Where did the candidates go?

May 7, 2015

1 Introduction

Frequent pattern mining is a classical data-mining task and plays an essential role in mining associations, correlations, sequential patterns etc [1]. Here, we focus on basket data which is a database of transactions each containing a set of database items I = $\{i_1, i_2, ..., i_n\}$. Frequent patterns are those item sets whose number of occurrences is above a certain threshold. Maximal frequent patterns are those item sets which cannot be extended to a larger maximal frequent pattern by adding items. A very popular algorithm for this task is **Apriori** [2]. The algorithm finds frequent patterns iteratively in breath-first order. That is, it finds all frequent patterns of length (k - 1) before those of length k. At a high level the algorithm has three main steps at the k^{th} iteration:

1. Generate candidate frequent patterns of length k (C_k) from those of length (k -1).

2. Check which of these candidates are really frequent by examining the database transactions.

3. Obtain the patterns of length k (L_k) which are really frequent by comparing against a minimum support threshold.

The candidates at the first step are produced through the Apriori property [3]: *A subset of a frequent item set must also be a frequent item set.* However,

the second step can be very expensive with respect to time and multiple passes over the database may be necessary.

In [4] Zaki et. al. proposes new algorithms for fast association mining which scan the database only once. Here they propose to find *potential maximal frequent item sets* (candidates) from smaller frequent item sets by finding cliques in a hyper graph whose vertices are these smaller frequent item sets. They partition these smaller frequent item sets L_k of size k into equivalence classes based on their common k - 1 length prefix. Consider the equivalence classes generated from frequent item sets of size k = 2 below: (reproduced from [4])

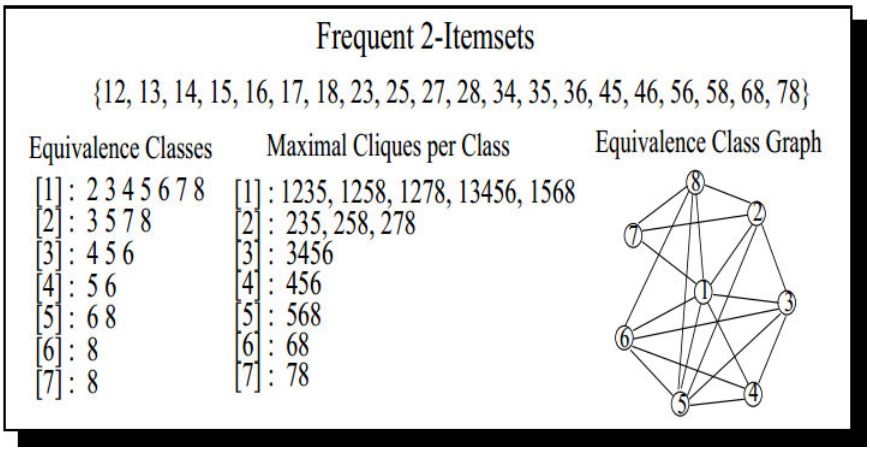

Figure 1: Equivalence classes

The equivalence class with prefix 1 i.e. 12345678 is a potential maximal item set (as each subset is frequent). Also, all subsets of length 3,..,7 in the subset lattice of 12345678 are candidate frequent item sets. They propose a hybrid top-down/bottom-up traversal algorithm on the candidate subset lattices to determine the true frequent item sets.

2 Mining Frequent Patterns without candidate Generation

The authors of [1] argue that the approach of Aripori and other similar methods like [4], which is *candidate set generation and test* (as described above)

is computationally very expensive. They propose that the performance of frequent pattern mining can be substantially improved by not generating candidates patterns.

They propose to do this using a novel data structure called frequent pattern tree or FP-tree. Each vertex in this tree corresponds to an item in the database and has a count value. This tree is grown in two steps:

1. Scan the database once to find the frequent 1-item sets F and sort F in decreasing support as L.

2. Create a root node called "null". For each each transaction (Trans) in the database sort the frequent items in order of L. Now, insert this transaction in the FP-tree. This starts from the "null" node and the first item in Trans. If the node being examined has a child corresponding to the next item in Trans, increment the count of that node by 1. Otherwise create a new node corresponding to the next item in Trans with count 1. This process is continued until nodes corresponding to each frequent item in Trans have been found/created . All nodes corresponding to a particular item are connected by *node-links*.

The FP-tree is compact representation of the database which allows frequent pattern mining. The diagram below (reproduced from [1]) illustrates this mining process:

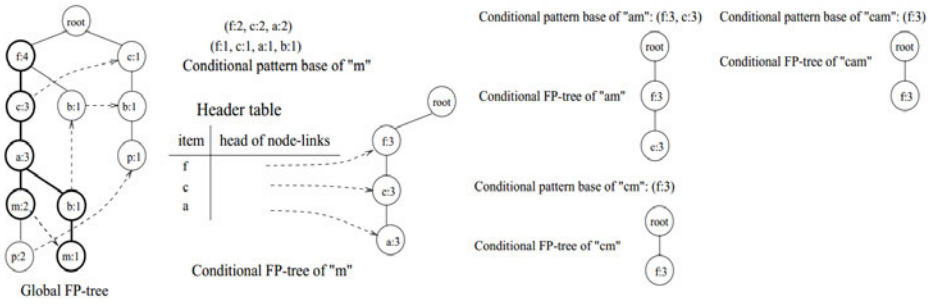

Figure 2: FP-tree Mining

The mining process first considers each item in the order of their support. Following node links, a conditional FP-tree is created which implicitly contains information only about transactions which contain the item under

consideration. The conditional FP-tree of the item **m** is shown. The counts on a node of item **m** tells us how many transactions contain **m** and all items in the path from "null" to the node of **m**. We can modify the counts on the other nodes accordingly (Property 3.2 from [1]). Now, the count values on the other nodes (say **a**) measure how many times that item co-occurs with **m**. Here, **am** occurs 3 times together and can be classified as frequent.

Let's compare Figure 1 and 2. The nodes **macf** in Figure 2 mirrors the equivalence classes in [4] and in Figure 1.

After all pairs which co-occurs with **m** have been determined longer frequent patterns are obtained in a depth first style. As **ma** is frequent we search for longer frequent patterns containing **ma**. In fact the pattern **cam** is found. A further recursive call finds **fcam**. After this, recursive calls to find longer patterns containing **cm** and **fm** are executed.

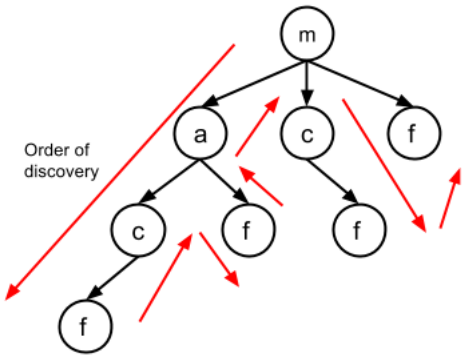

Figure 3: Order of FP-tree Mining

The key point here is: **The items in the conditional FP-tree of m i.e. macf can be considered as a candidate maximal frequent item set and all its subsets (of size 3) candidate frequent item sets**. This means that FP-Growth algorithm in [1] implicitly constructs candidate frequent item sets.

The gain in performance comes from having a compact representation of the database which makes the task of discovering larger frequent item sets from smaller ones computationally inexpensive (equivalent to finding frequent 1-item sets in the FP-tree).

3 Is FP-Growth is really different than Tree-Projection?

The authors of [1] argue that the Tree-Projection algorithm in [5] is really different from FP-Growth in [1]. This claim is valid as there are significant differences in the way they operate.

Tree-Projection works in a breath-first order in contrast to the depth-first order of FP-Growth. So, considering the example from the previous section first the frequent patterns **ma, mc, mf** will be generated before **mac, macf** etc. However the search tree will appear very similar to Figure 2 except that the order of discovery will be breath first. (A combined depth-first/breath-first search strategy for Tree-Projection is possible).

Moreover, and more importantly, the FP-Tree is designed in such a way that we can determine the frequency of larger item sets from smaller (frequent) item sets using Property 3.2 in [1]. The Tree-Projection algorithm operates by constructing matrices which contain support of item sets of size two at level (k-1) to generate nodes at level (k+1). In other words, this amounts to finding frequent 1-item sets in FP-Growth and frequent 2-item sets in Tree-Projection. If the number of transactions is large and length of each transaction is long this can be very expensive.

Thus, due to the above mentioned facts (and other unmentioned facts) the authors of [1] claim that FP-Growth is more efficient and scalable than Tree-Projection.

References

[1] Han, J., Pei, J. , Yin, Y. , *Mining frequent patterns without candidate generation*, In Proceedings of the ACM International Conference on Management of Data (SIGMOD), Dallas, TX, pages 1-12, ACM, 2000.

[2] Rakesh Agrawal, Ramakrishnan Srikant, *Fast Algorithms for Mining Association Rules*, Proceedings of the 20th VLDB Conference Santiago, Chile, 1994.

[3] Professor Anita Wasilewska, *APRIORI Algorithm*, Lecture Notes.

[4] Zaki, M.J. , Parthasarathy, S. , Ogihara, M. , Li, W. , *New algorithms for fast discovery of association rules*, In Proceedings of the 3rd ACM Inter-

national Conference on Knowledge Discovery and Data Mining (KDD), Newport Beach, CA, 1997.

[5] Agarwal, R.C., Aggarwal, C.C. , Prasad V. , *A tree projection algorithm for generation of frequent item sets*, J. Parallel Distr. Com., 61(3):350-371, 2001.