University
Politehnica
of Bucharest

Faculty of
Automatic
Control and
Computers

Computer
Science and
Engineering
Department

Association Rules and Sequential Patterns

Ciprian-Octavian Truică
ciprian.truica@cs.pub.ro

Overview

- Frequent Itemsets and Association Rules
- Apriori algorithm
- FP-Growth
- Class association rules
- Sequential patterns. GSP algorithm

Overview

- Frequent Itemsets and Association Rules
- Apriori algorithm
- FP-Growth
- Class association rules
- Sequential patterns. GSP algorithm

Frequent itemsets and Association Rules

- Frequent itemsets and rules:
  - Items and transactions
  - Association rules
  - Goals for mining transactions

Items and Transactions

- Let $I = \{i_1, i_2, \dots, i_m\}$ be a set $m = |I|$ of **items**.
- A **transaction** t is a set of items, with $t \subseteq I$.
- A **transaction dataset** (or database) $T = \{t_1, t_2, \dots, t_n\}$ is a set of $n = |T|$ transactions. Each transaction t_i may contain a different number of items.
- An **itemset** S is a subset of I $(S \subseteq I)$. If $v = |S|$ is the number of items in S (or the cardinal of S), then S is called a **v-itemset**.
- The **support** of an itemset X, $support(X)$, is equal to the number (or proportion) of transactions in T containing X.

$$support(X) = |t \in T : X \subseteq t| \text{ or } support(X) = \frac{|t \in T : X \subseteq t|}{|T|}$$

Frequent itemsets and Association Rules

- Frequent itemsets and rules:
  - Items and transactions
  - Association rules
  - Goals for mining transactions

Association Rules

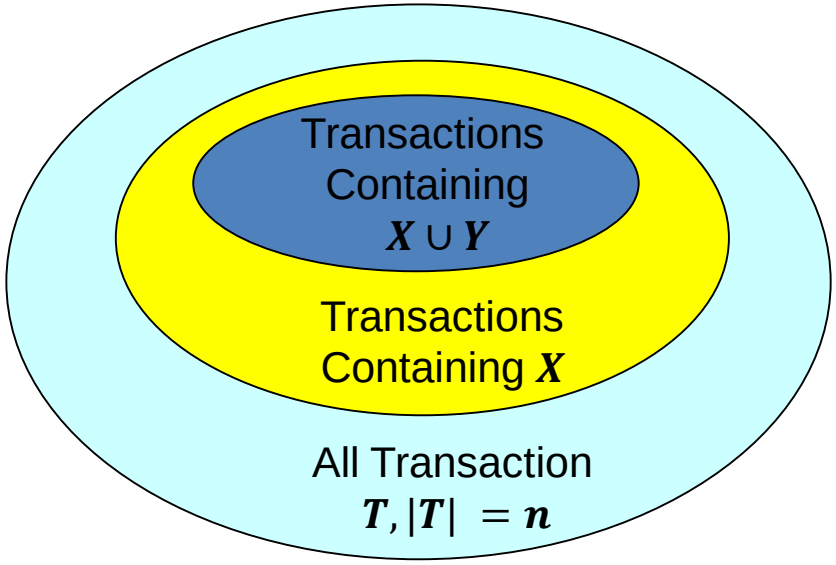
- If X and Y are two itemsets, an **association rule** is an implication of the form $X \rightarrow Y$ where $X \cap Y = \emptyset$.
- X is the antecedent of the rule
- Y is the consequent of the rule
- For each association rule we can compute the **support of the rule** and the **confidence of the rule**.
- If $n = |T|$ (n is the number of transactions in T) then:

$$support(X \rightarrow Y) = support(X \cup Y) \; or \; support(X \rightarrow Y) = \frac{support(X \cup Y)}{n}$$

$$confidence(X \longrightarrow Y) = \frac{support(X \rightarrow Y)}{support(X)} = \frac{support(X \cup Y)}{support(X)}$$

Association Rules

- The **support** of a rule $X \rightarrow Y$ is given by the proportion of transactions in T containing both X and Y
- The **confidence** of a rule $X \rightarrow Y$ is given by the proportion of transactions containing Y in the set of transactions containing X (the set of transactions containing $X \cup Y$ is included in the set of transactions containing X).



Frequent itemsets and rules

- If the **support** of a rule **is high** it describes a relationship between itemsets that are found together in many transactions
- If the **confidence** of a rule $X \rightarrow Y$ **is high** then if a transaction contains X then with a high probability (equal to the confidence of the rule) it also contains Y
- We accept a rule as a valid one if the support and the confidence of the rule are at least equal with some given **thresholds**

Algorithm

- Given a threshold for the support s and confidence c find all association rules:
  - **Step 1.** Find all frequent itemsets F containing at least two items for which $support(F) \geq s$
  - **Step 2.** For each frequent itemset $U \subseteq F$ list all splits (X, Y) with $X \cap Y = \emptyset$ and $X \cup Y = U$. Each split generates a rule $X \rightarrow Y$.
  - **Step 3.** Find all rules for which $confidence(X \rightarrow Y) \geq c$

Frequent itemsets and Association Rules

- Frequent itemsets and rules:
  - Items and transactions
  - Association rules
  - Goals for mining transactions

Goals for mining Transactions

- Goal 1: **Find frequent itemsets**. Frequent itemsets can be used to identify the most important items in a set.
- Goal 2: **Find association rules**. Such a rule tells that some items are highly correlated to some other items.

Example

- If items are words and transactions are documents, where each document is considered a bag of words, then we can have:
  - T = {Doc1, Doc2, …, Doc6}
  - Doc1 = {rule, tree, classification}
  - Doc2 = {relation, tuple, join, algebra, recommendation}
  - Doc3 = {variable, loop, procedure, rule}
  - Doc4 = {clustering, rule, tree, recommendation}
  - Doc5 = {join, relation, selection, projection, classification}
  - Doc6 = {rule, tree, recommendation}

Example

- In that case:
  - support({rule, tree}) = 3 or 50% or 0.5
  - support({relation, join}) = 2 or 33.33% or 1/3
  - support(rule→tree) = 50%
  - confidence(rule→tree) = 75%
  - support(tree→rule) = 50%
  - confidence(tree→rule) = 3/3 = 100%
- If the threshold is s = 50% (or 0.5) then {rule, tree} is frequent and {relation, join} is not.
- If the minimum confidence required is 80% then only the second rule is kept, the first being considered not enough powerful.

Algorithms

- There are many algorithms for finding frequent itemsets and consequently the association rules in a dataset.
- All these algorithms are developed for huge volumes of data, meaning that the dataset is too large to be loaded and processed in the main memory.
- For that reason minimizing the number of times the data are read from the disk become a key feature of each algorithm
- Algorithms:
  - Apriori
  - FP-Growth

Overview

- Frequent Itemsets and Association Rules
- Apriori algorithm
- FP-Growth
- Class association rules
- Sequential patterns. GSP algorithm

Apriori Algorithm

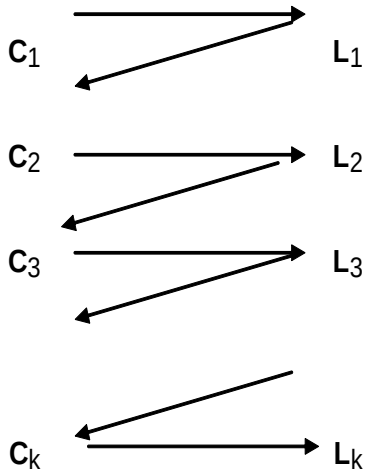
- This algorithm is based on the Apriori principle: *The Apriori principle states that any subset of a frequent itemset is also a frequent itemset*
- Consequently each frequent v-itemset is the reunion of v (v-1)-itemsets.
- That means we can determine the frequent itemsets with dimension v examining only the set of all frequent itemsets with dimension (v-1).
- It is a ***level-wise approach*** where each step requires a full scan of the dataset

Apriori Algorithm

1. Find frequent 1-itemsets (frequent items)
2. Find frequent 2-itemsets considering all pairs of frequent items found in step 1
3. Find frequent 3-itemsets considering all triplets with each subset in the frequent pairs set found in step 2
4. . . . and so on.

Apriori Algorithm

- C_i is the set of candidates for frequent i-itemsets and L_i is the actual set of frequent i-itemsets.
- C_i is the set of all itemsets found in transactions (a subset of I) and may be obtained either by a reunion of all transactions in T or by considering $C_1 = I$ (in that case some items may have a zero support)
- The process stops in two cases:
  - No candidate from C_k has the support at least min support s (L_k is empty)
  - There is no $(k+1)$-itemset with all k-subsets in L_k (meaning that C_{k+1} is empty)



Apriori Algorithm

Algorithm Apriori (T, s)

$L_1 \leftarrow I$ or $L_1 = \bigcup_{t \in T}\{a | a \in t\}$

for $(k = 2;\ L_{k-1} \neq \emptyset, k++)$ **do**

$C_k \leftarrow generateCandidates(L_{k-1})$

for each transaction t $\in T$ **do**

$C_t \leftarrow \{c | c \in C_k \wedge c \subseteq t\}$;

for each candidate $c \in C_t$ **do**

$count[c] \leftarrow count[c] + 1$

end

end

$L_k \leftarrow \{c | c \in C_k \wedge count[c] \geq s\}$

end

return $\bigcup_k L_k$

Apriori Algorithm

- Candidate generation is also described in the original algorithm as having two steps: the join step and the prune step.
- The join step builds a larger set of candidates
- C_k is generated by joining L_{k-1} with itself
- The prune step removes some of the candidates that proved impossible to be frequent.
- Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k-itemset.

Example

- Consider again the set of six transactions in:
  - Doc1 = {rule, tree, classification}
  - Doc2 = {relation, tuple, join, algebra, recommendation}
  - Doc3 = {variable, loop, procedure, rule}
  - Doc4 = {clustering, rule, tree, recommendation}
  - Doc5 = {join, relation, selection, projection, classification}
  - Doc6 = {rule, tree, recommendation}
- And a minimum support of 50% (minimum support $s = 3$).

Example

- At the first scan of the transaction dataset T the support for each item is computed:

Rule	4	recommendation	3
Tree	3	variable	1
classification	2	loop	1
relation	2	procedure	1
Tuple	1	clustering	1
Join	2	selection1	1
algebra	1	projection	1

- With a minimum support $s = 3$ then $L_1 = \{\{rule\}, \{tree\}, \{recommendation\}\}$.

Example

- Considering: rule < tree < recommendation
- From the join $C_2 = \{\{rule, tree\}, \{rule, recommendation\}, \{tree, recommendation\}\}$.
- The prune step does not modify C_2.
- The second scan of the transaction dataset leads to the following pair support values:

{rule, tree}	**3**
{rule, recommendation}	**2**
{tree, recommendation}	**2**

- The only frequent pair is {rule, tree}: $L_2 = \{\{rule, tree\}\}$.

Example

- **Step 3**. Because L_2 has a single element, $C_3 = \emptyset$, so $L_3 = \emptyset$, and the process stops.
- $L = L_1 \cup L_2 = \{\{rule\}, \{tree\}, \{recommendation\}, \{rule, tree\}\}$.
- If we consider only maximal itemsets, $L = \{\{recommendation\}, \{rule, tree\}\}$.

Apriori Algorithm

- Advantages
  - Uses large itemsets property
  - Easily parallelized
  - Easy to implement
- Disadvantages
  - Assumes transaction dataset is resident in memory
  - Requires many dataset scans

Overview

- Frequent Itemsets and Association Rules
- Apriori algorithm
- FP-Growth
- Class association rules
- Sequential patterns. GSP algorithm

FP-Growth

- FP-Growth (Frequent Pattern Growth) algorithm performs frequent itemsets discovery without candidate generation. It has a 2 steps approach:
  - **Step 1**: Build FP-tree. Requires only 2 passes over the dataset.
  - **Step 2**: Extracts frequent itemsets from the FP-tree

FP-Growth

- At the first pass over the data frequent items are discovered.
- All other items (infrequent items) are discarded: transactions will contain only frequent items.
- Also, these items are ordered by their support in decreasing order.

FP-Growth

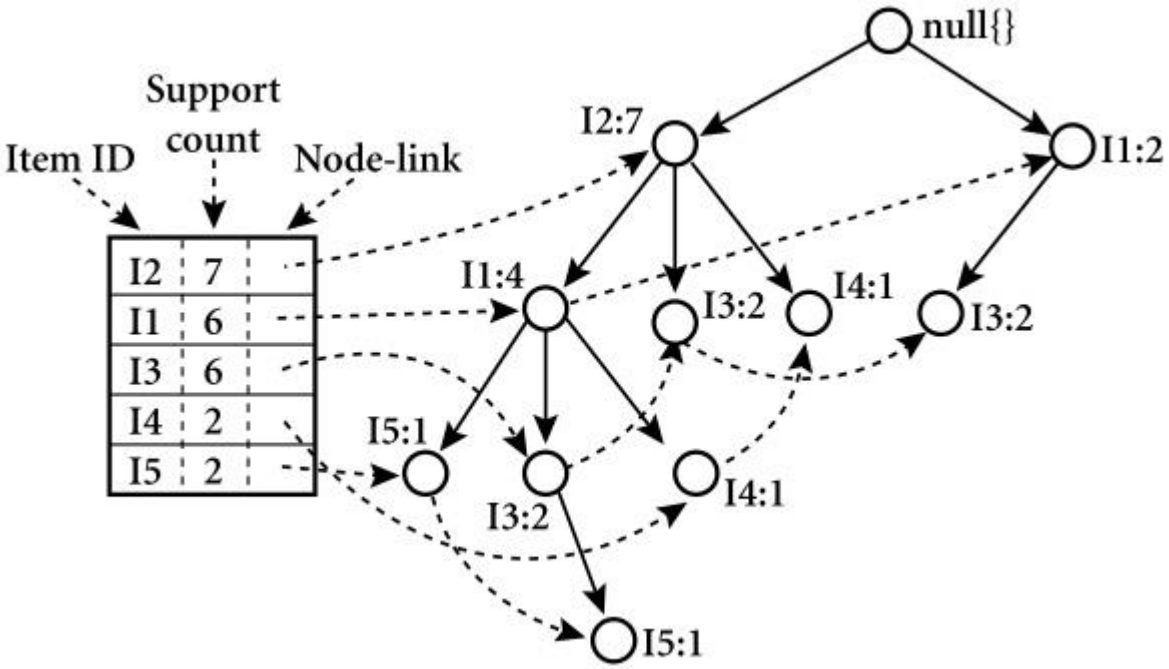
- At the second pass over the data the FP-tree is built:
  - FP-Growth considers an ordered set of items (frequent items ordered by their support).
  - Each transaction is written with items in that order.
  - The algorithm reads a transaction at a time and adds a new branch to the tree, branch containing as nodes the transaction items.
  - Each node has a counter.
  - If two transactions have the same prefix the two branches overlap on the nodes of the common prefix and the counter of those nodes are incremented.
  - Also, nodes with the same item are linked by orthogonal paths.

FP-Growth

- After building the FP-tree the algorithm starts to build partial trees (called conditional FP-trees) ending with a given item (a suffix).
- The item is not present in the tree but all frequent itemsets generated from that conditional tree will contain that item.
- In building the conditional FP-tree, non-frequent items are skipped (but the branch remains if there are still nodes on it).

FP-Growth

Trans.	Itemset
T1	I1,I2,I5
T2	I2,I4
T3	I2,I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3



Source:
https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm#/media/File:FPG_FIG_01.jpg

Overview

- Frequent Itemsets and Association Rules
- Apriori algorithm
- FP-Growth
- Class association rules
- Sequential patterns. GSP algorithm

Class Association Rules

- Given a set of $m = |I|$ of **items** $I = \{i_1, i_2, \dots, i_m\}$ and a set of $n = |T|$ transactions $T = \{t_1, t_2, \dots, t_n\}$
- Each transaction t_i is labeled with a class $c, c \in C$ where $C = \{c_1, c_2, \dots, c_p\}$ and $C \cap I = \emptyset$
- A class association rule is a construction with the following syntax:

$$X \rightarrow y \text{ where } X \subseteq I \text{ and } y \in C$$

- The definition of the **support** and **confidence** for a class association rule is the same with the case of association rules.

Class Association Rules

- Consider the following set of six transactions, now labeled with class labels from C = {database, datamining, programming}:

Doc1	**{rule, tree, classification}**	**datamining**
Doc2	**{relation, tuple, join, algebra, recommendation}**	**database**
Doc3	**{variable, loop, procedure, rule}**	**programming**
Doc4	**{clustering, rule, tree, recommendation}**	**datamining**
Doc5	**{join, relation, selection, projection, classification}**	**database**
Doc6	**{rule, tree, recommendation}**	**datamining**

Class Association Rules

Then the CARs:

rule → datamining;

recommendation → database

has:

- support(rule → datamining) = 3/6 = 50%,
- confidence(rule → datamining) = 3/4 = 75%.
- support(recommendation → database) = 1/6 ≅ 17%,
- confidence(recommendation → database) = 1/3 ≅ 33%

For a minimum support $s = 50\%$ and a minimum confidence $c = 50\%$ the first rule stands and the second is rejected.

Class Association Rules

- Algorithm for mining CARs using a modified Apriori algorithm:
  - At the first pass over the algorithm computes F_1 where

$$F_1 = \{ \text{the set of CARs with a single item on the left side verifying a given minimum support } s \text{ and minimum confidence } c \}.$$

  - At step k, C_k is built from F_{k-1} and then, passing through the data and counting for each member of C_k the support and the confidence, F_k is determined.
  - Candidate generation is almost the same as for association rules with the only difference that in the join step only CARs with the same class in the right side are joined.

Class Association Rules

$C_k = \emptyset$ // starts with an empty set
forall f1, f2 $\in F_{k-1}$ // for each pair of frequent CAR
    $f_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\} \rightarrow y$ // only last item
    $f_2 = \{i_1, \ldots, i_{k-2}, i'_{k-1}\} \rightarrow y$ // is different
    $i_{k-1} < i'_{k-1}$ do // and same class

    $c = \{i_1, \ldots, i_{k-1}, i'_{k-1}\} \rightarrow y$; // join step
    $C_k = C_k \cup \{c\}$; // add new candidate

    for each (k-1)-subset s of $\{i_1, \ldots, i_{k-1}, i'_{k-1}\}$ do
        if ($s \rightarrow y \notin F_{k-1}$) then
            $C_k = C_k - \{c\}$; // prune step
  endfor
endfor

Overview

- Frequent Itemsets and Association Rules
- Apriori algorithm
- FP-Growth
- Class association rules
- Sequential patterns. GSP algorithm

Sequential Patterns

- **Itemset**: a set of n distinct items $I = \{i_1, i_2, \ldots, i_n\}$

  Example: *I={A,B,C,D,E,M}*
- **Event**: a non-empty collection of items; we can assume that items are in a given order: $(i_1, i_2, \ldots, i_k)$
- **Sequence** : an ordered list of events: $< e_1, e_2, \ldots, e_m >$
- **Length** of a sequence: the number of items in the sequence

  Example: *<AM, CDE, AE>* has length 7
- **Size** of a sequence: the number of itemsets in the sequence

  Example: *<AM, CDE, AE>* has size 3
- **K-sequence** : sequence with k items, or with length k

  Example: *<B, AC>* is a 3-sequence

Sequential Patterns

- **Subsequence** and **supersequence**:
  - $E = < e_1, e_2, \ldots, e_u >$ is a subsequence of/or included in $F = < f_1\ f_2\ \ldots f_v >$ if there are some integers $1 \leq j_1 < j_2 < \ldots < j_{u-1} < j_u \leq v$ such that $e_1 \subseteq f_{j1}$ and $e_2 \subseteq f_{j2}$ and … and $e_u \subseteq f_{ju}$.
  - F is a supersequence of the first sequence E or it contains the entire sequence ($E \subseteq F$)

Sequential Patterns

- **Sequence dataset (database)** X: a set of sequences
- **Frequent sequence** (or **sequential pattern**): a sequence included in more than **s** members of the sequence database X;
- **s** is the user-specified minimum support.
- The number of sequences from X containing a given sequence is called the **support** of that sequence.
- So, a frequent sequence is a sequence with a support at least **s** where **s** is the **minimum support** specified by the user.

GSP Algorithm

- Similar with Apriori:

Algorithm GSP(I, X, minsup)

C_1 = I // initial n candidates

L_1 = {<{f}>| f∈ C_1, f.count/n ≥ minsup}; // first pass over X

for (k = 2; L_{k-1} ≠ ∅; k++) do // loop until L_{k-1} is empty

C_k = candidate-generation(L_{k-1});

foreach s ∈ X do //

foreach c ∈ C_k do

if c is-contained-in s then

c.count++;

endfor

endfor

L_k = {c ∈ C_k | c.count/n ≥ minsup}

endfor

return $\cup_k F_k$;

GSP Algorithm

- Candidate generation is made in a join and prune manner.
- At the join step two sequences f_1 and f_2 from L_{k-1} are joined if removing the first item from f_1 and the last item from f_2 the result is the same.
- The joined sequence is obtained by adding the last item of f_2 to f_1, with the same status (separate element or part of the last element of f_1).

Summary

- This third course presented:
  - What are frequent itemsets and rules and their relationship
  - Apriori and FP-growth algorithms for discovering frequent itemsets.
  - What are class association rules and how can be mined
  - An introduction to sequential patterns and the GSP algorithm