Data Mining Note

Chauncey Liu

November 13, 2017

Contents

1 Association Rules and Frequent Pattern Mining **2**
1.1 The Frequent Pattern Mining Model . . . 2
1.2 Association Rules . . . 3
1.2.1 Mining Association Rules . . . 3
1.2.2 Itemset Lattice . . . 3
1.2.3 Anti-monotonicity (downward closure) property . . . 3
1.2.4 Computing the Association Rules . . . 3
1.2.5 Interestingness of Association Rules . . . 4
1.3 The Apriori Algorithm . . . 4
1.3.1 Approach . . . 4
1.3.2 Candidate Generation . . . 5
1.3.3 Support Counting . . . 5
1.3.4 Hash Tree . . . 5
1.3.5 Methods of Efficiency Improvement . . . 6
1.4 Enumeration-Tree Algorithms . . . 6
1.5 Suffix-based Pattern Growth Methods . . . 6
1.6 Constraint-Based Association Mining . . . 6
1.7 Multi-level Association Rules . . . 6
1.8 Pattern Summarization . . . 6

1 Association Rules and Frequent Pattern Mining

1.1 The Frequent Pattern Mining Model

Basic Concepts:

- Items $I = \{i_1, i_2, ..., i_m\}$, a set of literals
- Itemset X (transaction): set of items $X \subseteq I$
- Database D: set of transactions T_i where $T_i \subseteq I$
- T contains X: $X \subseteq T$
- Items in transactions or itemsets are sorted in lexicographic order
- Length of itemset: number of elements of itemset
- k-itemset: itemset of length k

Supermarket example:

- Items $I = \{Bread, Butter, Milk, Eggs, Yogurt\}$
- Itemset X, Transaction T:
  - $X_1 = \{Bread, Butter\}$, $X_2 = \{Eggs, Milk\}$
  - $T_1 = \{Bread, Butter, Milk\}$, $T_2 = \{Eggs, Milk, Yogurt\}$
- Database D: set of transactions T_i where $T_i \subseteq I$
  - $T_1 = \{Bread, Butter, Milk\}$, $T_2 = \{Eggs, Milk, Yogurt\}$
  - $D = \{T_1, T_2\} = \{\{Bread, Butter, Milk\}, \{Eggs, Milk, Yogurt\}\}$
- T contains X: $X \subseteq T$
  - $X_1 = \{Bread, Butter\}$, $T_1 = \{Bread, Butter, Milk\}$
  - $X_1 \subseteq T_1$
- Items in transactions or itemsets are sorted in lexicographic order
- Length of itemset: number of elements of itemset
- k-itemset: itemset of length k

Further Concept

- Support of itemset X in D: percentage of transactions in D containing X
  - $sup(X, D) = \frac{|T \in D | X \subseteq T|}{|D|}$
- Frequent itemset X in D: item set X with
  - $freq(X, D) :\Leftrightarrow sup(X, D) \geq minsup$
- Association rule: implication of the form $X \Rightarrow Y$
  - where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \varnothing$
- Support s of association rule $X \Rightarrow Y$ in D:
  - indicates how frequently the itemset appears in the dataset
  - support of $X \cup Y$ in D
  - $s = \frac{|T \in D | (X \cup Y) \subseteq T|}{|D|}$
- Confidence c of association rule $X \Rightarrow Y$ in D:
  - indicates how often the rule has been found to be true
  - percentage of transactions containing Y in the subset of all transactions in D that contain X
  - $c = \frac{|T \in D | (X \cup Y) \subseteq T|}{|T \in D | X \subseteq T|} = \frac{\text{sup}(X \cup Y)}{\text{sup}(X)}$

1.2 Association Rules

1.2.1 Mining Association Rules

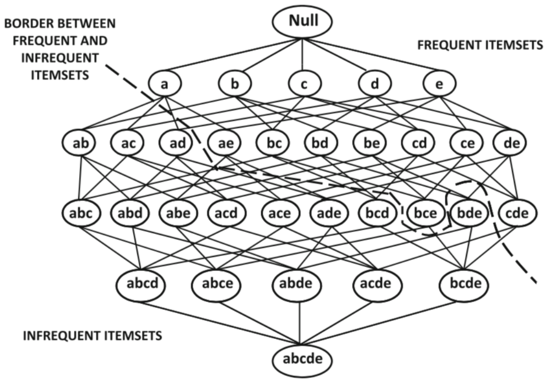
1. Determine the frequent itemsets in the database
Naive algorithm: count the frequencies of all k-itemsets $\subseteq I$, inefficient since $\binom{m}{k}$ such itemsets

2. Generate the association rules from the frequent itemsets
Itemset X frequent and $A \subseteq X$
$A \Rightarrow (X - A)$ satisfies minimum support constraint (confidence remaining to be checked)

1.2.2 Itemset Lattice

- Lattice: a partially ordered set with unique least upper bound and greatest lower bound.
- Itemset lattice:
  - elements: itemsets $X_1 \subseteq I, X_2 \subseteq I, ..., X_n \subseteq I$
  - partial order: $X_1 < X_2 :\Leftrightarrow X_1 \subset X_2$
  - least upper bound: I
  - greatest lower bound: $\varnothing$

Figure 1: Itemset Lattice



1.2.3 Anti-monotonicity (downward closure) property

Each subset of a frequent itemset is also frequent.
$\forall T_1 \subseteq I, T_2 \subseteq I : T_1 \subseteq T_2 \wedge freq(T_2, D) \Rightarrow freq(T_1, D)$
because of
$\forall T_1 \subseteq I, T_2 \subseteq I : T_1 \subseteq T_2 \Rightarrow sup(T_1, D) \geq sup(T_2, D)$

If one subset is not frequent, then superset cannot be frequent.
This property makes frequent itemset mining efficient, since in practice most itemsets are infrequent.

1.2.4 Computing the Association Rules

- Given a frequent itemset X
- For each subset A of X, form the rule $A \Rightarrow (X - A)$
- Compute confidence of the rule $A \Rightarrow (X - A)$
  - $confidence(A \Rightarrow (X - A)) = \frac{sup(X)}{sup(A)}$
- Discard rules that do not have minimum confidence
- Store frequent itemsets with their supports in a hash table
  - no DB acesses, no disk I/O

1.2.5 Interestingness of Association Rules

- Filter out misleading association rules
- Expected support for the rule $A \Rightarrow B$
  - $P(A \cup B) = P(A) \cdot P(B)$
  - assuming the idependence of A and B
- Interestingness measure for rule $A \Rightarrow B$
  - $\frac{P(A \cup B)}{P(A)} - P(B)$
  - The larger this measure, the more interesting the discovered association between A and B
- An alternative interestingness measure is the lift of an association rule
- If A and B are independent, then $P(A \cup B) = P(A) \cdot P(B)$
  - i.e. $\frac{P(A \cup B)}{P(A) \cdot P(B)} = 1$
- We define the lift of a rule $A \Rightarrow B$ as follows:
  - $lift(A \Rightarrow B) = \frac{P(A \cup B)}{P(A) \cdot P(B)}$
  - Can also be formulated as:
    - $lift(A \Rightarrow B) = \frac{P(A \cup B)/P(A)}{P(B)} = \frac{support_{actual}}{support_{expected}}$
    - as the ratio of the conditional probability $P(B|A)$ and the unconditional probability $P(B)$
  - A lift >> 1 indicates that the discovered association between A and B is interesting.

1.3 The Apriori Algorithm

1.3.1 Approach

- Determine first the frequent 1-itemsets, then frequent 2-itemsets, ...
- To determine the frequent k+1-items, consider only the k+1-items for which all k-subsets are frequent
- Calculation of support: one database scan counting the support for all relevant itemsets

Algorithm 1 Algorithm Apriori

```
/* C_k : set of candidate itemsets of length k */
/* F_k : set of all frequent itemsets of length k */

function APRIORI(D, minsup)
    while F_k ≠ ∅ do
        Generate C_{k+1} by joining itemset-pairs in F_k;
        Prune itemsets from C_{k+1} that violate anti-monotonicity;
        Determine F_{k+1} by counting supoort of C_{k+1} in D and retaining itemsets fron C_{k+1} with support at least minsup;
        k = k + 1;
    return ∪_k F_k;
```

1.3.2 Candidate Generation

Requirements for set C_k of candidate itemsets

- Superset of F_k
- Significantly smaller than set of all k-subsets of I

Step1: Join

- Frequent k-1-itemsets p and q, p and q are joined if they aggre in their first $k-2$ items
- E.g. $p \in F_{k-1} = \{1, 2, 3\}$, $q \in F_{k-1} = \{1, 2, 4\} \Rightarrow (1, 2, 3, 4) \in C_k$
- Choose first k-2 items to avoid duplication without missing any candidates

Step2: Pruning

- Remove all elements from C_k having a k-1-subset not contained in F_{k-1}
- E.g. $F_3 = \{(1, 2, 3), (1, 2, 4), (1, 3, 4), (1, 3, 5), (2, 3, 4)\}$
- After join step: $C_4 = \{(1, 2, 3, 4), (1, 3, 4, 5)\}$
- In pruning step: remove $(1, 3, 4, 5)$ since subsets $(1, 4, 5), (3, 4, 5)$ are missing
- $C_4 = \{(1, 2, 3, 4)\}$

1.3.3 Support Counting

for each candidate $c \in c_k$ **do** c.count = 0;
for each transaction $T \in D$ **do**
    $CT := subset(C_k, T)$; // all candidates from C_k that are contained in transaction T
    for each candidate $c \in CT$ **do** c.count++;
$F_k := \{c \in C_k | (c.count/|D|) \geq minsup\}$

To achieve one scan over the database D, $subset(C_k, T)$ should be implemented properly. Thus we need Hash Tree.

1.3.4 Hash Tree

Hash tree as a data stucture for C_k

- Leaf node: records list of itemsets (with frequencies)
- Inner node: contains hash table (apply hash function to d-th elements), each hash bucket at level d references son node at level $d + 1$
- Root has level 1

Finding an itemset

- Start from the root
- At level d: apply hash function h to the d-th element of the itemset.

Inserting an itemset

- Find the corrsponding leaf node and insert new itemset
- In case of overflow:
  - Covert leaf node into inner node and create all its son nodes (new leaves).
  - Distribute all entries over the new leaf nodes according to hash function h.

Find all candidates contained in $T = (t_1 t_2 t_3 ... t_m)$

- At root
  - Determine hash values $h(t_i)$ for each item t_i in T
  - Continue search in all correspoding son nodes
- At inner node of level d
  - Assumption: innder node has been reached by hashing t_i
  - Determine hash values and continue search for all items t_k in T with $k > i$
- At leaf node
  - For each itemset X in this node, test whether $X \subseteq T$

1.3.5 Methods of Efficiency Improvement

- Support counting using a hash table
  - Hash table instad of hash tree, support counters for hash buckets
  - k-itemset with correspoding bucket counter ¡ minsup cannot be frequent
    - more efficient access to candidates but inaccurate counts
- Reduction of transactions
  - Transactions that do not contain any frequenct k-itemset are irrelevant
  - Remove such transactions for future phases
    - more efficient database scan, but additional writing of database
- Partitioning of the database
  - Itemset is only frequent if frequent in at least one partition
  - Form memory-resident partions of the database
    - more efficient on partitions, but expensive combination of intermediate results
- Sampling
  - Apply algorithm to sample to find frequent itemsets
  - Count support of these frequent itemsets in the whole database
  - Determine further candicates and support counting on the whole database

1.4 Enumeration-Tree Algorithms

1.5 Suffix-based Pattern Growth Methods

1.6 Constraint-Based Association Mining

1.7 Multi-level Association Rules

1.8 Pattern Summarization