CSE 5095: Research Topics in Big Data Analytics
Lecture 18 (April 01, 2014)

Recap of the last class:

- In the last lecture, we learnt about two methods for generating candidates for frequent itemsets. Once we generate the candidates, we have to compute the support for each candidate. We also introduced hash tree for efficiently checking candidates.

Expected time for candidates checking:

It takes $O(k^2|C_k|)$ time.

Computing support of candidates in C_k:

A simple algorithm takes $O(Nk\,|C_k|)$ time.

Another way:

For each transaction, increment the support by 1 of each candidate that is contained in the transaction. Use a hash tree to store all the candidates in C_k.

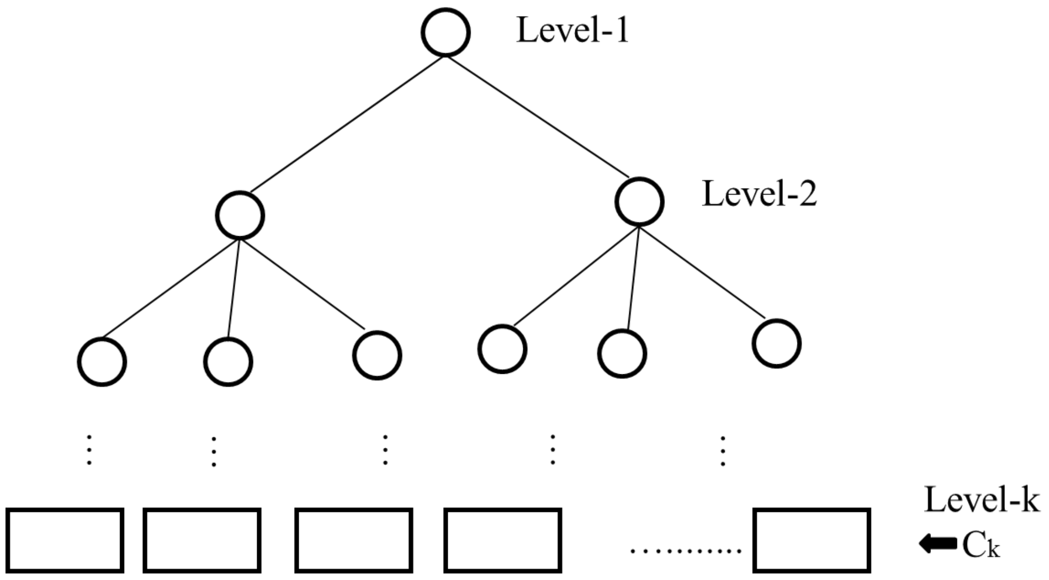


Figure 1: A hash tree is used to store all the candidates in C_k.

```
For each transaction $T$ do
    For each $k-$ subset $Q$ of $T$ do
        Hash $Q$ into the hash tree. Once a leaf is reached,
        For each candidate in the leaf
            Increment the support by 1 if the candidate is the same as $Q$.
```

<u>**Expected Run Time**</u> $= O\left(\binom{|T|}{k}.\, k\right)$ where k denotes the level of the hash tree. The size of each candidate is also *k.*

<u>**Rules Generation:**</u>

- Let X be any frequent itemset. We generate rules from X as follows:
    - Consider all non-empty and proper subsets Y of X.
    - Compute the confidence of the rule $Y \rightarrow X - Y$

<u>**Note:**</u> Confidence for this rule is $\frac{\sigma(X)}{\sigma(Y)}$

- If $Y' \subset Y$ then $\sigma(Y') \geq \sigma(Y)$.

Therefore, $(the\ confidence\ of\ Y \rightarrow X - Y) \geq (the\ confidence\ of\ Y' \rightarrow X - Y')$

<u>**Idea:**</u> Use the above observation and a level wise approach. At level k generate rules with k items in the consequent, starting from $k = 1, 2, \cdots$

<u>**Example:**</u>

If the rule $\{abc\} \rightarrow \{d\}$ does not have enough confidence then we can ignore the following rules: $\{ab\} \rightarrow \{cd\}$, $\{bc\} \rightarrow \{ad\}$ and so on.

<u>**Note:**</u> We can generate candidates at each level just like in the case of frequent itemsets generation.

$\{ab\} \rightarrow \{cd\}$ is a candidate if $\{abd\} \rightarrow \{c\}$, $\{abc\} \rightarrow \{d\}$ have enough confidence.

■ **Apriori uses horizontal layout of the data:** We keep the transactions.

■ **Vertical Layout:** For every item keep a list of transactions in which the item occurs.

A randomized rules mining algorithm (Toivanan 1996)

Basic idea:

- Let DB be the given database
- Let $minSupport$ be the target support
- Pick a random sample S from DB
- Identify itemsets from S with a support of $\geq ms$ where $ms < minSupport$
- Let Q be the collection of frequent itemsets in S
- For each itemset $q \in Q$ calculate its support in DB by examining the rest of the DB.
- Output each $q \in Q$ whose support in DB is $\geq minSupport$.

Note: If every frequent itemset in DB is also frequent in *S,* then we are in good shape. If *ms* is small enough, then this will happen with a high probability. However, is there any way to make sure that we have captured all the frequent itemsets of DB in *S?* The notion of *negative border* will be helpful in this direction.

Definition:

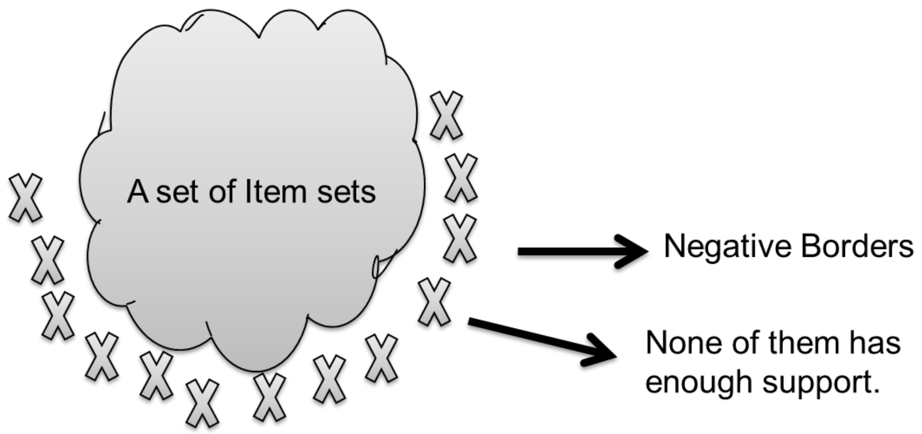
- Let S be a collection of itemsets.
- Then the negative border of S, denoted as $NB(S)$, is defined as the collection of itemsets that are not in S, but each subset of them is in S.

Example:

$I = \{a, b, c, d, e, f\}$

$S = \{\{ac\}, \{abd\}, \{de\}, \{abe\}\}$

$NB(S) = \{\{f\}, \{bc\}, \{cd\}, \{ce\}\}$



Note:

Let S be a set of frequent itemsets. If none of the sets in $NB(S)$ has a support of $\geq minSupport$ then S is a superset of **all the** frequent itemsets in DB.

Modified Algorithm:

- Pick a random sample S from the DB.
- Identify itemsets from S with a support of $\geq ms$, where $ms < minSupport$
- Let Q be the collection of frequent itemsets in S.
- Let $Q^{`} = NB(Q)$
- For each $q \in Q \cup NB(Q)$ do
    - Compute its support in DB examining the rest of the database.
- If no set in $NB(Q)^{`}$ has a support of $\geq minSupport$ then output all the sets in Q whose support in DB is $\geq minSupport$.
- If not, use another 2 phase algorithm.

Analysis: To be continued…