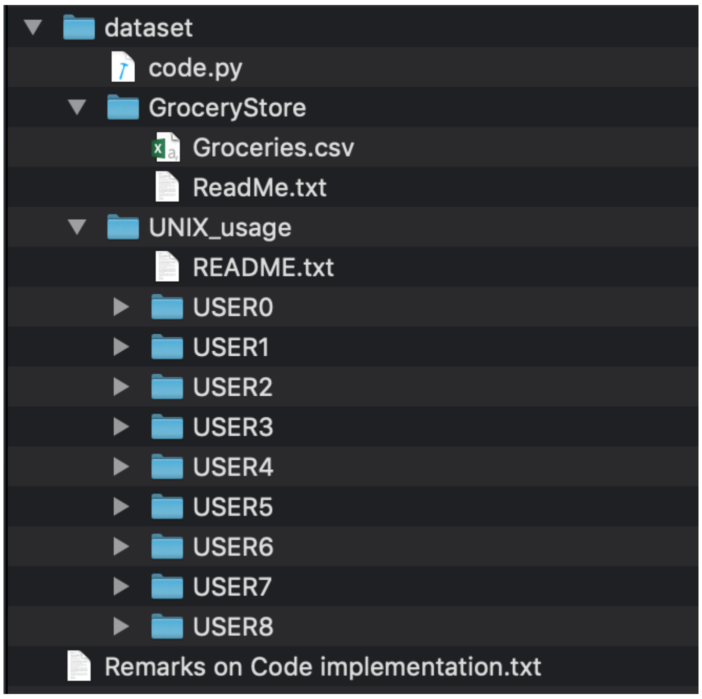
```
Procedure fp_growth(input:database of transactions DB; min_sup; min_conf)
    //得到 1-频繁项集
    F-list=scan(DB).sort().apply(item.freq>min_sup);
    FP_tree.init();
    for item in F-list:
        scan_and_construct_FP_tree(item,DB);
    F-list.sort(Reverse=True);//对 F-list 逆序
    for item in F-list:
        cond_pb.append(conditional_pattern_base(item,FP_tree));
        cond_fpt.append(conditional_FP_tree(item,FP_tree));
        if FP_tree is NULL or FP_tree.path==1:
            break;
    C={}
    for ptr in cond_fpt:
        //通过 freq_pattern_set(T) = freq_pattern_set(P)*freq_pattern_set(Q)生成频繁项集
        C.append(generate_frequent_patterns(ptr));
    //生成关联规则
    rules.append(rules_gen(C_{k+1},C_k,min_conf)

Procedure rules_gen(C_{k+1},C_k ,min_conf)
    for(c_i in C_{k+1}){
        for(c_j in C_{k}){
            //构造关联规则,筛选出大于置信度且提升度大于 1 的强关联规则
            if (c_j in c_i and freq(c_j and c_i)/freq(c_i)>=min_conf and freq(c_j and c_i)/freq(c_i)>1.0*freq(c_i)/len(DB))
                rules.append(c_j ->c_i,freq(c_j and c_i)/freq(c_i))
        }}
return rules
```

3. implementation

在 main 函数中,对两类数据的读取分别被封装为 groceries_data()和 unix_data(),三种算法的实现分别被封装为 dummy()、apriori()、fpgrowth(),对三种算法分别预设了 minimum_support=0.01,minimum_confidence=0.5,itemset_size=10,association_rules_open=True,save_in_files=False 的参数,分别代表支持度、置信度、最大频繁项集所含 item 个数(如果存在的话)、是否同时计算关联规则、是否将结果写入文件.

选择对应的函数进行组合,并选择适当参数即可运行代码.

值得注意的是,code.py 需要放置于 dataset 文件夹下,即文件目录满足以下条件,程序方可正常运行.



experimental protocol

# specifying how your record data and how you compare the methods

我们通过在代码中插入 time()记录时间点,作差得到时间差,为减小系统误差,我们对每项测试运行三次取平均值.对形如 1-item,2-item,3-item,...这样的表格,因难以确定切割点,每一项所计算的时间包含前面的时间长,即累加.对于 fpgrowth 算法则不存在累加情况.

最后我们将数据记录成表格对比分析不同情况下各算法的运行效率.

result and discussions

1.result

1.1 apriori on Groceries(min_sup=0.01)

仅选取部分结果展示(如需完整结果,运行代码文件中的 groceries_data()函数和 apripri()函数,并设置参数 save_in_files=True 即可)

```
bottled water,other vegetables,whole milk : 106
butter,other vegetables,whole milk : 113
citrus fruit,other vegetables,root vegetables : 102
citrus fruit,other vegetables,whole milk : 128
citrus fruit,whole milk,yogurt : 101
curd,whole milk,yogurt : 99
domestic eggs,other vegetables,whole milk : 121
fruit/vegetable juice,other vegetables,whole milk : 103
other vegetables,pastry,whole milk : 104
other vegetables,pip fruit,whole milk : 133
...
(butter,other vegetables) -> (whole milk)       confidence:0.5736040609137056
(citrus fruit,root vegetables) -> (other vegetables)     confidence:0.5862068965517241
(curd,yogurt) -> (whole milk)   confidence:0.5823529411764706
(domestic eggs,other vegetables) -> (whole milk)       confidence:0.5525114155251142
(other vegetables,pip fruit) -> (whole milk)    confidence:0.5175097276264592
(rolls/buns,root vegetables) -> (other vegetables)      confidence:0.502092050209205
(root vegetables,tropical fruit) -> (other vegetables) confidence:0.5845410628019324
(root vegetables,yogurt) -> (other vegetables)          confidence:0.5
(other vegetables,whipped/sour cream) -> (whole milk)
confidence:0.5070422535211268
(other vegetables,yogurt) -> (whole milk)       confidence:0.5128805620608899
(rolls/buns,root vegetables) -> (whole milk)   confidence:0.5230125523012552
(root vegetables,tropical fruit) -> (whole milk)          confidence:0.5700483091787439
(root vegetables,yogurt) -> (whole milk)        confidence:0.562992125984252
(tropical fruit,yogurt) -> (whole milk) confidence:0.5173611111111112
(whipped/sour cream,yogurt) -> (whole milk)          confidence:0.5245098039215687
```

1.2 apriori on UNIX (min_sup=0.01)

仅选取部分结果展示(如需完整结果,运行代码文件中的 unix_data()函数和 apripri()函数,并设置参数 save_in_files=True 即可)

```
-l,<1>,<2>,cd,cp,rm,vi : 114
-l,<1>,<2>,cd,ls,mv,rm : 125
-l,<1>,<2>,cd,ls,mv,vi : 120
-l,<1>,<2>,cd,ls,rm,vi : 149
-l,<1>,<2>,cd,mv,rm,vi : 119
<1>,<2>,>,cd,ll,rm,vi : 115
<1>,<2>,cat,cd,ll,rm,vi : 112
<1>,<2>,cd,cp,exit,ls,vi : 114
<1>,<2>,cd,cp,ll,mv,rm : 117
<1>,<2>,cd,cp,ll,rm,vi : 149
...
```

```
(<2>,cd,cp,ls,mv,rm,vi) -> (<1>)        confidence:1.0
(<1>,cd,cp,ls,mv,rm,vi) -> (<2>)        confidence:1.0
(cd,cp,ls,mv,rm,vi) -> (<1>,<2>)        confidence:1.0
(<1>,<2>,cp,ls,mv,rm,vi) -> (cd)        confidence:0.9856115107913669
(<2>,cp,ls,mv,rm,vi) -> (<1>,cd)        confidence:0.9856115107913669
(<1>,cp,ls,mv,rm,vi) -> (<2>,cd)        confidence:0.9856115107913669
(cp,ls,mv,rm,vi) -> (<1>,<2>,cd)        confidence:0.9856115107913669
...
```

2.time

因实验数据规模较小,以下数据均通过 3 次运行取平均值,但仍有较小浮动.单位:秒.

2.1 including generate association rules

data=groceries_data,minimum_support=0.01,minimum_confidence=0.5

	1-item	2-item	3-item
baseline	0.013	5.271	6.045
apriori	0.012	4.915	5.443
fpgrowth	\	\	0.826

data=unix_data,minimum_support=0.01,minimum_confidence=0.5

	1-item	2-item	3-item	4-item	5-item	6-item	7-item	8-item
baseline	0.013	6.322	15.842	24.785	29.486	30.793	30.964	30.971
apriori	0.014	5.913	12.524	17.680	19.705	20.115	20.150	20.151
fpgrowth	\	\	\	\	\	\	\	13.991

2.2 not including generate association rules

data=groceries_data,minimum_support=0.01,minimum_confidence=0.5

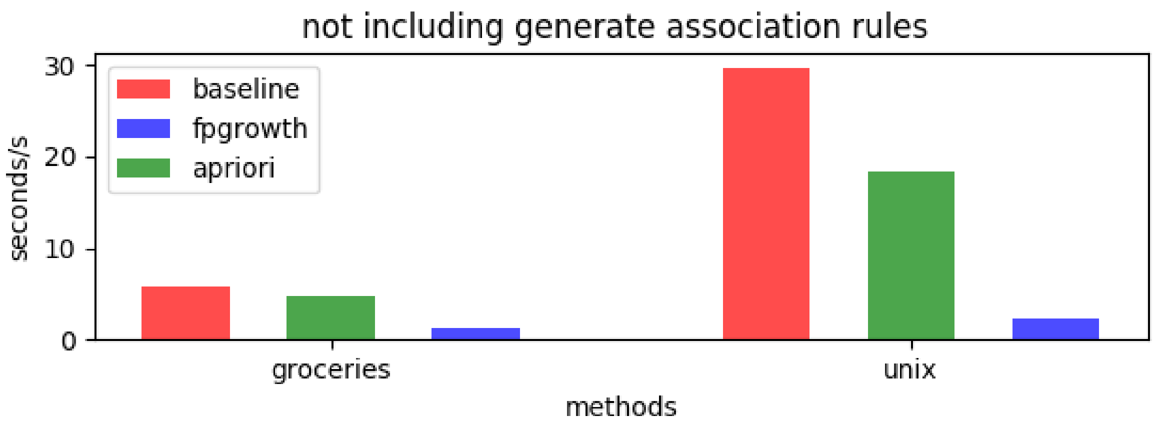
	1-item	2-item	3-item
baseline	0.013	4.925	5.808
apriori	0.012	4.153	4.721
fpgrowth	\	\	1.300

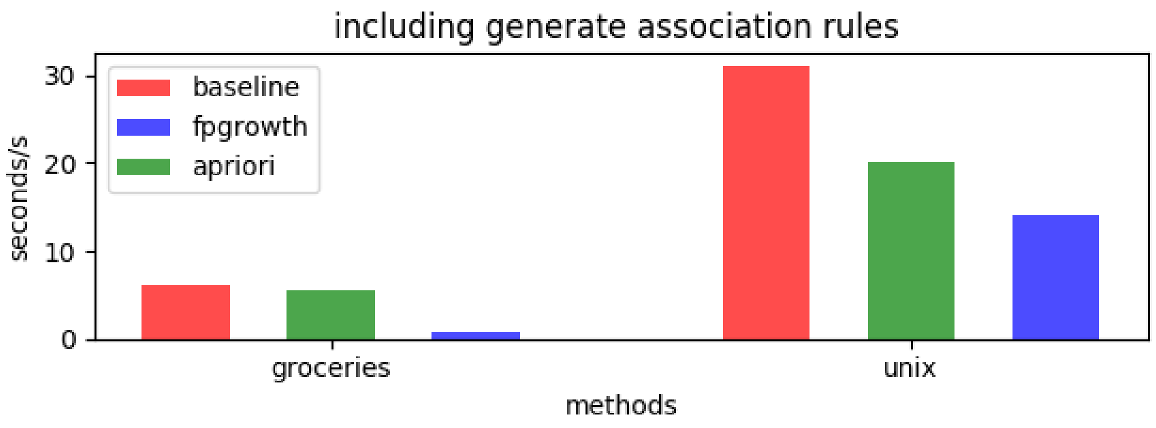
data=unix_data,minimum_support=0.01,minimum_confidence=0.5

	1-item	2-item	3-item	4-item	5-item	6-item	7-item	8-item
baseline	0.013	5.875	15.079	23.631	28.146	29.474	29.686	29.707
apriori	0.014	5.487	11.201	15.831	17.689	18.128	18.210	18.226
fpgrowth	\	\	\	\	\	\	\	2.269

2.3 plot

基于上述表格,在 minimum_support=0.01,minimum_confidence=0.5 的条件下绘制下述柱状图.





3.discussion

从表中可以看出,对于未剪枝算法和 apriori 算法,生成关联规则所需时间很少,而对于 fpgrowth 算法,要花大量时间生成关联规则.

此外,就三种算法消耗时间的对比,我们可以发现 apriori 算法相较于未剪枝算法可以快 10%左右,然而,对于 fpgrowth 算法,在不求关联规则的前提下,可以达到 apriori 算法十倍的速度,即使求关联规则,也能达到 apriori 算法 2-3 倍的运行速度.

conclusions

1.groceries

以 minimum_support=0.01,minimum_confidence=0.5 为例,可以挖掘出 88 个频繁 1-项集, 221 个频繁 2-项集, 32 个频繁 3-项集共 341 个频繁项集.以及 15 条关联规则.其中关联规则如下.

关联规则	置信度
['root vegetables', 'citrus fruit']=>['other vegetables']	0.586
['root vegetables', 'tropical fruit']=>['other vegetables']	0.585
['yogurt', 'curd']=>['whole milk']	0.582
['butter', 'other vegetables']=>['whole milk']	0.574
['root vegetables', 'tropical fruit']=>['whole milk']	0.570
['yogurt', 'root vegetables']=>['whole milk']	0.563
['domestic eggs', 'other vegetables']=>['whole milk']	0.553
['yogurt', 'whipped/sour cream']=>['whole milk']	0.525

['root vegetables', 'rolls/buns']=>['whole milk']	0.523
['pip fruit', 'other vegetables']=>['whole milk']	0.518
['yogurt', 'tropical fruit']=>['whole milk']	0.517
['yogurt', 'other vegetables']=>['whole milk']	0.513
['other vegetables', 'whipped/sour cream']=>['whole milk']	0.507
['root vegetables', 'rolls/buns']=>['other vegetables']	0.502
['yogurt', 'root vegetables']=>['other vegetables']	0.500

可以发现,当购买蔬菜、鸡蛋等食物时,顾客有超过一半的概率会购买牛奶.同时我们也可以根据顾客已购买的食物,预测顾客是否会购买蔬菜或是牛奶等其他食物.

2.unix_data

以 minimum_support=0.01,minimum_confidence=0.5 为例,可以挖掘出从频繁 1-项集到 频繁共 4279 个频繁项集.以及 22645 条关联规则,即使是我们将置信度设置为 1,依旧存在 2819 条关联规则.

当我们将 minimum_support 设置为 0.10 时,共 34 个频繁项集与 65 条关联规则,分析置信度较高的关联规则我们发现,当出现 cd、vi、rm 等指令时,几乎可以确定一定会出现<1>指令,反过来,当出现<1><2>和 ls 等指令时,仅有较大概率(约 90%)出现 cd.