Mining Free Itemsets under Constraints

By Jean-Francois Boulicaut and Baptiste Jeudy
International Database Engineering and Application Symposium

Content

Introduction

- Frequent itemset mining
  - A set of items is referred to as itemset
  - X is an item(or itemset), $Support(X) = \frac{\# X}{n}$
  - Support is bounded by a threshold r
  - A frequent itemset is an itemset with a support larger than the minimum support
  - Given a database, find all the frequent itemsets

Introduction

- Problems with frequent itemset mining algorithms
  - The computation may be intractable for a user-given frequency threshold: the number of frequent itemsets may explode
  - Lack of focus leads to huge output of frequent itemsets

Introduction

- Two issues to tackle these problems
  - Constraint-based extraction of the frequent itemsets: only a subset of the collection of frequent itemsets is interesting.
  - Condensed representation of frequent itemsets: extract a subset of the frequent patterns and regenerate the whole collection when necessary

Introduction

- Constraint-based extraction of frequent itemsets
  - Syntactic constraints
    - an item must not appear in the itemsets
  - Constraints related to objective measures of interestingness
    - the itemsets must be frequent
- Push constraint checking into algorithms
  - Anti-monotone constraints
  - Monotone constraints

•Decrease the size of output
• Improve user guidance

Introduction

- Condensed representation of frequent itemsets
  - Extract a particular subset of the frequent itemset collection
  - The condensed subset is much smaller than the original collection
  - Can be extracted efficiently
  - The whole frequent itemsets can be regenerated

Introduction

- Main idea of the paper
  - Combine the above two approaches into one algorithm
  - This algorithm is based on the structure of Apriori

Content

- Introduction
- Constrained itemset mining
  - Apriori revisit
  - Anti-monotone constrains
  - Monotone constrains
  - Generic algorithm
- Frequent closed itemset mining
  - CLOSE algorithm
  - Incorporating constraints into Apriori
- Conclusion

Summary of paper

- Definition of constraints
  - T : transactional database
  - 2^{Items} : set of all itemsets
  - C : constraint
  - S : itemset, $S \in 2^{Items}$
  - I : subset of 2^{Items}
  - S satisfies C in T *iff* $C(S, T) = ture$
  - $SAT_C(I) = \{ S \in I, S \text{ satisfies } C \}$
  - SAT_C denotes $SAT_C(2^{Items})$

Summary of paper

TID	Items
1	ABCD
2	AC
3	AC
4	ABCD
5	BC
6	ABC

Itemset	Support	Frequency
A	1,2,3,4,6	0.83
B	1,4,5,6	0.67
AB	1,4,5	0.5
AC	1,2,3,4,6	0.83
CD	1,4	0.33
ACD	1,4	0.33

$C_{freq}(S) \equiv F(S) \geq r$: an itemset must be at least r frequent.

r = 0,6 $SAT_{C_{freq}} = \{A, B, C, AC, BC\}$

$C_{size}(S) \equiv |S| \leq 2$ and $C_{miss}(S) \equiv B \notin S$, then

$SAT_{C_{size} \wedge C_{miss}} = \{A, C, D, AC, AD, CD\}$, $SAT_{C_{size} \wedge C_{miss} \wedge C_{freq}} = \{A, C, AC\}$

Summary of paper

- Constrained itemset mining
  - T : transactional database
  - C : constraint
  - Computation of the collection of itemsets that satisfy C together with their frequecies
  - $R_C = \{(S, F(S)), S \in SAT_C\}$
- Use Apriori for constrained itemset mining where C is C_{freq}

Summary of paper - Apriori

- Apriori Algor…
  - 1. $C_1^g :=$ …
  - 2. $k := 1$
  - 3. *while* C_k^g …
  - 4. C_k :=safe-…
  - 5. $L_k := SAT_{C_{freq}}(C_k)$
  - 6. $C_{k+1}^g := generate_{apriori}(L_k)$
  - 7. $k := k+1$
  - 8. $U_{i=0}^{k-1} L_i$

Phase 1 – Candidate safe pruning
Eliminate candidates for which a subset of length k

Phase 3 – candidate generation for level k+1, fuse two elements that share the same k-1 first items

The completeness of Apriori relies on the anti-monotonicity of the constraint

$generate_{apriori}(L_k) = \{A \cup B,$ where $A, B \in L_k,$ A and B share the k-1 first items(in lexicographic order)$\}$

Anti-monotone constraints

- Definition: an anti-monotone constraint is a constraint C such that for all itemsets S, S':

  $(S' \subseteq S \wedge S \text{ satisfy } C) \Rightarrow S' \text{ satisfy } C$
- If S does not satisfy C_{am}, every superset of S does not satisfy C_{am}
- Example: $sum(S, price) \le v$
- A disjunction or conjunction of anti-monotone constraints is an anti-monotone constraint

Anti-monotone constraints

- Apriori can be changed:
  - Let C_{am} be an anti-monotone constraint. St… Apriori is replaced by $L_k := SAT_{C_{am}}(C_k)$ it is still correct and complete.
- Apriori can be used to mine constrained itemsets when the given constraint is anti-monotone

What about monotone constraints?

Monotone Constraints

- Definition
  - $S \in Items, C_m(S)$ is true $\Rightarrow \forall S' \supset S, C_m(S')$ is true
- Example
  - $sum(S, price) \ge v$
- Given a monotone constraint C_m, simply replacing Step 5 in Apriori with $L_k := SAT_{C_m}(C_k)$ leads to the loss of the completeness of Apriori.

Monotone Constraints

- Example
  - Assume $C(S) \equiv C \in S$. Itemset ABC generated by $generate_{apriori}$ from AB since $C(AB) = false$, ACB is not gen $C(ABC) = true$
  - Assume $C(S) \equiv A \in S$. Itemset ABC is *correctly* generated by $generate_{apriori}$ from AB and AC but since $C(AB) = false$, ACB is *incorrectly* pruned whereas $C(ABC) = true$

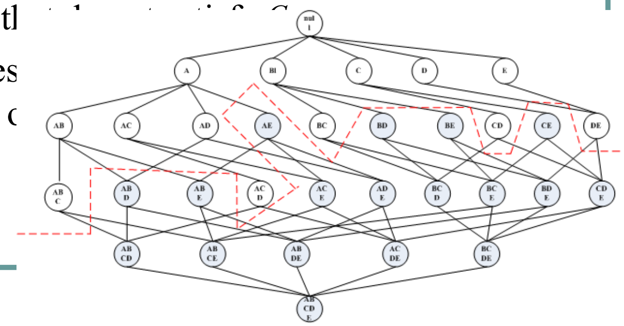
The generation step in Apriori must be complete: i.e., it must not miss any itemset satisfying C

The pruning step (Phase 1) must be correct, i.e., it must not prune an itemset that verify C

The generation step and pruning step need to be modified in order to include monotone constraints

Monotone Constraints

- Some definition in modified generation procedure
  - Negative border: If C_{am} denotes an anti-monotone constraint, $Bd_{\bar{C}_{am}}$ is the collection of the minimal itemsets tl
  - C_m denotes negation c



Monotone Constraints

- Generation procedure
  - $generate_1(L_k) = \{A \cup B, where\ A \in L_k$ and B is a 1-itemset $\}$
  - $generate_2(L_k) = \{A \cup B, where\ A,B \in L_k\}$
  - *Assume* $C = C_{am} \wedge \neg C'_{am}$ *and* $ms = Max_{S \in Bd_{\bar{C}'_{am}}} |S|$
  - $generate_m$

    $generate_m(L_0) = Bd_{\bar{C}'_{am}} \cap Items_1$

    For $k \geq 1$,
    - *If* $k<ms$, $generate_m(L_k) = generate_1(L_k) \cup (B$
    - *If* $k=ms$, $generate_m(L_k) = generate_1(L_k)$
    - *If* $k>ms$, $generate_m(L_k) = generate_2(L_k)$
- This generation procedure is complete d ensures that every candidate itemset verifies $\neg C'_{am}$ (C_m)

- C_{am} denotes an anti-monotone constraint
- $\neg C'_{am}$ denotes a monotone constraint

e

at do

We do not need to verify the monotone constraint after this generation procedure

Monotone Constraints

- Pruning procedure $prune_m$
  - For all $S \in C^g_{k+1}$ and for all $S' \subset S$ such that $|S'|=k$ do if $S' \notin L_k$ and $\boxed{C_m(S') = true}$ then delete S from C^g_{k+1}
- $prune_m$ is correct and complete

The algorithm is correct because it does not prune any itemset that verify $C = C_{am} \wedge \neg C'_{am}$ .Its completeness means that if an itemset is not pruned then every proper subset of that itemset verify C_{am} .

Generic Algorithm

- For a constraint $C = C_{am} \wedge C_m = C_{am} \wedge \neg C'_{am}$, the generic algorithm uses the structure of Apriori and the procedures $generate_m$ and $pruning_m$
  - 1. $C_1^g := Bd_{\bar{C}'_{am}} \cap Items_1; L_0 = \{\varnothing\}$
  - 2. $k:=1$
  - 3. while $C_k^g \neq \varnothing$ do
  - 4. Phase 1 – candidate safe pruning
    $C_k := pruning_m(C_k^g, L_{k-1})$
  - 5. Phase 2 - anti-monotone constraint checking
    $L_k := SAT_{C_{am}}(C_k)$
  - 6. Phase 3 – candidate generation for level k+1
    $C_{k+1}^g := generate_m(L_k)$
  - 7. $k:=k+1$
  - 8. output $\cup_{i=0}^{k-1} L_i$

Apriori Algorithm

1. $C_1^g := Items_1; L_0 = \{\varnothing\}$
2. $k:=1$
3. while $C_k^g \neq \varnothing$ do
4. Phase 1–candidate safe pruning
   $C_k := safe-pruning-on(C_k^g, L_{k-1})$
5. Phase 2-frequency constraint
   $L_k := SAT_{C_{freq}}(C_k)$
6. Phase 3-candidate generate
   $C_{k+1}^g := generate_{apriori}(L_k)$
7. $k:=k+1$
8. Output $\cup_{i=0}^{k-1} L_i$

Generic Algorithm-example

1	ABCD
2	AC
3	AC
4	ABCD
5	BC
6	ABC

- Constraints: $C_{am} \equiv A \notin S, C_m \equiv B \in S$
- $Bd_{\bar{C}_{am}} = \{B, AB\}$
- $ms = Max_{S \in Bd_{\bar{C}_{am}}} |S| = 2$
- $C_1 = \{B\}$
- $L_1 = \{B\}$
- $C_2 = \{AB, BC, BD\} \cup \{AB\} = \{AB, BC, BD\}$
- $L_2 = \{BC, BD\}$
- $C_3 = \{ABC, BCD, ABD\}$
- $L_3 = \{BCD\}$

{B,BC,BD,BCD}

- $generate_1(L_k) = \{A \bigcup B, where\ A \in L_k$ and B is 1-itemset}
- $generate_2(L_k) = \{A \bigcup B, where$ A,B$\in L_k$ }
- k<ms, $generate_m(L_k) = generate_1(L_k) \cup (Bd_{\bar{C}'_{am}} \cap Items_{k+1})$
- k=ms, $generate_m(L_k) = generate_1(L_k)$
- k<ms, $generate_m(L_k) = generate_2(L_k)$

For all $S \in C_{k+1}^g$ and for all $S' \subset S$ such that |S'|=k
do if $S' \notin L_k$ and $\boxed{C_m(S') = true}$
then delete S from C_{k+1}^g

Content

- Introduction
- Constrained itemset mining
  - Apriori revisit
  - Anti-monotone constrains
  - Monotone constrains
  - Generic algorithm
- Frequent closed itemset mining
  - CLOSE algorithm
  - Incorporating constraints into Apriori
- Conclusion

CLOSE algorithm-
frequent closed itemset mining

- The closure of an itemset *S(closure(S))* is the maximal superset of *S* which has the same support as *S*.
- A closed itemset is an itemset that is equal to its closure
- The set of closed itemset is a lattice called the closed itemset lattice

CLOSE algorithm

- We can consider CLO[SE as an instance] of the classical itemset [mining problem with a specific] constraint

> We can use this constraint in our generic algorithm together with other constraints to achieve constrained free-set mining

- A constraint for CLOSE

$$C'_{Free}(S) \equiv S' \subset S \Rightarrow S \not\subseteq closure(S')$$

- Free itemsets: itemsets that are not included in any closure of their proper sub-set. Equivalently, free itemsets are itemsets that verify C'_{Free}

CLOSE algorithm

> The closure of an itemset $S(closure(S))$ is the maximal superset of S which has the same support as S

TID	Items
1	ABCD
2	AC
3	AC
4	ABCD
5	BC
6	ABC

- Example
  - *Closure(AB)*: items A and B are simultaneously in transactions *1,4,6*. Item C is the only other item that is also present in these three transactions, thus *closure(AB)=ABC*.
  - *Closure(A)=AC, Closure(B)=BC,* $AB \not\subseteq closure(A)$ and $AB \not\subseteq closure(B)$. Therefore $C'_{Free}(AB)$ is true.
  - If frequency threshold r = ½, $SAT_{C_{freq} \wedge C'_{Free}} = \{\emptyset^C, A^C, B^C, D^{AC}, AB^C\}$ where AB^C means that $C'_{Free}(AB) = true, closure(AB) = ABC$

CLOSE algorithm

- The C'_{Free} constraint is anti-monotone, it needs a database pass to be checked
- Checking this constraint seems expensive if the closure of every subset of S has to be computed

> •We can use an equivalent constraint $C_{Free}(S) \equiv (S' \subset S \wedge |S'| = |S| - 1) \Rightarrow S \not\subseteq closure(S')$
> The equivalence means that $C'_{Free}(S)$ is true *iff* $C_{Free}(S)$ *is true.*
> •We need the closure of every subset of S of size $|S|$-*1*, then check if $S \subseteq closure(S')$

Incorporating constraints into Apriori

- Directly using $C = C_{Free} \wedge C_{am} \wedge C_m$ causes two problems
  - The closures of some candidates of level k are not computed => impossible to check C_{Free} at level $k+1$
  - $SAT_{C_{Free} \wedge C_{am} \wedge C_m}$ will no longer enables to compute $SAT_{C_{am} \wedge C_m}$

Incoporating constraints

- Assume we replace
  - C'_{Free} with $C'_{Free \wedge C_m}(S) \equiv (S' \subset S \wedge C_m(S')) \Rightarrow S \not\subseteq closure(S')$
  - C_{Free} with $C_{Free \wedge C_m}(S) \equiv (S' \subset S \wedge |S'| = |S| - 1 \wedge C_m(S')) \Rightarrow S \not\subseteq closure(S')$
- Then: the constraints $C_{Free \wedge C_m}$ and $C'_{Free \wedge C_m}$ are equivalent and anti-monotone. The set $SAT_{C_{am} \wedge C_m}$ can be efficiently computed using the same method as in CLOSE using $SAT_{C_{am} \wedge C_m \wedge C_{Free \wedge C_m}}$

  i.e., the output of the generic algorithm with the constraint

  $C = C_{Free \wedge C_m} \wedge C_{am} \wedge C_m$

Now we can find free-itemsets that verify conjunctions of anti-monotone and monotone constraints ☺

Content

- Introduction
- Constrained itemset mining
  - Apriori revisit
  - Anti-monotone constrains
  - Monotone constrains
  - Generic algorithm
- Frequent closed itemset mining
  - CLOSE algorithm
  - Incorporating constraints into Apriori
- Conclusion

Conclusion

- Frequent itemset mining can be intractable for a given support threshold and a particular database
- Two issues to address this problem: constraint-based itemset mining and condensed representation of frequent itemsets
- The generic algorithm can be used to achieve constrained free-set mining when $C = C_{am} \wedge C_m$