

ICCS361 - Week 4

Association Rules & Frequent Itemsets

Sunsern Cheamanunkul

Ref Credit: Association Rules & Sequential Patterns by Bing Liu, UIC



Mahidol University
International College



Today

- Association Rules Mining
- Frequent Itemsets
- Apriori Algorithm

Association rule mining

- Proposed by **Agrawal et al in 1993**.
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are categorical.
- No good algorithm for numeric data.
- Initially used for **Market Basket Analysis** to find how items purchased by customers are related.

Bread \rightarrow Milk [sup = 5%, conf = 100%]

The model: data

- $I = \{i_1, i_2, \dots, i_m\}$: a set of *items*.
- Transaction t :
 - t a set of items, and $t \subseteq I$.
- Transaction Database T : a set of transactions
 $T = \{t_1, t_2, \dots, t_n\}$.

Transaction data: supermarket data

- Market basket transactions:

t1: {bread, cheese, milk}

t2: {apple, eggs, salt, yogurt}

...

...

tn: {biscuit, eggs, milk}

- Concepts:

- *An item*: an item/article in a basket
- *I*: the set of all items sold in the store
- *A transaction*: items purchased in a basket; it may have TID (transaction ID)
- *A transactional dataset*: A set of transactions

Transaction data: a set of documents

- **A text document data set. Each document is treated as a “bag” of keywords**

| | |
|-------|-------------------------------|
| doc1: | Student, Teach, School |
| doc2: | Student, School |
| doc3: | Teach, School, City, Game |
| doc4: | Baseball, Basketball |
| doc5: | Basketball, Player, Spectator |
| doc6: | Baseball, Coach, Game, Team |
| doc7: | Basketball, Team, City, Game |

The model: rules

- A transaction t contains X , a set of items (itemset) in I , if $X \subseteq t$.
- An association rule is an implication of the form:
$$X \rightarrow Y, \text{ where } X, Y \subset I, \text{ and } X \cap Y = \emptyset$$
- An itemset is a set of items.
 - E.g., $X = \{\text{milk, bread, cereal}\}$ is an itemset.
- A k -itemset is an itemset with k items.
 - E.g., $\{\text{milk, bread, cereal}\}$ is a 3-itemset

Rule strength measures

- **Support:** The rule holds with **support** sup in T (the transaction data set) if $sup\%$ of transactions contain $X \cup Y$.
 - $sup = \Pr(X \cup Y)$.
- **Confidence:** The rule holds in T with **confidence** $conf$ if $conf\%$ of transactions that contain X also contain Y .
 - $conf = \Pr(Y | X)$
- An association rule is a pattern that states when X occurs, Y occurs with certain probability.

Support and Confidence

- **Support count:** The support count of an itemset X , denoted by $X.count$, in a data set T is the number of transactions in T that contain X . Assume T has n transactions.
- Then,

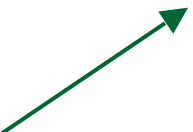
$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Goal and key features

- **Goal:** Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).
- **Key Features**
 - ❑ **Completeness:** find all rules.
 - ❑ **No target item(s)** on the right-hand-side
 - ❑ Mining with data on **hard disk** (not in memory)

An example



t1: Beef, Chicken, Milk
 t2: Beef, Cheese
 t3: Cheese, Boots
 t4: Beef, Chicken, Cheese
 t5: Beef, Chicken, Clothes, Cheese, Milk
 t6: Chicken, Clothes, Milk
 t7: Chicken, Milk, Clothes

- Transaction data

- Assume:

minsup = 30%

minconf = 80%

- An example **frequent itemset**:

{Chicken, Clothes, Milk} [sup = 3/7]

- Association rules** from the itemset:

Clothes \rightarrow Milk, Chicken [sup = 3/7, conf = 3/3]

...

...

Clothes, Chicken \rightarrow Milk, [sup = 3/7, conf = 3/3]

Transaction data representation

- A simplistic view of shopping baskets,
- Some important information not considered.
E.g,
 - ❑ the quantity of each item purchased and
 - ❑ the price paid.

Many mining algorithms

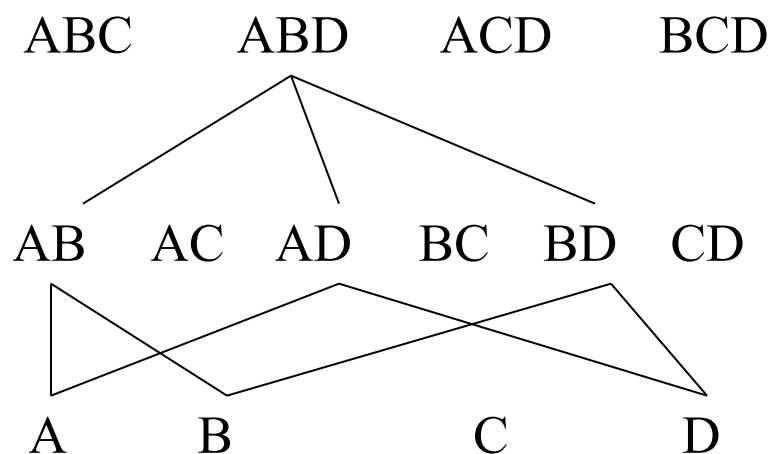
- There are a large number of them!!
- They use different strategies and data structures.
- Their resulting sets of rules are all the same.
 - Given a transaction data set T , and a minimum support and a minimum confident, the set of association rules existing in T is uniquely determined.
- Any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different.
- We study only one: the Apriori Algorithm

The Apriori algorithm

- **Two steps:**
 - Find all itemsets that have minimum support (*frequent itemsets*, also called large itemsets).
 - Use frequent itemsets to **generate rules**.
- E.g., a frequent itemset
 {Chicken, Clothes, Milk} [sup = 3/7]
and one rule from the frequent itemset
 Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

Step 1: Mining all frequent itemsets

- A **frequent *itemset*** is an itemset whose support is $\geq \text{minsup}$.
- **Key idea:** The apriori property (**downward closure property**): any subsets of a frequent itemset are also frequent itemsets



The Algorithm

- **Iterative algo.** (also called **level-wise search**):
Find all 1-item frequent itemsets; then all 2-item frequent itemsets, and so on.

- In each iteration k , only consider itemsets that contain some $k-1$ frequent itemset.

- Find frequent itemsets of size 1: F_1
- **From $k = 2$**
 - C_k = candidates of size k : those itemsets of size k that could be frequent, given F_{k-1}
 - F_k = those itemsets that are actually frequent, $F_k \subseteq C_k$ (need to scan the database once).

Example - Finding frequent itemsets

Dataset T
minsup=0.5

| TID | Items |
|------|------------|
| T100 | 1, 3, 4 |
| T200 | 2, 3, 5 |
| T300 | 1, 2, 3, 5 |
| T400 | 2, 5 |

itemset:count

1. scan T → C_1 : {1}:2, {2}:3, {3}:3, {4}:1, {5}:3

→ F_1 : {1}:2, {2}:3, {3}:3, {5}:3

→ C_2 : {1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}

2. scan T → C_2 : {1,2}:1, {1,3}:2, {1,5}:1, {2,3}:2, {2,5}:3, {3,5}:2

→ F_2 : {1,3}:2, {2,3}:2, {2,5}:3, {3,5}:2

→ C_3 : {2, 3, 5}

3. scan T → C_3 : {2, 3, 5}:2

→ F_3 : {2, 3, 5}

| TID | Items |
|------|------------|
| T100 | 1, 3, 4 |
| T200 | 2, 3, 5 |
| T300 | 1, 2, 3, 5 |
| T400 | 2, 5 |

Details: ordering of items

- The items in I are sorted in **lexicographic order** (which is a total order).
- The order is used throughout the algorithm in each itemset.
- $\{w[1], w[2], \dots, w[k]\}$ represents a k -itemset w consisting of items $w[1], w[2], \dots, w[k]$, where $w[1] < w[2] < \dots < w[k]$ according to the total order.

Details: the algorithm

Algorithm Apriori(T)

```

 $C_1 \leftarrow \text{init-pass}(T);$ 
 $F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq \text{minsup}\};$  //  $n$ : no. of transactions in  $T$ 
for ( $k = 2; F_{k-1} \neq \emptyset; k++$ ) do
     $C_k \leftarrow \text{candidate-gen}(F_{k-1});$ 
    for each transaction  $t \in T$  do
        for each candidate  $c \in C_k$  do
            if  $c$  is contained in  $t$  then
                 $c.\text{count}++;$ 
            end
        end
     $F_k \leftarrow \{c \in C_k \mid c.\text{count}/n \geq \text{minsup}\}$ 
end
return  $F \leftarrow \bigcup_k F_k;$ 

```

Apriori candidate generation

- The **candidate-gen** function takes F_{k-1} and returns a **superset** (called the candidates) of the set of all **frequent k -itemsets**. It has two steps
 - **join step**: Generate all possible candidate itemsets C_k of length k
 - **prune step**: Remove those candidates in C_k that cannot be frequent.

Candidate-gen function

Function candidate-gen(F_{k-1})

$C_k \leftarrow \emptyset;$

forall $f_1, f_2 \in F_{k-1}$

 with $f_1 = \{i_1, \dots, i_{k-2}, i_{k-1}\}$

 and $f_2 = \{i_1, \dots, i_{k-2}, i'_{k-1}\}$

 and $i_{k-1} < i'_{k-1}$ **do**

$c \leftarrow \{i_1, \dots, i_{k-1}, i'_{k-1}\};$

// join f_1 and f_2

$C_k \leftarrow C_k \cup \{c\};$

for each $(k-1)$ -subset s of c **do**

if $(s \notin F_{k-1})$ **then**

 delete c from $C_k;$

// prune

end

end

return $C_k;$

An example

- $F_3 = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 3, 5\}, \{2, 3, 4\}\}$
- After join
 - ▣ $C_4 = \{\{1, 2, 3, 4\}, \{1, 3, 4, 5\}\}$
- After pruning:
 - ▣ $C_4 = \{\{1, 2, 3, 4\}\}$

because $\{1, 4, 5\}$ is not in F_3 ($\{1, 3, 4, 5\}$ is removed)

Step 2: Generating rules from frequent itemsets

- Frequent itemsets \neq association rules
- One more step is needed to generate association rules
- For each frequent itemset X ,
For each proper nonempty subset A of X ,
 - Let $B = X - A$
 - $A \rightarrow B$ is an association rule if
 - Confidence($A \rightarrow B$) \geq minconf,
support($A \rightarrow B$) = support($A \cup B$) = support(X)
confidence($A \rightarrow B$) = support($A \cup B$) / support(A)

Generating rules: an example

- Suppose $\{2,3,4\}$ is frequent, with $\text{sup}=50\%$
 - Proper nonempty subsets: $\{2,3\}$, $\{2,4\}$, $\{3,4\}$, $\{2\}$, $\{3\}$, $\{4\}$, with $\text{sup}=50\%$, 50% , 75% , 75% , 75% , 75% respectively
 - These generate these association rules:
 - $2,3 \rightarrow 4$, confidence= 100%
 - $2,4 \rightarrow 3$, confidence= 100%
 - $3,4 \rightarrow 2$, confidence= 67%
 - $2 \rightarrow 3,4$, confidence= 67%
 - $3 \rightarrow 2,4$, confidence= 67%
 - $4 \rightarrow 2,3$, confidence= 67%
 - All rules have support = 50%

Generating rules: summary

- To recap, in order to obtain $A \rightarrow B$, we need to have $\text{support}(A \cup B)$ and $\text{support}(A)$
- All the required information for confidence computation has already been recorded in itemset generation. No need to see the data T any more.
- This step is not as time-consuming as frequent itemsets generation.

On Apriori Algorithm

Seems to be very expensive

- Level-wise search
- K = the size of the largest itemset
- It makes at most K passes over data
- In practice, K is bounded (10).

More on association rule mining

- Clearly the space of all association rules is **exponential**, $O(2^m)$, where m is the number of items in I .
- The mining exploits **sparseness of data**, and **high minimum support** and **high minimum confidence** values.
- Still, it always produces a **huge number of rules**, thousands, tens of thousands, millions, ...

Improving Apriori Algo.

- Further reduce the size of the candidate sets C_i for $i \geq 2$
- Simultaneously find F_1, F_2, F_3, \dots in one or two passes, rather than a pass per level.