CSE5095 Research Topics in Big Data Analytics

Instructor: Professor Sanguthevar Rajasekaran
Note taker: Marius Nicolae

Mar 27, 2014

1 Association Rules Mining

Definition. An **itemset** is a set of items. A **k-itemset** is an itemset of size k.

Definition. A **transaction** is an itemset.

Definition. A **rule** is represented as $X \to Y$ where $X \neq \emptyset, Y \neq \emptyset, X \cap Y = \emptyset$.

From here on, assume that we are given a database DB of transactions and the number of transactions in the database is n. Let I be the set of distinct items in the database and let $d = |I|$.

Definition. For an itemset X, we define $\sigma(X)$ as the number of transactions in which X occurs, i.e. $\sigma(X) = |\{T \in DB | X \subseteq T\}|$

Definition. The **support** of any rule $X \to Y$ is $\frac{\sigma(X \cup Y)}{n}$.

Definition. The **confidence** of any rule $X \to Y$ is $\frac{\sigma(X \cup Y)}{\sigma(X)}$.

Problem. Association Rules Mining
Input: A DB of transactions and two numbers: minSupport and minConfidence.
Output: All rules $X \to Y$ whose support is $\geq$ minSupport and whose confidence is $\geq$ minConfidence.

Definition. An itemset is **frequent** if $\sigma(X) \geq n \cdot \text{minSupport}$

Finding association rules is generally a two step process: 1) identify all the frequent itemsets and 2) for each frequent itemset generate relevant rules. For example, if X is a frequent itemset, then consider rules of the kind: $X \setminus Y \to Y$ for all $Y \neq \emptyset$ (or, equivalently, rules of the form $X_1 \to X_2, X_1 \cup X_2 = X$).

1.1 Identifying frequent itemsets

Idea: Use a level-wise strategy: generate all frequent 1-itemsets, then all frequent 2-itemsets, and so on.

Note: The problem of finding the maximum k such that there exist frequent k-itemsets is NP-hard.

1.1.1 A Brute Force Algorithm

To generate the frequent k-itemsets, a naive algorithm generates all the k-itemsets. For each itemset it scans the database and checks if the itemset is frequent. Assume that we store every transaction T as a bit array of size d where $T[i]$ is 1 if item i is in the transaction and 0 otherwise. Therefore, it takes $O(k)$ time to check if an itemset of size k can be found in a transaction. The running time of the algorithm is then $O(\binom{d}{k} nk)$.

1.1.2 The Apriori Principle:

- If X is not frequent then no superset of X is frequent.
- If X is frequent then every subset of X is also frequent.

Example: Assume minSupport= 1/4 and the database is:

#	Transaction
t_1	Break, Milk, Salt
t_2	Salt, Pepper, IceCream
t_3	Milk, Salt
t_4	Sugar, IceCream, Salt
t_5	Milk, Coffee, Sugar
t_6	IceCream, Salt
t_7	IceCream
t_8	IceCream, Sugar, Salt

Let F_k stand for the set of frequent k-itemsets, for any k. Then we have:
$F_1 = \{(\text{Milk}), (\text{Salt}), (\text{IceCream}), (\text{Sugar})\}$
$F_2 = \{(\text{Milk, Salt}), (\text{Salt, IceCream}), (\text{Salt, Sugar}), (\text{IceCream, Sugar})\}$
$F_3 = \{(\text{Salt, Sugar, IceCream})\}$
$F_4 = \emptyset$

Note that the brute force algorithm will generate 7 1-itemsets followed by $\binom{7}{2}$ 2-itemsets, followed by $\binom{7}{3}$ 3-itemsets, followed by $\binom{7}{4}$ 4-itemsets. In total, the brute force algorithm will generate 98 itemsets. On the other hand, the Apriori algorithm will generate 7 $1-$itemsets, then $\binom{4}{2}$ $2-$itemsets, then $4 \cdot 2 = 8$ $3-$itemsets then a single $4-$itemset, for a total of 22 itemsets.

The pseudocode for the Apriori algorithm is given next.

```
k := 1;
Compute F_1 = {i ∈ I | σ(i) ≥ n · minSupport};
while F_k ≠ ∅ do
    k := k + 1;
    Generate candidates C_k from F_{k-1};
    for T ∈ DB do
        for C ∈ C_k do
            if C ⊆ T then
                σ(C) := σ(C) + 1;
            end
        end
    end
    F_k := ∅;
    for C ∈ C_k do
        if σ(C) ≥ n · minSupport then
            F_k := F_k ∪ {C};
        end
    end
end
```

Algorithm 1: Apriori algorithm

The time to generate F_1 is $O(\sum_{i=1}^{n} |t_i|) = O(nw)$ where w is the maximum length of any transaction.

A heuristic: To avoid generating a candidate many times, we can keep any itemset in increasing order of the items in it. When we generate a new itemset from an existing one, we will only add elements larger than the largest element in the existing itemset.

Questions:
A. How do we generate candidates C_k from F_{k-1}?
B. How do we compute the support of the candidates?

A. Generation of candidates

1. $F_{k-1} \times F_1$ method: To every frequent $k-1$ itemset add every frequent item, to generate candidates.

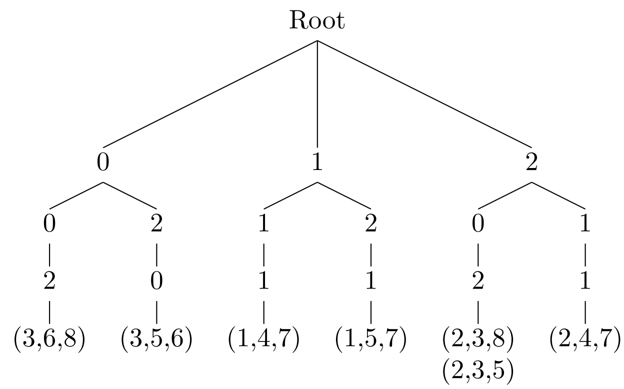
2. $F_{k-1} \times F_{k-1}$ method: Let:
   $a_1, a_2, \ldots, a_{k-2}, a_{k-1}$ and
   $b_1, b_2, \ldots, b_{k-2}, b_{k-1}$ belong to F_{k-1}.

   If $a_i = b_i, \forall i = 1, \ldots, k-2$ then generate candidate $(a_1, a_2, \ldots, a_{k-1}, b_{k-1})$. The time for candidate generation using this method is $O(|F_{k-1}|^2 k)$

Candidate Pruning: We can prune candidates using the Apriori principle, as follows: if C is a candidate in C_k, check if every $k-1$ subset of C is frequent ($\in F_{k-1}$). If not, discard the candidate. To check if a $k-1$ itemset belongs to

F_{k-1} we can use a **Hash Tree**. A Hash Tree is a tree where every node contains a hash table. Itemsets are inserted in the tree based on the hash values of their items. Specifically, at the root, hashing is done on the first item of the itemset hashed. In the next level of the tree, hashing is done on the second item of the itemset, etc. Thus, if the itemsets are of size k, then there will be k levels in the tree.

Hash Tree example: Consider the following itemsets: $(2,3,8)$, $(3,5,6)$, $(1,4,7)$, $(2,3,5)$, $(3,6,8)$, $(1,5,7)$, $(2,4,7)$ and the hash function $h(x) = x \mod 3$. Then the hash tree looks as follows (empty subtrees omitted because of space limitations).



If we build a Hash Tree for F_{k-1} then we can check if an itemset is in F_{k-1} in $O(k)$ time.

An itemset of size k has k different subsets of size $k-1$. We can search each subset in the hash tree in time $O(k)$. Therefore the time for pruning is $O(|C_k|k^2)$.

B. Support counting - Next time.