

# **723A75 Advanced Data Mining**

## **TDDD41 Data Mining - Clustering and Association Analysis**

### Lecture 6: Apriori Algorithm

---

Johan Alenlöv  
IDA, Linköping University, Sweden

- Content
  - Association Rules
  - Frequent Itemsets
  - Apriori Algorithm
  - Exercise
  - Rule Generation Algorithm
  - Exercise
  - Summary
- Literature
  - Course Book. 2nd ed.: 5.2.1–2, 5.4. 3rd ed.: 6.2.1–2, 6.4
  - Agrawal, R and Srikant, R. **Fast Algorithms for Mining Association Rules**. In Proc. of the 20th Int. Conf. on Very Large Databases, 1994.

# Association Rules

- Assume that we have access to some transactions

Transaction	Items
1	A, B, D
2	A, C, D
3	A, D, E
4	B, E, F
5	B, C, D, E, F

- Assume that the items are **sorted**.
- Want to find association rules,

$$\text{Item}_1, \dots, \text{Item}_m \rightarrow \text{Item}_{m+1}, \dots, \text{Item}_n.$$

If the items in the antecedent are purchased, so are the items in the consequent,  
e.g.

$$\text{Bread, Butter} \rightarrow \text{Cheese}$$

- Application: Market basket analysis to support business decisions,
  - Rules with "cheese" in the consequent may help decide how to boost sales of "cheese".
  - Rules with "bread" in the antecedent may help determine what happens in "bread" is sold out.
- Note that rules **do not** convey causality.

## Association Rules

- We are interested in finding rules of the form

$$X_1, \dots, X_m \rightarrow Y_1, \dots, Y_n \equiv X \rightarrow Y$$

- All rules **are not** equally interesting.
- Interested in rules with minimum **support** and **confidence**

**Support** Fraction of the transactions which contain  $X$  and  $Y$  ( $p(X, Y)$ )

**Support** "How general the rule is."

**Confidence** Fraction of the transactions that contain  $X$  which also contain  $Y$   
( $p(Y|X)$ )

**Confidence** "How accurate the rules is."

- confidence =  $p(Y|X) = p(X, Y)/p(X) = \text{support}(X, Y)/\text{support}(X)$ .
- Assume that we have access to some transactions

Transaction	Items
1	A, B, D
2	A, C, D
3	A, D, E
4	B, E, F
5	B, C, D, E, F

- $A \rightarrow D$  has support 0.6 and confidence 1.
- $D \rightarrow A$  has support 0.6 and confidence 0.75.

- We are interested in finding rules of the form

$$X_1, \dots, X_m \rightarrow Y_1, \dots, Y_n \equiv X \rightarrow Y$$

with user-specified minimum **support** and **confidence**

- We define a **frequent** or **large** itemset as a set of items that has minimum support.
  - E.g.,  $\{A, D\}$  is a frequent itemset in the previous example with minimum support 0.5.
- We will find the desired rules in two steps:
  1. Find all **frequent itemsets** (using apriori or FP grow algorithm).
  2. Generate all rules with minimum confidence from the frequent itemsets.
- The first step above will use the following **apriori property**:
  - Every subset of a frequent itemset is frequent.
  - Alternatively, every superset of an infrequent itemset is infrequent.

# Apriori Algorithm

**Algorithm:** apriori( $D$ ,  $minsup$ )

**Input:** A transactional database  $D$  and the minimum support  $minsup$ .

**Output:** All the large itemsets in  $D$ .

```
1  $L_1 = \{ \text{large 1-itemsets} \}$ 
2 for ( $k = 2; L_{k-1} \neq \emptyset; k++$ ) do
3    $C_k = \text{apriori-gen}(L_{k-1})$  // Generate candidate large  $k$ -itemsets
4   for all  $t \in D$  do
5     for all  $c \in C_k$  such that  $c \in t$  do
6        $c.count++$ 
7    $L_k = \{c \in C_k | c.count \geq minsup\}$ 
8 return  $\bigcup_k L_k$ 
```

**Algorithm:** apriori-gen( $L_{k-1}$ )

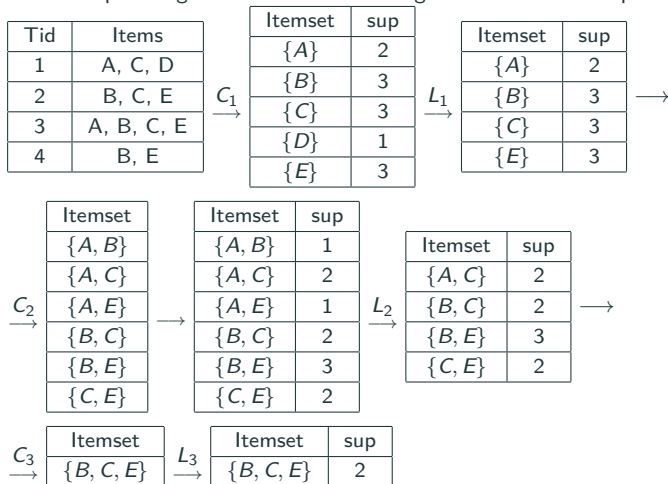
**Input:** Large  $(k-1)$ -itemsets.

**Output:** A superset of  $L_k$ .

```
1  $C_k = \emptyset$  // Self-join
2 for all  $I, J \in L_{k-1}$  do
3   if  $I_1 = J_1, \dots, I_{k-2} = J_{k-2}$  and  $I_{k-1} < J_{k-1}$  then
4     add  $\{I_1, \dots, I_{k-1}, J_{k-1}\}$  to  $C_k$ 
5 for all  $c \in C_k$  do // Prune
6   for all  $(k-1)$ -subsets  $s$  of  $c$  do
7     if  $s \notin L_{k-1}$  then
8       remove  $c$  from  $C_k$ 
9 return  $C_k$ 
```

## Example: Apriori Algorithm

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



- Self-join step in MySQL:

```
insert into Ck
select l.item1, ..., l.itemk-1, J.itemk-1
from Lk-1 l, Lk-1 J
where l.item1 = J.item1, ..., l.itemk-2 = J.itemk-2, l.itemk-1 < J.itemk-1
```

- Self-join step in R:

```
merge(Lk-1, Lk-1, by=c(Lk-1.item1, ..., Lk-1.itemk-2))
```

Note that **duplicates** will be produced because the condition  $l.item_{k-1} < J.item_{k-1}$  is **not** enforced.

- To make the prune step fast, store the results in a **hash table**.
- Clever data structures are typically used for counting the support. (line 4-6 in apriori algorithm)



- Run the apriori algorithm on the database below with minimum support 2.

Tid	Items
1	A, B, C
2	A, B, C, D, E
3	A, C, D
4	A, C, D, E
5	A, B, C, D

- Show the execution details (i.e. self-join, prune, support counting) not just the large itemsets.

$$L_1 = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\}$$

$$L_2 = \{\{A, B\}, \{A, C\}, \{A, D\}, \{A, E\}, \{B, C\}, \{B, D\}, \{C, D\}, \{C, E\}, \{D, E\}\}$$

$$L_3 = \{\{A, B, C\}, \{A, B, D\}, \{A, C, D\}, \{A, C, E\}, \{A, D, E\}, \{B, C, D\}, \{C, D, E\}\}$$

$$L_4 = \{\{A, B, C, D\}, \{A, C, D, E\}\}$$

# Apriori Algorithm Proof

**Algorithm:** apriori( $D$ ,  $minsup$ )

**Input:** A transactional database  $D$  and the minimum support  $minsup$ .

**Output:** All the large itemsets in  $D$ .

```
1  $L_1 = \{ \text{large 1-itemsets} \}$ 
2 for ( $k = 2; L_{k-1} \neq \emptyset; k++$ ) do
3    $C_k = \text{apriori-gen}(L_{k-1})$  // Generate candidate large  $k$ -itemsets
4   for all  $t \in D$  do
5     for all  $c \in C_k$  such that  $c \in t$  do
6        $c.count++$ 
7    $L_k = \{c \in C_k \mid c.count \geq minsup\}$ 
8 return  $\bigcup_k L_k$ 
```

**Algorithm:** apriori-gen( $L_{k-1}$ )

**Input:** Large  $(k-1)$ -itemsets.

**Output:** A superset of  $L_k$ .

```
1  $C_k = \emptyset$  // Self-join
2 for all  $I, J \in L_{k-1}$  do
3   if  $I_1 = J_1, \dots, I_{k-2} = J_{k-2}$  and  $I_{k-1} < J_{k-1}$  then
4     add  $\{I_1, \dots, I_{k-1}, J_{k-1}\}$  to  $C_k$ 
5 for all  $c \in C_k$  do // Prune
6   for all  $(k-1)$ -subsets  $s$  of  $c$  do
7     if  $s \notin L_{k-1}$  then
8       remove  $c$  from  $C_k$ 
9 return  $C_k$ 
```

- We prove by induction on  $k$  that the algorithm is enough.
- $k = 1$  is trivial.
- Induction hypothesis: Assume that the algorithm is correct up to  $k - 1$ .  
We now want to prove that the algorithm is correct for  $k$ .  
It is enough to show that  $L_k \subseteq C_k$ .
- We perform a proof by contradiction. Assume that  $I \in L_k$  but  $I \notin C_k$ . Then,
  - $\{I_1, I_2, \dots, I_{k-2}, I_{k-1}\} \in L_{k-1}$  follows from  $I \in L_k$  by the apriori property and the induction hypothesis.
  - $\{I_1, I_2, \dots, I_{k-2}, I_k\} \in L_{k-1}$  follows from  $I \in L_k$  by the apriori property and the induction hypothesis.
  - Then  $I \in C_k$  by the self-join step.
  - Since  $I \in L_k$ , every subset of  $I$  is large by the apriori property.
  - Thus  $I$  will **not** be removed in the prune-step and  $I \in C_k$
  - This is a contradiction and thus the algorithm is correct for  $k$ .

# Rule Generation Algorithm

- Having a large itemset  $L$  we wish to generate rules of the form

$$X \rightarrow L \setminus X,$$

where  $X \subseteq L$ .

- These rules should have a minimum confidence.
- The following apriori property will be used:
  - If  $X$  does not result in a rule with minimum confidence for  $L$ , then neither does **any subset**  $X' \subseteq X$ ,

$$\text{confidence}(X \rightarrow L \setminus X) = \frac{\text{support}(L)}{\text{support}X} \geq \frac{\text{support}(L)}{\text{support}(X')} = \text{confidence}(X' \rightarrow L \setminus X')$$

- for all large itemsets  $l_k$  with  $k \geq 2$  do
- call  $\text{genrules}(l_k, l_k, \text{minconf})$

**Algorithm:**  $\text{genrules}(l_k, a_m, \text{minconf})$

**Input:** A large itemset  $l_k$ , a set  $a_m \subseteq l_k$ , the minimum confidence  $\text{minconf}$ .

**Output:** All the rules of the form  $a \rightarrow l_k \setminus a$  with  $a \subseteq a_m$  and confidence equal or above  $\text{minconf}$ .

- $\mathbb{A} = \{(m-1)\text{-itemsets } a_{m-1} \mid a_{m-1} \subseteq a_m\}$
- for all  $a_{m-1} \in \mathbb{A}$  do
- conf =  $\text{support}(l_k) / \text{support}(a_{m-1})$  // Confidence of the rule  $a_{m-1} \rightarrow l_k \setminus a_{m-1}$
- if conf  $\geq \text{minconf}$  then
- output the rule  $a_{m-1} \rightarrow l_k \setminus a_{m-1}$  with confidence=conf and support=support( $l_k$ )
- if  $m-1 > 1$  then call  $\text{genrules}(l_k, a_{m-1}, \text{minconf})$

- Run the genrule algorithm on the database below for the large itemset  $\{A, B, C\}$  with minimum confidence 0.8.

Tid	Items
1	A, B, C
2	A, B, C, D, E
3	A, C, D
4	A, C, D, E
5	A, B, C, D

- Show the execution details (i.e. antecedent generation, recursive calls) not just the rules.

$$A, B \rightarrow C$$

$$B, C \rightarrow A$$

$$B \rightarrow A, C$$

# Rule Generation Algorithm Proof

```
1  for all large itemsets  $l_k$  with  $k \geq 2$  do
2    call  $\text{genrules}(l_k, l_k, \text{minconf})$ 

Algorithm:  $\text{genrules}(l_k, a_m, \text{minconf})$ 
Input: A large itemset  $l_k$ , a set  $a_m \subseteq l_k$ , the minimum confidence  $\text{minconf}$ .
Output: All the rules of the form  $a \rightarrow l_k \setminus a$  with  $a \subseteq a_m$  and confidence equal or above  $\text{minconf}$ .

1   $\mathbb{A} = \{(m-1)\text{-itemsets } a_{m-1} \mid a_{m-1} \subseteq a_m\}$ 
2  for all  $a_{m-1} \in \mathbb{A}$  do
3     $\text{conf} = \text{support}(l_k) / \text{support}(a_{m-1})$            // Confidence of the rule  $a_{m-1} \rightarrow l_k \setminus a_{m-1}$ 
4    if  $\text{conf} \geq \text{minconf}$  then
5      output the rule  $a_{m-1} \rightarrow l_k \setminus a_{m-1}$  with confidence= $\text{conf}$  and support= $\text{support}(l_k)$ 
6      if  $m-1 > 1$  then call  $\text{genrules}(l_k, a_{m-1}, \text{minconf})$ 
```

- We prove by contradiction that the rule generation algorithm is correct.
- Assume that the algorithm missed a rule. Let  $a_{m-1} \rightarrow l_k \setminus a_{m-1}$  denote one of the missing rules with the largest antecedent. Then,
  - Note that  $l_k$  has minimum support and, thus, it is outputted by the apriori algorithm since this is correct.
  - Then, the rule generation algorithm cannot have missed the rule if  $m = k$ .
  - Moreover if  $m < k$ , then

$$\begin{aligned} \text{confidence}(a_m \rightarrow l_k \setminus a_m) &= \text{support}(l_k) / \text{support}(a_m) \geq \text{support}(l_k) / \text{support}(a_{m-1}) \\ &= \text{confidence}(a_{m-1} \rightarrow l_k \setminus a_{m-1}) \geq \text{minconf}. \end{aligned}$$

- Note that the algorithm didn't miss the rule  $a_m \rightarrow l_k \setminus a_m$ .
- Then the algorithm couldn't have missed the rule  $a_{m-1} \rightarrow l_k \setminus a_{m-1}$ .
- This contradicts our assumption and, thus, the algorithm is correct.

- Processing transactions to find rules of the form,

$$\text{Item}_1, \dots, \text{Item}_m \rightarrow \text{Item}_{m+1}, \dots, \text{Item}_n,$$

with a user-defined minimum support and confidence.

- We use a two-step solution:
  1. Find all the **large itemsets**.
  2. Generate all the **rules** with minimum confidence.
- We use the **apriori properties**.
- Drawbacks of the apriori algorithm:
  - Candidate generate-and-test.
  - Too many candidates to generate, e.g. if there are  $10^4$  large 1-itemsets, then more than  $10^7$  candidate 2-itemsets.
  - Each candidate implies expensive operations, e.g. pattern matching, subset checking, storing.
- Can candidate generation be avoided?