**CSC240 – Data Mining**

**Student Name: Hizkia Febianto**

**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.**

Objective: Determine whether a given itemset X is frequent or not; calculate the support if it is frequent

Input:

* *C*, a set of frequent closed itemsets
* *min\_sup*, the minimum support count threshold

Output:

* *bool*: {frequent; infrequent}
* *sup\_count*

Algorithm:

1. Initiate a variable *temp* to store an itemset and another variable *bool* to store a boolean value that indicates whether *X* is frequent or not.
2. Iterate through *C* and find each itemset that is a superset of X and get its support count. Store the first itemset to *temp* and compare the rest of the itemsets with *temp*. Find the shortest itemset which has *X* as its subset.
3. Return *bool* and the support count of *X*. The support count of *X* is the same as the support count of the shortest itemset that has *X* as its subset in *C*.

Pseudocode:

1. temp = [ ] //a temporary variable to store an array of itemsets
2. bool = “infrequent”
3. **for each** itemset c **in** *C*
4. **if** (X c) **then**
5. **if** temp == [ ] **then**
6. temp = c;
7. bool = “frequent”;
8. **else**
9. **if** length(temp) > length(c) **then**
10. temp = c**;**
11. bool = “frequent”;
13. **if** temp = [ ] **then**
14. **return** bool
15. **else**
16. **return** bool, support\_count(temp)

Philosophy behind the algorithm:

We can determine whether itemset *X* is frequent and calculate its support by iterating through the database *D* but this method requires a lot of computation. Since *C* lists all frequent closed itemsets on *D,* iterating through *C* will provide the same answers but with less computation. For an itemset *X* to be frequent, it has to be a subset of at least one itemset in *C*.

The support of *X* is the same as the support of the shortest itemset which has *X* as its subset. Suppose the iteration though *C* results in two itemsets with different lengths, the shorter itemset must be a subset of the longer itemset since both of them are closed. If this is not the case, then there must be a shorter closed itemset which has *X* as its subset. The shortest possible itemset is *X* itself.

**6.3.**

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

1. **Prove that all nonempty subsets of a frequent itemset must also be frequent.**

Suppose we have a frequent itemset *I* and a subset of *I,* which is denoted by *s*. Since *I* is a frequent itemset, it must have a support value greater than *min\_sup*, hence P(*I*)  *min\_sup.* The subset *s* is also an itemset that has a support value of P(*s*) in the *D* database. We need to prove that P(*s*) is also greater than or equal to *min­\_sup*.

We know that: P(*I*) *min\_sup*.

The occurrence or support value of *s* can be calculated using Bayes’ rule.

P(*s*) = P(*s*|*I*)P(*I*) + P(*s*|*I’*)P(*I*’)

Since *s* *I*, P(*s*|*I*) will always be equal to 1. We also know that P(*s*|*I*’) will be greater than or equal to zero and P(*I*’) will never be negative. Therefore, we know that P(s) will always be equal to or greater than P(*I*) since P(*s*) = P(*I*) + P(*s*|*I*’)P(*I*’). The following relationship will prove that *s* is also frequent if *I* is frequent.

P(*s*) P(*I*) *min\_sup*

1. **Prove that the support of any nonempty subset of itemset s must be at least as great as the support of s.**

Following the same logic as in part a, we will prove that P(*s*’) is always greater than or equal to P(*s*). From its definition, a support of an itemset *I* is equal to the probability of *I*, P(*I*). Therefore, the support of itemset s is P(*s*) and the support of s’ is P(*s*’). Using Bayes’ rule, the following equation can be derived.

P(*s*’) = P(*s*’|*s*)P(s) + P(*s*’|*s*c)P(*s*c)

Since *s*’ is a subset of *s*, the probability of *s*’ given *s* is always equal to 1. The equation above can be written as P(*s*’) = P(*s*) + P(*s*’|*s*c)P(*s*c). Since the value of P(*s*’|*s*c) and P(*s*c) will be between 0 and 1, the value of P(*s*’) must be greater than or equal to P(*s*). Hence, support of *s*’ must be greater than or equal to support of *s*.

1. **Given frequent itemset l and subset s of l, prove that the confidence of the rule cannot be more than the confidence of where is a subset of s.**

The confidence of the rule is calculated as and the confidence of the rule is calculated as . The numerator in both formulas has the same value and can be simplified as . In part b, we have proved that the support of any nonempty subset s’ of itemset s must be at least as great as the support of s. Since the numerators are the same, the rule will have a smaller confidence than the rule since the support value of s’ is bigger than the support value of s.

1. **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.**

Proof by contradiction: Let us assume that an itemset *I* is not frequent in any of the partitions of *D* but it is frequent in *D*. Let *a* be the total number of transactions in *D* and let *c* be the total number of transactions in *D* that contain the itemset *I*. Because *I* is frequent, .

Now let us partition *D* into *n* non-overlapping partitions, *d*1, *d*2, *d*3, …, *d*n. Let *a*1, *a*2, *a*3, …, *a*n be the total number of transactions in partitions *d*1, *d*2, *d*3, …, *d*n. Let *c*1, *c*2, *c*3, …, *c*n­ be the total number of transactions in *d*1, *d*2, *d*3, …, *d*n that contain itemset *I*.

The previous equation can be rewritten as (*c*1 + *c*2 + *c*3 + … + *c*n) (*a*1 + *a*2 + *a*3 + … + *a*n) *min\_sup.* If *I* is not frequent in any partitions of *D*, then *c*1 *a*1 *min\_sup*; *c*2 *a*2 *min\_sup*; … ; *c*n *a*n *min\_sup*. Adding these inequalities, we get (*c*1 + *c*2 + *c*3 + … + *c*n) (*a*1 + *a*2 + *a*3 + … + *a*n)*min\_sup* which contradicts the previous inequality.

**6.4.**

**Let c be a candidate itemset in Ck 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?**

In the prune step, we need to check subsets. The improved version of procedure *has\_infrequent­­\_subset* is as follows.

procedure has\_infrequent\_subset(c, Lk-1)

for each itemset a in c

for each (k-1)-subset s of a

if s Lk-1 then

return TRUE;

else return FALSE;

**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.)**

Given a frequent itemset *I*, the method described in Section 6.2.2 generates all nonempty subsets of *I* and then test them for potential rules. This method is inefficient because it also generates and tests unnecessary subsets. In Exercise 6.3(b) and 6.3(c), we proved that the confidence of the rule generated by a subset of length *k* will always be greater than or equal to the confidences of all potential rules generated by any subsets with length less than *k*. Hence, it is unnecessary to check any nonempty subsets of *x* if its confidence does not meet the minimum confidence. The more efficient method is as follows.

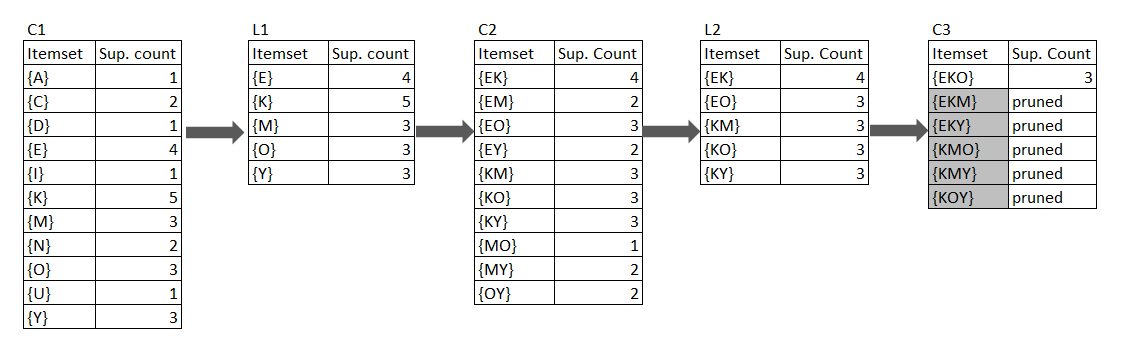
Suppose *I* has a length of *n*. We start by listing all possible subsets of *I* that have a length of (n-1). After calculating the confidence of each subset, we eliminate subsets that do not meet the minimum confidence. For (n-1)-subsets that meet the minimum confidence, we then construct all possible (n-2)-subsets, calculate their confidences, and eliminate all subsets that do not meet the minimum confidence. We work our way down to the 1-subsets or to the point where there is no subset that meets the minimum confidence.

**6.6. A database has five transactions. Let *min\_sup = 60*% and ­*min\_conf* = 80%.**

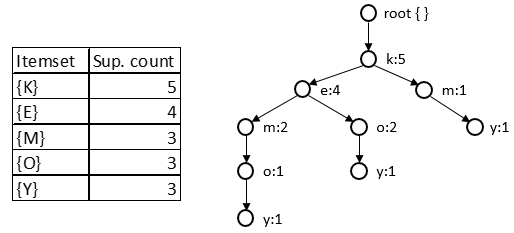
| **TID** | **items\_bought** |
| --- | --- |
| T100 | {M, O, N, K, E, Y} |
| T200 | {D, O, N, K, E, Y } |
| T300 | {M, A, K, E} |
| T400 | {M, U, C, K, Y} |
| T500 | {C, O, O, K, I, E} |

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

Apriori:



FP-Growth:





Apriori algorithm iterates through the database many times while FP-growth algorithm does it only once. If the number of itemsets in database is very large, Apriori algorithm will be very slow compared to FP-growth algorithm. In this case, FP-growth gives more possible itemsets than apriori.

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

The list of strong association rules that match the metarule is as follows.

{OK}{E}, support = 60%, confidence = 100%

{OE}{K}, support = 60%, confidence = 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.**

We can consider the item and its occurrence as a combined item. If there are 4 cakes in a transaction, we can represent this item using a tuple as (*cake,4*). If another transaction has 2 cakes, we represent this as a different item (*cake*,2). Using Apriori and FP-growth, we can find frequent itemsets of this transaction database.