Homework #3: Frequent Patter Mining

Textbook (3rd Edition!)

- 6.1, 6.3, 6.4, 6.5, 6.6, 6.11

Aradhya Mathur

6.1 Suppose you have the set C of all frequent closed itemsets on a data set D, as well as the support count for each frequent closed itemset. Describe an algorithm to determine whether a given itemset X is frequent or not, and the support of X if it is frequent.

Answer)

Apriori Algorithm can be used.

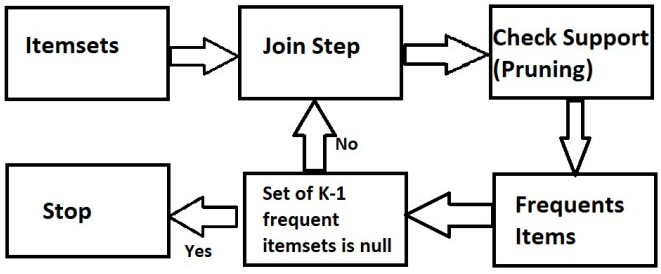
Given: Set C of all frequent closed itemsets
Data set D
Support count for each frequent closed itemset

To find: Whether a given itemset X is frequent or not and support of X if it is frequent

In Apriori we do self-joining and pruning to generate candidates. Using pruning we find redundant and infrequent itemsets.

for X: // Checking for X
$support_x$ = count(X);
if $support_x \geq min\_support$
then print (X,$support_x$)
for i in X': // X' is the subset of X
$support_i$ = count(X'$_i$);
if $support_i \geq min\_support$
then print (i,$support_i$)
i++
else
print(i "X' Not Frequent")
else
print("X Not Frequent")

A set of items is mentioned as X-itemset if it has exactly X distinct things. If the support count for an itemset is greater than the minimum support count, then we refer itemset as frequent itemset.



6.3 The Apriori algorithm makes use of *prior knowledge* of subset support properties.

(a) Prove that all nonempty subsets of a frequent itemset must also be frequent.

(b) Prove that the support of any nonempty subset s' of itemset s must be at least as great as the support of s.

(c) Given frequent itemset l and subset s of l, prove that the confidence of the rule "$s' \Rightarrow (l - s')$" cannot be more than the confidence of "$s \Rightarrow (l - s)$," where s' is a subset of s.

(d) A *partitioning* variation of Apriori subdivides the transactions of a database D into n nonoverlapping partitions. Prove that any itemset that is frequent in D must be frequent in at least one partition of D.

a) To prove: All nonempty subsets of a frequent itemset must also be frequent

D – Database
count – number of transactions in D
min_support – Minimum Support
s – frequent itemset
s' is nonempty subset of s
l - frequent itemset.
sup - Support
As s is a frequent itemset we can say: sup_count(s) = min_support x count
Now, transaction that contains an itemset s will also contain its subset that is itemset s′.
Which means sup_count(s') ≥ sup_count(s) = min_sup × count.
And hence, s' which is nonempty subset of a frequent itemset s is also frequent.

b) To prove: the support of any nonempty subset s′ of itemset s must be as great as the support of s.
D – Database
count – number of transactions in D
min_support – Minimum Support
s – frequent itemset
s' is nonempty subset of s
l - frequent itemset.
sup - Support
We know that: sup(s) = sup_count(s)/ count
Similarly sup(s') =sup_count(s')/count
Above we proved that the sup(s′) ≥ sup(s) which proves the fact that the support of any nonempty subset s′ of itemset s must be as great as the support of s.

c) To prove:
Given frequent itemset l and subset s of l, prove that the confidence of the rule "$s' \Rightarrow (l - s')$" cannot be more than the confidence of "$s \Rightarrow (l - s)$," where s' is a subset of s.

D – Database
count – number of transactions in D

min_support – Minimum Support
s – frequent itemset
s' is nonempty subset of s
l - frequent itemset.
sup - Support

Given is s subset of l
So, confidence(s => (l-s)) = sup(l) / sup(s)
Similarly for s',
confidence(s' => (l-s')) = sup(l) / sup(s')

$sup(s') \geq sup(s)$ is proved above, and we know that confidence $\propto 1/sup(s)$
Which means that confidence(s'=> (l-s')) = sup(l)/sup(s') ≤ confidence(s => (l-s)) = sup(l)/sup(s)

d) C =>frequent itemset.
D =>a set of database transactions.
L => total number of transactions in D.
X => total number of transactions in D containing the itemset C.
m_supp => minimum support.
Let n=3 for ease (nonoverlapping partitions)

C is a frequent itemset so, $X \geq L \times m\_supp$
Now partition D into d1, d2, d3.
Similarly, partition total number of transactions into l1, l2, l3 such that L = l1+ l2+l3.
Similarly, partition X into x1, x2, x3 such that X = x1 + x2 + x3.

$X = L \times m\_supp$ can be written from $(x1 + x2 + x3) = (l1 + l2 + l3) \times m\_supp$.

Let us assume that C is not frequent in any of the partitions of D.
Which further means that $x1 < l1 \times m\_supp$; $x2 < l2 \times m\_supp$; $x3 < l3 \times m\_supp$.
We add above equations to obtain $(x1+x2+x3+) < (l1 + l2 + l3) \times m\_supp$
Now x1+x2+x3=X and l1+l2+l3=L which means X/L < m_supp.
This contradicts the fact that C was defined as a frequent itemset, and our assumption is wrong.

If this is the case for n=3, it will be valid for n nonoverlapping partitions.

Therefore, any itemset that is frequent in D must be frequent in at least one partition of D.

6.4 Let c be a candidate itemset in C_k generated by the Apriori algorithm. How many length-$(k-1)$ subsets do we need to check in the prune step? Per your previous answer, can you give an improved version of procedure has_infrequent_subset in Figure 6.4?

Answer)

We need to check k-2 subsets only. C is the candidate itemset which is generated from self-join of two frequent itemsets, each of length k-1. When we do this, we need not generate min_sup of these two subsets and checking k-2 subsets would provide insights about any infrequent itemsets.

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ φ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
                  ∧ ... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

To improvise has_frequent_subset we need to reduce searching by not running l1 and l2 as thet are already used while doing the joining step and we have established that they are frequent. We can bypass checking of l1 and l2 and directly send them to reduce computation.

6.5 Section 6.2.2 describes a method for *generating association rules* from frequent itemsets. Propose a more efficient method. Explain why it is more efficient than the one proposed there. (*Hint:* Consider incorporating the properties of Exercises 6.3(b), (c) into your design.)

Answer)

To find: An efficient method to generate association rules from frequent itemsets.

The method proposed in 6.2.2 is inefficient because it generates unrequired subsets. After generating these subsets, it tests each subset to find association rules. This process can be easily simplified.

A more efficient can be proposed in which only required subsets are generated. To find required subset we can find minimum confidence for s' of length i. If it does not that satisfy the criteria, we don't need to generate for subsets of s' as seen in part b and c of 6.3. And if the min confidence criteria are satisfied then we generate (i − 1) subsets of s'. Algorithm below shows the process we discussed earlier:

i = length(s)
find (i − 1) subsets
for each (i − 1) subset s'
if (sup_count(l) / sup_count(s') ≥ m_conf)
then rule "s'⇒(l – s')"
else return -1}

If we use the efficient method, we iteratively move from k-1 subsets of k itemsets to 1 subset which is very efficient unlike the method proposed in 6.2.2. That method generates all the nonempty subsets of a frequent including the unrequired ones which do not satisfy minimum confidence.

Let's take an example, in which there are i-itemset named s and none of s's (i − 1)-subsets satisfy min_confidence. Using the efficient method, we would only need to find s's(i-1) subsets and after finding that min_confidence is not met, we can reduce the process by not generating more subsets. While the traditional method would find all the s's nonempty subsets and find rules for each of them and increase significant unrequired computation.

6.6 A database has five transactions. Let *min sup* D 60% and *min conf* D 80%.

TID	*items_bought*
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y }
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

(a) Find all frequent itemsets using Apriori and FP-growth, respectively. Compare the efficiency of the two mining processes.

(b) List all the *strong* association rules (with support s and confidence c) matching the following metarule, where X is a variable representing customers, and $item_i$ denotes variables representing items (e.g., "A," "B,"):

$$\forall x \in transaction,\ buys(X, item_1) \wedge buys(X, item_2) \Rightarrow buys(X, item_3) \quad [s, c]$$

Support = 5*60/100=3 (Total transactions = 5 and min sup = 60%)

Min conf = 80%

a)

APRIORI

DATABASE

TID	Item-Bought
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

C1

Itemset	Support
{A}	1
{C}	2
{D}	1
{E}	4
{I}	1
{K}	5
{M}	3
{N}	2
{O}	3
{U}	1
{Y}	3

Remove itemset with Support< Min_Support

L1

Itemset	Support
{E}	4
{K}	5
{M}	3
{O}	3
{Y}	3

C2

Itemset	Support
{E,K}	4
{E,M}	2
{E,O}	3
{E,Y}	2
{K,M}	3
{K,O}	3
{K,Y}	3
{M,O}	1
{M,Y}	2
{O,Y}	2

Remove itemset with Support< Min_Support

L2

Itemset	Support
{E,K}	4
{E,O}	3
{K,M}	3
{K,O}	3
{K,Y}	3

C3

Itemset	Support
{E,K,O}	3
{E,K,M}	2
{E,K,Y}	2
{K,M,O}	1
{K,M,Y}	2
{K,O,Y}	2

Remove itemset with Support< Min_Support

L3

Itemset	Support
{E,K,O}	3

Complete set for Apriori:

{{E}, {K}, {M}, {O}, {Y}, {E,K}, {E,O}, {K,M}, {K,O}, {K,Y}, {E,K,O} }

FP Growth

DATABASE

TID	Item-Bought
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

Itemset	Frequency
{A}	1
{C}	2
{D}	1
{E}	4
{I}	1
{K}	5
{M}	3
{N}	2
{O}	3
{U}	1
{Y}	3

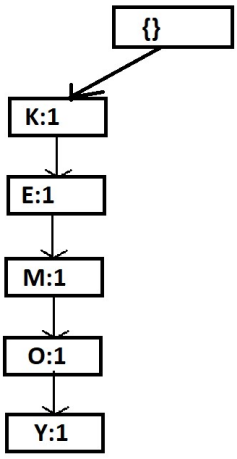
Remove itemset with Support< Min_Support

Sort based on frequency

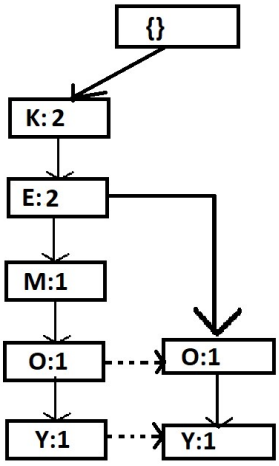
Itemset	Support
{K}	5
{E}	4
{M}	3
{O}	3
{Y}	3

TID	ITEMS_BOUGHT	ORDERED_FREQUENCY_ITEMS
T100	{M, O, N, K, E, Y}	{K,E,M,O,Y}
T200	{D, O, N, K, E, Y}	{K,E,O,Y}
T300	{M, A, K, E}	{K,E,M}
T400	{M, U, C, K, Y}	{K,M,Y}
T500	{C, O, O, K, I, E}	{K,E,O}

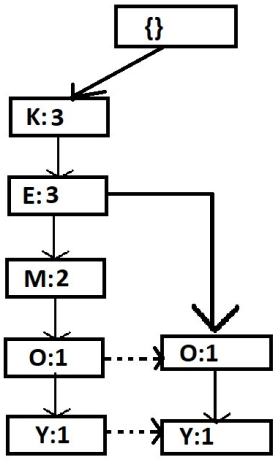
Step 1:



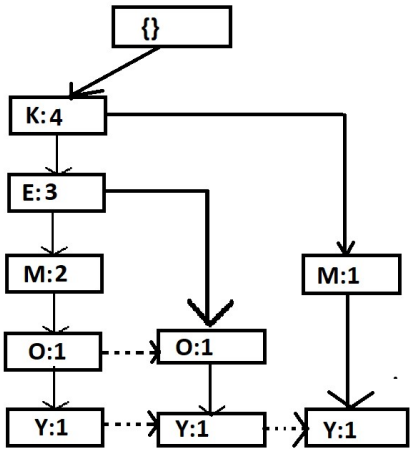
Step 2:



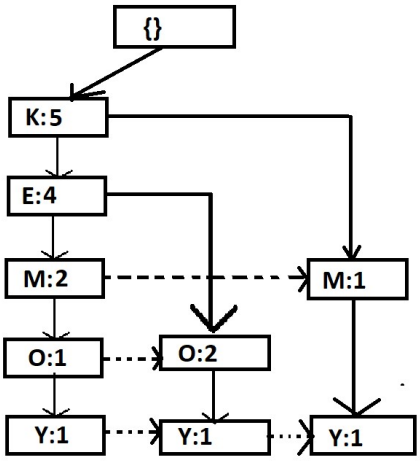
Step 3:



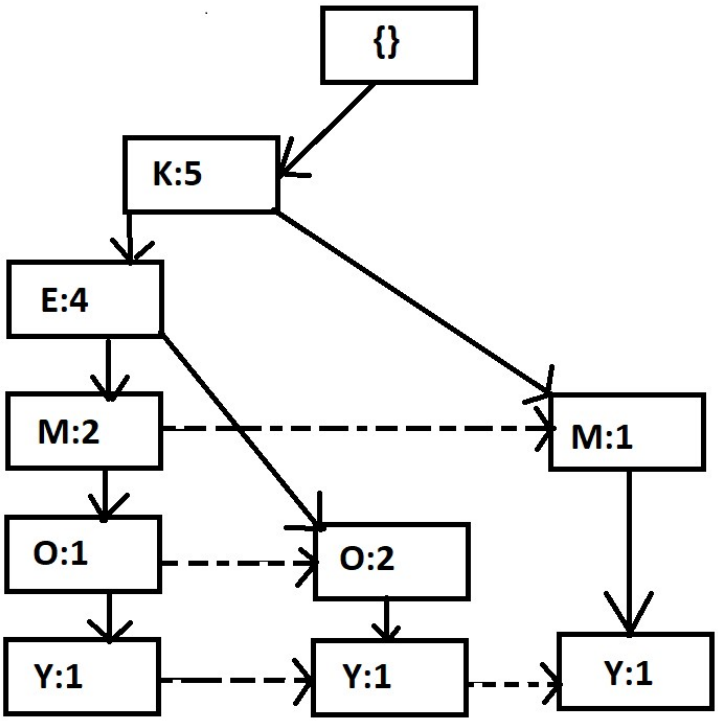
Step 4:



Step 5:



Final FP Growth Tree



Items	Conditional Pattern Base
Y	{K,E,M,O:1}, {K,E,O:1}, {K,M:1}
O	{K,E,M:1} {K,E:2}
M	{K,E:2} {K:1}
E	{K:4}
K	{}

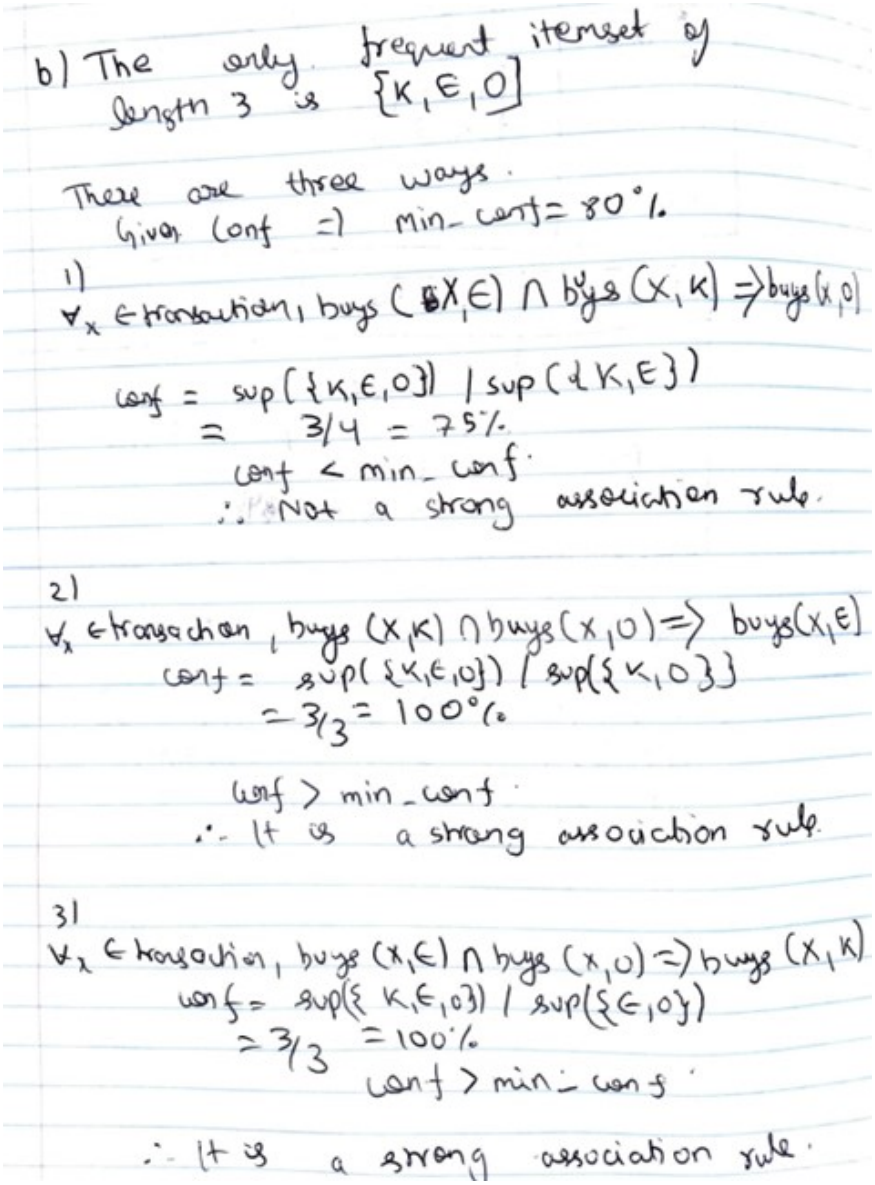
Items	Conditional Frequent Pattern Base
Y	{K:3}
O	{K:3, E:3}
M	{K3}
E	{K:4}
K	{}

Items	Frequent Patterns
Y	{K,Y:3}
O	{K,O:3} {E,O:3} {K,E,O:3}
M	{K,M:3}
E	{K,E:4}
K	{}

Complete Set for FP Growth:

{ {K: 5}, {E: 4}, {M: 3}, {O: 3}, {Y: 3}, {K,Y: 3}, {K,O: 3}, {E,O: 3}, {K,E,O: 3}, {K,M:3}, {K,E: 4} }

Because of small dataset Apriori was more efficient here as it gave the frequent itemset in less time. But as the dataset increases FP growth tree will be more efficient as Apriori scans the whole database again and again. Ability of FP growth to generate conditional pattern bases helps in reducing search time as the data it also reduced. In general, FP growth is more efficient than Apriori.



K,O - > E (60%,100%)

O,E -> K (60%,100%)

6.11 Most frequent pattern mining algorithms consider only distinct items in a transaction. However, multiple occurrences of an item in the same shopping basket, such as four cakes and three jugs of milk, can be important in transactional data analysis. How can one mine frequent itemsets efficiently considering multiple occurrences of items? Propose modifications to the well-known algorithms, such as Apriori and FP-growth, to adapt to such a situation.

Answer)
It is very easy to accommodate the multiple occurrences of an item in Apriori and FP Growth algorithms.
Apriori: In this algorithm we need to consider multiple occurrences of items like in D100: Bread, Bread, Butter -> Bread – count 1 and Bread - count 2. Here, we consider both breads as different items and count them as 2 rather than 1. Now we store the count values for each itemset and check minimum support. If minimum support is met, we move forward and follow next steps associated with Apriori. Count is very important in checking whether itemset is frequent or not.

TID	*items_bought*
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y }
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

In this support of {O} will be 4 as there are 2 O's in T500.

Similarly, in FPgrowth algorithm. D100: Bread, Bread, Butter -> Bread – count 1 and Bread - count 2. While generating frequency of items we consider the frequency of Bread 2 rather than 1. When we form tables, we need to consider the updated frequency of items and rest of the procedure remains same.

TID	*items_bought*
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y }
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

In this frequency of {O} will be 4 as there are 2 O's in T500. And the step 1 in which we sort on the basis of frequency will be changed to{K-5,E-4,O-4,M-3,Y-3} considering min_support=60%.

Sort based on frequency

Itemset	Support
{K}	5
{E}	4
{O}	4
{M}	3
{Y}	3

TID	ITEMS_BOUGHT	ORDERED_FREQUENCY_ITEMS
T100	{M, O, N, K, E, Y}	{K,E,M,O,Y}
T200	{D, O, N, K, E, Y}	{K,E,O,Y}
T300	{M, A, K, E}	{K,E,M}
T400	{M, U, C, K, Y}	{K,M,Y}
T500	{C, O, O, K, I, E}	{K,E,O1,O2}

After than we follow the same procedure. To make both O distinct we name it O1 and O2.