Data Mining Note

Chauncey Liu

November 15, 2017

Contents

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
  - $c = \frac{|T \in D | (X \cup Y) \subseteq T|}{|T \in D | X \subseteq T|} = \frac{\sup(X \cup Y)}{\sup(X)}$

1.2 Association Rules

1.2.1 Mining Association Rules

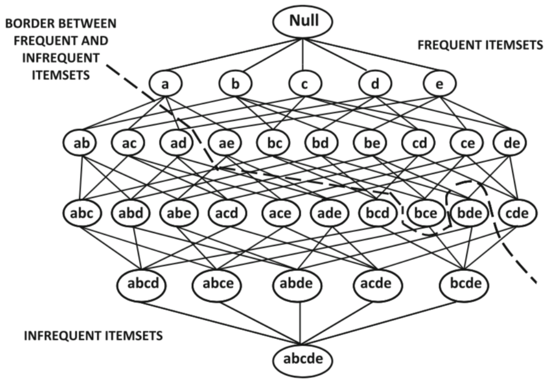
1. Determine the frequent itemsets in the database
Naive algorithm: count the frequencies of all k-itemsets $\subseteq I$, inefficient since $\binom{m}{k}$ such itemsets

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

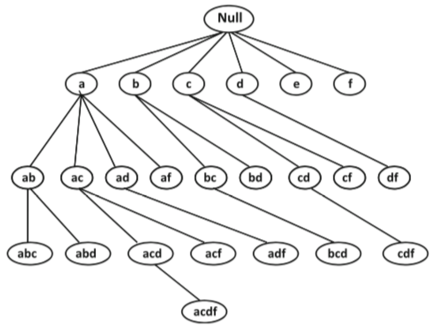
- Support counting using a hash table
  - Hash table instad of hash tree, support counters for hash buckets
  - k-itemset with correspoding bucket counter ¡ minsup cannot be frequent
    - more efficient access to candidates but inaccurate counts
- Reduction of transactions
  - Transactions that do not 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

- Frequent itemsets are stored in a tree-like data structure, the enumeration tree, which provides an abstract, hierarchical representation of the lattice of itemsets.
- Items within aset are ordered lexicographically (Lexicographic tree).
- Hierarchical structure supports systematic and non-redundant exploration of the lattice of itemsets.

- Nodes represent itemsets
- Edges represent subset relationships
- A child node $C = \{i_1, i_2, ..., i_k\}$ extends the parent node $P = \{i_1, i_2, ..., i_k - 1\}$ by one item that is lexicographically larger than all items of the parent node
- The root represents the empty itemset (null)

Figure 2: The lexicographic or enumeration tree of frequent itemsets



Algorithm 2 Generic Enumeration Tree

function GenericEnumerationTree(D, minsup)
  /* Initialize enumeration tree ET to Null Node*/
  while any node in ET has not been examined **do**
    select one or more unexamined nodes P;
    for each p in P **do**
      Generate candidate extensions $C(p)$;
    Count support in D for all n in any $C(p)$;
    if support of n $\geq$ minsup **then**
      extend node p by node n
  return ET;

1.4.1 Example Algorithms

- Apriori
  - candidate generation is level-wise (breadth-first)
  - joining siblings
  - single databse scan to count support of all candidates of a level
- FP-growth
  - candidate generation is depth-first
  - create projected databse of transactions supporting an itemset
  - count support of candidate extensions only in projected database
    * Minimize the number of candidate itemsets generated and counted, without missing any frequent itemsets

1.5 Suffix-based Pattern Growth Methods

1.5.1 Recursive Suffix-based Pattern Growth

- In Apriori, have to count support from scratch at every level
- In order not to waste the computational effort of counting, form projected database for a frequent itemset P: all transactions containing itemset P
- If a transaction does not contain the itemset corresponding to an enumeration-tree node, then this will not be relevant for counting at any descendent (superset itemset) of tht node.
- Count support of extensions of P only in projected database of P.
- Use absolute minsup, not relative minsup.
- Start with empty pattern (suffix) and complete database D, where D has been filtered to contain only frequent items.
- Recursive calls for all extensions and their projected databases.

Algorithm 3 Algorithm Recursive Suffix Growth confusion waiting to be solved

```
/* D: transactions in terms of frequent 1-items, i.e. without infrequent items */
/* P: current suffix itemset */
/* reports all frequent itemsets with suffix P */

function RecursiveSuffixGrowth(D, minsup, P)
    for each item i in D do
        report itemset P_i = {i} ∪ P as frequent;
        Form D_i with all transactions from D containing item i;
        Remove all items from D_i that are lexicographically y ≥ i;
        Rmove all infrequent items from D_i
        if D_i ≠ ∅ then RecursiveSuffixGrowth(D_i, minsup, P_i nb)
```

1.5.2 FP-Tree

- Space-efficient data structure for projected database
- Trie structure represents conditional database by consolidating the prefixes
- Path fron the root to a leaf represents a transaction (or a set of identical transactions)
- Path from the root to internal node represents a prefix of a transaction (or a transaction)
- Each node has count (in the original database) of transactions that support that prefix (or transaction)
- Prefixes are sorted in dictionary order
- Lexicographic ordering of items from most frequent to least frequent
  - Maximizes the effect of prefix-based compression
    - Item with a large support is more likely to be the prefix of many other itemsets
  - Balances the size of different conditional databases

Construction of FP-Tree

- Create an empty tree
- Remove infrequent items from the transactions
- Insert the modified transactions into the tree, one by one
- When the prefix of the transaction overlaps with an existing path, increment the counts of that path by 1
- For the non-overlapping part of the transaction, create new nodes with a count of 1.
- If applicable, create pointer to "next" node with the same item

Extraction of conditional FP-Tree of item i

- Chase pointers for item i to extract the tree of its conditional prefix paths. Prune remaining branches.
- Adjust counts in the prefix paths to account for the pruned branches
- Count frequency of each item by aggregating the counts of that item in the tree of prefix paths. Remove infrequent items. Item i is also removed.
  - conditional FP-tree may have to be re-created by successive insertion of prefix paths

1.6 Constraint-Based Association Mining

1.6.1 Motivation

- Too many frequent itemsets
  - Mining is inefficient
- Too many association rules
  - hard to evaluate
- Constraints may be known apriori
  - Constraints on the frequent itemsets
  - e.g. "association rules on product A but not on product B"
  - e.g. "only association rules with toal price > 100"

1.6.2 Types of constraints

- Domain Constraints
  - $S\theta v, \theta \in \{=, \neq, <, \leq, >, \geq\}$
    - e.g. $S.price < 100$
  - $v\theta S, \theta \in \{\in, \notin\}$
    - e.g. $snack \notin S.type$
  - $V\theta S$ or $S\theta V$, $\theta \in \{\subseteq, \subset, \not\subset, =, \neq\}$
    - e.g. $\{snacks, wines\} \subseteq S.type$
- Aggregation Constraints
  - $agg(S)\theta v$, where
    - $agg \in \{min, max, sum, count, avg\}$
    - $\theta \in \{=, \neq, <, \leq, >, \geq\}$
  - $count(S_1.type) = 1, avg(S_2.price) > 100$

1.6.3 Application of the Constraints

- When determining the association rules
  - Solves the evaluation problem
  - But not the efficiency problem
- When determining the frequent itemsets
  - Can also solve the efficiency problem
  - Challenge for candidate generation

1.6.4 Anti-Monotonicity

- Definition: If an itemset S violates an anti-monotone constraints C, then all supersets of S violate this constraint.
- Examples
  - $sum(S.price) \leq v$ is anti-monotone
  - $sum(S.price) \geq v$ is not anti-monotone
  - $sum(S.price) = v$ is partly anti-monotone
- Application
  - Push anti-monotone constraints into candidate generation

1.7 Multi-level Association Rules

1.7.1 Definitions

- $I = \{i_1, i_2, ..., i_m\}$ a set of literals (Items)
- H a directed acyclic graph over I
- Edge in H i to j:
  - i is a genralization of j
  - i is called $father$ or direct predecesor of j
  - j is a son or direct sucessor of i
- $\bar{x}$ is predecessor of x (x successor of $\bar{x}$) w.r.t H:
  - there is a path from x to x in H
- Set of items $\bar{Z}$ is *predecessor* of set items Z:
  - at least one item in $\bar{Z}$ predecessor of an item in Z
- D is a set of transaction T, where $T \subseteq I$
- Typically, transactions T contain only elements from the leaves of graph H
- Transaction T supports item $i \in I$
  - $i \in T$ or i is predecessor of an item $j \in T$
- T supports set $X \subseteq I$ of items
  - T supports each item in X
- Support of set $X \subseteq I$ of items in D
  - Percentage of transactions in D supporting X.
- Multilevel association rule:
  - $X \Rightarrow Y$ where $X \subseteq I, Y \subseteq I, X \cap T = \varnothing$
  - and no item in Y is predecessor w.r.t. H of an item in X
- Support s of a multilevel association rule $X \Rightarrow Y$ in D:
  - Support of set $X \cup Y$ in D
- Confidence c of a multilevel association rule $X \Rightarrow Y$ in D:
  - Percentage of transactions containing Y in the subset of all transactions in D that contain X

1.8 Determining the Frequent Itemsets

1.8.1 Idea

- Extend database transactions by all predecessors of items contained in that transaction
- Method
  - Insert each item i transaction T together with all its predessors w.r.t. H into new transaction T'
  - Do not insert duplicates
- Then Determine frequent itemsets for basic association rules (e.g. Apriori algorithm)
- Basic algorithm for multilevel association rules

1.8.2 Optimizations of the Basic Algorithm

- Materialization of Predecessors
  - Additional data structure H: Item $\rightarrow$ list of all its predecessors
  - More efficient access to the predecessors
- Filtering the predecessors to be added
  - Add only those predecessors that occur in an element of candidate set C_k
  - Example: $C_k = \{\{Clothes, Shoes\}\}$, replace "JacketXT" by "Clothes"
- Discard redundant item sets
  - Let X an k-item set, i an item an $\bar{i}$ a predecessor of i
  - $X = \{i, \bar{i}, ...\}$
  - Support of $X - i$ = support of X
  - X can be discarded during candidate generation
  - Do not need to count supoort of k-itemset that contains item i and predecessor of i
- Algorithm *Cumulate*

1.8.3 Stratification

- Alternative to the basic algorithm (Apriori-algorithm)
- Stratification = form layers in the candidates sets
- Property: Itemset $\bar{X}$ is infrequent and $\bar{X}$ is predecessor of X: X is infrequent
- Method:
  - Do not count all k-itemsets at the same time
  - Count support first for the more general itemsets and cunt more special item serts only if necessary
- Example:
  - $C_k = \{\{Clothes, Shoes\}, \{Outerwear, Shoes\}, \{JacketsShoes\}\}$
  - Count support first for $\{Clothes, Shoes\}$
  - Count support for $\{Outerwear, Shoes\}, \{JacketsShoes\}\}$ only if $\{Clothes, Shoes\}$ is frequent
- Notations
  - *Depth* of an itemset
    - For itemsets X in candidate set C_k without direct predecessor in C_k : $Depth(X) = 0$
    - For all other itemsets X in C_k:
      $Depth(X) = max\{Depth(\bar{X}) | \bar{X} \in C_k$ is direct predecessor of X$\}$ + 1
  - C_k^n : Set of itemsets from C_k with depth n, $0 \leq n \leq$ maximal depth t

1.9 Pattern Summarization