723A75 Advanced Data Mining
TDD41 Data Mining - Clustering and Association Analysis

Lecture 8: Constrained Frequent Itemset Mining

Johan Alenlöv
IDA, Linköping University, Sweden

Outline

- Content
  - Recap
  - Monotone and anitmonotone constraints
  - Apriori algorithm with constraints
  - FP grow algorithm with constraints
  - Convertible (anti) monotone constraints
  - Summary
- Litterature
  - Course Book. 2nd ed.: 5.5. 3rd ed.: 7.3
  - Pei, J., and Han, J. **Can We Push More Constraints into Frequent Pattern Mining?**. In Proc. of the 2000 Int. Conf. on Knowledge Discovery and Data Mining, 2000.

Recap: FP Grow Algorithm

Algorithm: FP-tree(D, *minsup*)
Input: A transactional database D, and the minimum support *minsup*.
Output: The FP tree for D and *minsup*.

Count support for each item in D
Remove the infrequent items from the transactions in D
Sort the items in each transaction in D in support descending order
Create a FP tree with a single node T with $T.name = NULL$
for each transaction $I \in D$ do
    insert-tree(I, T)

Algorithm: insert-tree($I_1, \ldots I_m, T$)
Input: An itemset $I_1, \ldots, I_m$, and a node T in the FP tree.
Output: Modified FP tree.

if T has a child N such that $N.name = I_1.name$ then
    $N.count{+}{+}$
else
    create a new child N of T with $N.name = I_1.name$ and $N.count = 1$
if $m > 1$ then
    insert-tree ($I_2, \ldots, I_m, N$)

Recap: FP Grow Algorithm

- To mine the FP tree *Tree*, call FP-grow(*Tree*, NULL, *minsup*).

Algorithm: FP-grow(*Tree*, α, *minsup*)
Input: A FP tree *Tree*, an itemset α, and the minimum support *minsup*.
Output: All the itemsets in *Tree* that end with α and have *minsup*.

1 for each item X in *Tree* do
2   output the itemset $\beta = X \cup \alpha$ with support=$X.count$
3   build the β conditional database and the corresponding FP tree $Tree_\beta$
4   if $Tree_\beta$ is not empty then call FP-grow($Tree_\beta$, β, *minsup*)

- The algorithm above can be made more efficient by adding the lines below.

0.1 if *Tree* has a single branch then
0.2   for each combination β of the nodes in the branch do
0.3   output the itemset $\beta \cup \alpha$ with support $= \min_{X \in \beta} X.count$
0.4 else

- The FP grow algorithm is correct.

Exercise

- Run the FP grow algorithm on the database below with minsup 2.

Tid	Items
1	A, B, E
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	A, B, C, E
9	A, B, C

- Show the execution details (i.e. FP tree construction, conditional databases, recursive calls), not just the frequent itemsets found.

Monotone and Antimonotone Constraints

- A constraint is a function that returns **true** or **false** for every itemset.
- It tells us if the itemset satisfies the constraint or not.
  - The itemset has support equal or greater than a given value.
  - The sum of the prices of the items in the itemset is greater than a given value.
  - The most expensive item in the itemset cost less than a given value.
  - The itemset contains a specific value.
  - The itemset contains exactly a certain number of items.
- A constraint C is monotone when for every itemset A and B such that $A \subseteq B$, if $C(A) = \text{true}$ then $C(B) = \text{true}$.
  - The itemset contains a certain item.
- A constraint C is antimonotone when for every itemset A and B such that $A \subseteq B$, if $C(B) = \text{true}$ then $C(A) = \text{true}$.
  - The support of the itemset is equal or greater than a given value.
- Alternatively, C is antimonotone when for every itemset A and B such that $A \subseteq B$, if $C(A) = \text{false}$ then $C(B) = \text{false}$.
- Note that the **apriori property** applies to every antimonotone constrain, i.e. no need to check the constraint for supersets of A if $C(A) = \text{false}$.

Examples of Monotone and Antimonotone Constraints

- Here S is the set of prices in the itemset and v is a given values.
- Examples of monotone constraints:
  - $\text{sum}(S) \geq v$
  - $\text{min}(S) \leq v$
  - $\text{range}(S) \geq v$
- Examples of antimonotone constraints:
  - $\text{sum}(S) \leq v$
  - $\text{max}(S) \leq v$
  - $\text{range}(S) \leq v$

Examples of Monotone and Antimonotone Constraints

Constraint	Antimonotone	Monotone
$v \in S$	no	yes
$S \supseteq V$	no	yes
$S \subseteq V$	yes	no
$\min(S) \leq v$	no	yes
$\min(S) \geq v$	yes	no
$\max(S) \leq v$	yes	no
$\max(S) \geq v$	no	yes
$\text{count}(S) \leq v$	yes	no
$\text{count}(S) \geq v$	no	yes
$\text{sum}(S) \leq v \ (a \in S, a \geq 0)$	yes	no
$\text{sum}(S) \geq v \ (a \in S, a \geq 0)$	no	yes
$\text{range}(S) \leq v$	yes	no
$\text{range}(S) \geq v$	no	yes
$\text{avg}(S)\, \theta\, v,\ \theta \in \{\leq, \geq\}$	No but convertible	No but convertible
$\text{support}(S) \geq v$	yes	no
$\text{support}(S) \leq v$	no	yes

Apriori Algorithm with Antimonotone Constraint

Algorithm: apriori(D, *minsup*, C)
Input: A transactional database D, *minsup*, **and an antimonotone constraint** C.
Output: All the large itemsets in D **that satisfy** C.

```
L_1 = { large 1-itemsets that satisfy C }
for (k = 2; L_{k-1} ≠ ∅; k++) do
    C_k = apriori-gen(L_{k-1})        // Generate candidate large k-itemsets
    for all t ∈ D do
        for all c ∈ C_k such that c ∈ t do
            c.count++
    L_k = {c ∈ C_k | c.count ≥ minsup and C(c)=true}
return ⋃_k L_k
```

Apriori Algorithm with Antimonotone Constraint

Run the Apriori algorithm with the following database with minsup 2.

Tid	Items
1	1, 3, 4
2	2, 3, 5
3	1, 2, 3, 5
4	2, 5

$\xrightarrow{C_1}$

Itemset	sup
$\{1\}$	2
$\{2\}$	3
$\{3\}$	3
$\{4\}$	1
$\{5\}$	3

$\xrightarrow{L_1}$

Itemset	sup
$\{1\}$	2
$\{2\}$	3
$\{3\}$	3
$\{5\}$	3

$\longrightarrow$

$\xrightarrow{C_2}$

Itemset
$\{1,2\}$
$\{1,3\}$
$\{1,5\}$
$\{2,3\}$
$\{2,5\}$
$\{3,5\}$

$\longrightarrow$

Itemset	sup
$\{1,2\}$	1
$\{1,3\}$	2
$\{1,5\}$	1
$\{2,3\}$	2
$\{2,5\}$	3
$\{3,5\}$	2

$\xrightarrow{L_2}$

Itemset	sup
$\{1,3\}$	2
$\{2,3\}$	2
$\{2,5\}$	3
$\{3,5\}$	2

$\longrightarrow$

$\xrightarrow{C_3}$

Itemset
$\{2,3,5\}$

$\xrightarrow{L_3}$

Itemset	sup
$\{2,3,5\}$	2

Apriori Algorithm with Antimonotone Constraint

Run the Apriori algorithm with the following database with minsup 2 and constraint sum$(S) < 5$ where the item price coincides with the item id.

Tid	Items
1	1, 3, 4
2	2, 3, 5
3	1, 2, 3, 5
4	2, 5

$\xrightarrow{C_1}$

Itemset	sup
$\{1\}$	2
$\{2\}$	3
$\{3\}$	3
$\{4\}$	1
$\{5\}$	3

$\xrightarrow{L_1}$

Itemset	sup
$\{1\}$	2
$\{2\}$	3
$\{3\}$	3
~~$\{5\}$~~	~~3~~

$\longrightarrow$

$\xrightarrow{C_2}$

Itemset
$\{1,2\}$
$\{1,3\}$
~~$\{1,5\}$~~
$\{2,3\}$
~~$\{2,5\}$~~
~~$\{3,5\}$~~

$\longrightarrow$

Itemset	sup
$\{1,2\}$	1
$\{1,3\}$	2
~~$\{1,5\}$~~	~~1~~
$\{2,3\}$	2
~~$\{2,5\}$~~	~~3~~
~~$\{3,5\}$~~	~~2~~

$\xrightarrow{L_2}$

Itemset	sup
$\{1,3\}$	2
~~$\{2,3\}$~~	~~2~~
~~$\{2,5\}$~~	~~3~~
~~$\{3,5\}$~~	~~2~~

$\longrightarrow$

$\xrightarrow{C_3}$

Itemset
~~$\{2,3,5\}$~~

$\xrightarrow{L_3}$

Itemset	sup
~~$\{2,3,5\}$~~	~~2~~

Apriori Algorithm with Monotone Constraint

Algorithm: apriori(D, $minsup$, C)
Input: A transactional database D, $minsup$, **and a monotone constraint** C.
Output: All the large itemsets in D **that satisfy** C.

$L_1 = \{$ large 1-itemsets $\}$
for $(k = 2; L_{k-1} \neq \emptyset; k++)$ do
    $C_k =$ apriori-gen(L_{k-1}) // Generate candidate large k-itemsets
    for all $t \in D$ do
        for all $c \in C_k$ such that $c \in t$ do
            $c.count++$
    $L_k = \{c \in C_k | c.count \geq minsup\}$
return $\{c \in \bigcup_k L_k | C(c) = true$ **or** $C(d) = true$ **for some** $d \subseteq c\}$

Apriori Algorithm with Antimonotone Constraint

Run the Apriori algorithm with the following database with minsup 2 and constraint $\text{sum}(S) \geq 5$ where the item price coincides with the item id.

Tid	Items
1	1, 3, 4
2	2, 3, 5
3	1, 2, 3, 5
4	2, 5

$\xrightarrow{C_1}$

Itemset	sup
$\{1\}$	2
$\{2\}$	3
$\{3\}$	3
$\{4\}$	1
$\{5\}$	3

$\xrightarrow{L_1}$

Itemset	sup
$\{1\}$	2
$\{2\}$	3
$\{3\}$	3
$\{5\}$	3

$\longrightarrow$

$\xrightarrow{C_2}$

Itemset
$\{1,2\}$
$\{1,3\}$
$\{1,5\}$
$\{2,3\}$
$\{2,5\}$
$\{3,5\}$

$\longrightarrow$

Itemset	sup
$\{1,2\}$	1
$\{1,3\}$	2
$\{1,5\}$	1
$\{2,3\}$	2
$\{2,5\}$	3
$\{3,5\}$	2

$\xrightarrow{L_2}$

Itemset	sup
$\{1,3\}$	2
$\{2,3\}$	2
$\{2,5\}$	3
$\{3,5\}$	2

$\longrightarrow$

$\xrightarrow{C_3}$

Itemset
$\{2,3,5\}$

$\xrightarrow{L_3}$

Itemset	sup
$\{2,3,5\}$	2

FP Grow Algorithm with Monotone Constraint

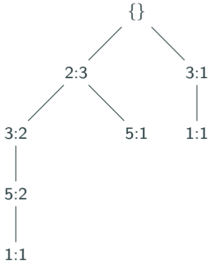
Algorithm: FP-grow(*Tree*, α, *minsup*, *C*)
Input: A FP tree *Tree*, an itemset α, *minsup*, **and a monotone constraint** *C*.
Output: All the itemsets in *Tree* that end with α, have *minsup* **and satisfy** *C*.

if $C(\alpha) = true$ **then**
    replace *C* **with** C_{true} // C_{true} **is a constraint that always returns true**
for each item *X* in *Tree* do
    output the itemset $\beta = X \cup \alpha$ with support=*X.count* **if** $C(\beta) = true$
    build the β conditional database and the corresponding FP tree $Tree_{\beta}$
    if $Tree_{\beta}$ is not empty then call FP-grow($Tree_{\beta}$, β, *minsup*, *C*)

FP Grow Algorithm with Monotone Constraint

Run the FP Grow algorithm with the following database with minsup 2 and constraint $\text{sum}(S) \geq 5$ where the item price coincides with the item id.

Tid	Items
1	1, 3, 4
2	2, 3, 5
3	1, 2, 3, 5
4	2, 5



Item	Conditional database
2	-
3	2:2
5	2:1, 2,3:2
1	3:1, 2,3,5:1

5-conditional database

Tid	Items
1	2
2	2, 3
3	2, 3

Now $C(5) = \mathrm{true}$ so we replace C with C_{true}.

FP Grow Algorithm with Antimonotone Constraint

Algorithm: FP-tree(D, *minsup*, C)
Input: A transactional database D, *minsup*, **and an antimonotone constraint** C.
Output: The FP tree for D, *minsup* **and** C.

Count support for each item in D
Remove the infrequent items from the transactions in D
Remove the items that do not satisfy C from the transactions in D
Sort the items in each transaction in D in support descending order
Create a FP tree with a single node T with $T.name = NULL$
for each transaction $I \in D$ do
    insert-tree(I, T)

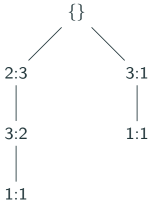
Algorithm: FP-grow($Tree$, α, *minsup*, C)
Input: A FP tree $Tree$, an itemset α, *minsup*, **and an antimonotone constraint** C.
Output: All the itemsets in $Tree$ that end with α, have *minsup* **and satisfy** C.

let δ denote all the items in $Tree$
if $C(\alpha \cup \delta) = true$ then
    replace C with C_{true}   **// C_{true} is a constraint that always returns true**
for each item X in $Tree$ do
    if $C(X \cup \alpha) = true$ then
        output the itemset $\beta = X \cup \alpha$ with support=$X.count$
        build the β conditional database and the corresponding FP tree $Tree_\beta$
        if $Tree_\beta$ is not empty then call FP-grow($Tree_\beta$, β, *minsup*, C)

FP Grow Algorithm with Antimonotone Constraint

Run the FP Grow algorithm with the following database with minsup 2 and constraint $\text{sum}(S) < 5$ where the item price coincides with the item id.

Tid	Items
1	1, 3, 4
2	2, 3, ~~5~~
3	1, 2, 3, ~~5~~
4	2, ~~5~~



Item	Conditional database
2	-
3	2:2
1	3:1, 2,3:1

3-conditional database

Tid	Items
1	2
2	2

Now $C(3 \cup 2) = \text{true}$ so we replace C with C_{true}.

Convertible Monotone and Antimonotone Constraint

- We saw that $\text{avg}(S) \leq v$ and $\text{avg} \geq v$ was neither monotone nor antimonotone.
- A constraint C is convertible monotone when there exists an item order R such that for every itemsets A and B respecting R such that A is a suffix of B, if $C(A) = \text{true}$ then $C(B) = \text{true}$.
  - $\text{avg}(S) \geq v$ with respect to decreasing price order.
- A constraint C is convertible antimonotone when there exists an item order R such that for every itemsets A and B respecting R such that B is a suffix for A, if $C(A) = \text{true}$ then $C(B) = \text{true}$.
  - $\text{avg}(S) \geq v$ with respect to increasing price order.
- Alternatively, C is convertible antimonotone when there exists an item order R such that for every itemsets A and B respecting R such that B is a suffix for A, if $C(B) = \text{false}$ then $C(A) = \text{false}$.
- A constraint that is both convertible monotone and antimonotone is called strongly convertible.

Convertible Monotone and Antimonotone Constraints

Constraint	Convertible antimonotone	Convertible monotone	Strongly convertible
$\text{avg}(S) \leq, \geq v$	Yes	Yes	Yes
$\text{median}(S) \leq, \geq v$	Yes	Yes	Yes
$\text{sum}(S) \leq v$[1], $v \geq 0$	Yes	No	No
$\text{sum}(S) \leq v$[1], $v \leq 0$	No	Yes	No
$\text{sum}(S) \geq v$[1], $v \geq 0$	No	Yes	No
$\text{sum}(S) \geq v$[1], $v \leq 0$	Yes	No	No
⋮	⋮	⋮	⋮

[1] For the sum constraints the prices are of any value (negative or positive)

Summary

- Constraints can be added to the mining process to find itemsets that satisfy a certain constraint.
- Constraints can be **antimonotone**, **monotone**, **convertible** or **inconvertible**
- Depending on the type of constraint different modification to the mining algorithms are made.