Mining Association Rules in Large Databases
Association rules

- Given a set of transactions $D$, find rules that will predict the occurrence of an item (or a set of items) based on the occurrences of other items in the transaction.

Market-Basket transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Examples of association rules

- \{Diaper\} → \{Beer\},
- \{Milk, Bread\} → \{Diaper, Coke\},
- \{Beer, Bread\} → \{Milk\},
An even simpler concept: frequent itemsets

- Given a set of transactions $D$, find combination of items that occur frequently

### Market-Basket transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Examples of frequent itemsets

- $\{\text{Diaper, Beer}\}$
- $\{\text{Milk, Bread}\}$
- $\{\text{Beer, Bread, Milk}\}$
Lecture outline

• **Task 1:** Methods for finding all frequent itemsets efficiently

• **Task 2:** Methods for finding association rules efficiently
Definition: Frequent Itemset

- **Itemset**
  - A set of one or more items
    - E.g.: \{Milk, Bread, Diaper\}
  - \(k\)-itemset
    - An itemset that contains \(k\) items

- **Support count \((\sigma)\)**
  - Frequency of occurrence of an itemset (number of transactions it appears)
  - E.g. \(\sigma(\{\text{Milk, Bread, Diaper}\}) = 2\)

- **Support**
  - Fraction of the transactions in which an itemset appears
  - E.g. \(s(\{\text{Milk, Bread, Diaper}\}) = 2/5\)

- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a \textit{minsup} threshold

<table>
<thead>
<tr>
<th>(TID)</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>
Why do we want to find frequent itemsets?

• Find all combinations of items that occur together

• They might be interesting (e.g., in placement of items in a store 😊)

• Frequent itemsets are only positive combinations (we do not report combinations that do not occur frequently together)

• Frequent itemsets aims at providing a summary for the data
Finding frequent sets

- **Task:** Given a transaction database $D$ and a $\text{minsup}$ threshold find all frequent itemsets and the frequency of each set in this collection.

- **Stated differently:** Count the number of times combinations of attributes occur in the data. If the count of a combination is above $\text{minsup}$ report it.

- **Recall:** The input is a transaction database $D$ where every transaction consists of a subset of items from some universe $I$. 
How many itemsets are there?

Given \( d \) items, there are \( 2^d \) possible itemsets.
When is the task sensible and feasible?

• If \( \text{minsupt} = 0 \), then all subsets of \( I \) will be frequent and thus the size of the collection will be very large.

• This summary is very large (maybe larger than the original input) and thus not interesting.

• The task of finding all frequent sets is interesting typically only for relatively large values of \( \text{minsupt} \).
A simple algorithm for finding all frequent itemsets

null

A B C D E

AB AC AD AE BC BD BE CD CE DE

ABC ABD ABE ACD ACE ADE BCD BCE BDE CDE

ABCD ABCE ABDE ACDE BCDE

ABCDE
Brute-force algorithm for finding all frequent itemsets?

• Generate all possible itemsets (lattice of itemsets)
  – Start with 1-itemsets, 2-itemsets,...,d-itemsets

• Compute the frequency of each itemset from the data
  – Count in how many transactions each itemset occurs

• If the support of an itemset is above \textit{minsup} report it as a frequent itemset
Brute-force approach for finding all frequent itemsets

• Complexity?

  – Match every candidate against each transaction

  – For $M$ candidates and $N$ transactions, the complexity is $O(NMw)$ => Expensive since $M = 2^d$ !!!
Speeding-up the brute-force algorithm

• Reduce the number of candidates (M)
  – Complete search: M=2^d
  – Use pruning techniques to reduce M

• Reduce the number of transactions (N)
  – Reduce size of N as the size of itemset increases
  – Use vertical-partitioning of the data to apply the mining algorithms

• Reduce the number of comparisons (NM)
  – Use efficient data structures to store the candidates or transactions
  – No need to match every candidate against every transaction
Reduce the number of candidates

• Apriori principle (Main observation):
  – If an itemset is frequent, then all of its subsets must also be frequent

• Apriori principle holds due to the following property of the support measure:

\[ \forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y) \]

  – The support of an itemset *never exceeds* the support of its subsets
  – This is known as the *anti-monotone* property of support
## Example

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>s(Bread) &gt; s(Bread, Beer)</th>
<th>s(Milk) &gt; s(Bread, Milk)</th>
<th>s(Diaper, Beer) &gt; s(Diaper, Beer, Coke)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Illustrating the Apriori principle

Found to be Infrequent

Pruned supersets
Illustrating the Apriori principle

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bread</td>
<td>4</td>
</tr>
<tr>
<td>Coke</td>
<td>2</td>
</tr>
<tr>
<td>Milk</td>
<td>4</td>
</tr>
<tr>
<td>Beer</td>
<td>3</td>
</tr>
<tr>
<td>Diaper</td>
<td>4</td>
</tr>
<tr>
<td>Eggs</td>
<td>1</td>
</tr>
</tbody>
</table>

Items (1-itemsets)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Bread, Milk}</td>
<td>3</td>
</tr>
<tr>
<td>{Bread, Beer}</td>
<td>2</td>
</tr>
<tr>
<td>{Bread, Diaper}</td>
<td>3</td>
</tr>
<tr>
<td>{Milk, Beer}</td>
<td>2</td>
</tr>
<tr>
<td>{Milk, Diaper}</td>
<td>3</td>
</tr>
<tr>
<td>{Beer, Diaper}</td>
<td>3</td>
</tr>
</tbody>
</table>

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

minsup = 3/5

If every subset is considered, 
\[ 6C_1 + 6C_2 + 6C_3 = 41 \]

With support-based pruning, 
\[ 6 + 6 + 1 = 13 \]

Triplets (3-itemsets)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Bread, Milk, Diaper}</td>
<td>3</td>
</tr>
</tbody>
</table>
Exploiting the Apriori principle

1. Find frequent 1-items and put them to \( L_k \) (\( k=1 \))
2. Use \( L_k \) to generate a collection of candidate itemsets \( C_{k+1} \) with size \( (k+1) \)
3. Scan the database to find which itemsets in \( C_{k+1} \) are frequent and put them into \( L_{k+1} \)
4. If \( L_{k+1} \) is not empty
   - \( k=k+1 \)
   - Goto step 2

The Apriori algorithm

$C_k$: Candidate itemsets of size $k$

$L_k$: frequent itemsets of size $k$

$L_1 = \{\text{frequent 1-itemsets}\}$;

for $(k = 2; L_k \neq \emptyset; k++)$

$$C_{k+1} = \text{GenerateCandidates}(L_k)$$

for each transaction $t$ in database do

increment count of candidates in $C_{k+1}$ that are contained in $t$

endfor

$L_{k+1} = \text{candidates in } C_{k+1} \text{ with support } \geq \text{min\_sup}$

endfor

return $\bigcup_k L_k$;
GenerateCandidates

• Assume the items in $L_k$ are listed in an order (e.g., alphabetical)

• **Step 1: self-joining $L_k$ (IN SQL)**

  insert into $C_{k+1}$
  select $p.item_1, p.item_2, ..., p.item_k, q.item_k$
  from $L_k p, L_k q$
  where $p.item_1=q.item_1, ..., p.item_{k-1}=q.item_{k-1}, p.item_k < q.item_k$
Example of Candidates Generation

- $L_3 = \{abc, abd, acd, ace, bcd\}$

- **Self-joining**: $L_3 \star L_3$
  - $abcd$ from $abc$ and $abd$
  - $acde$ from $acd$ and $ace$
GenerateCandidates

- Assume the items in $L_k$ are listed in an order (e.g., alphabetical)

- **Step 1:** *self-joining* $L_k$ *(IN SQL)*
  
  insert into $C_{k+1}$
  
  select $p.item_1, p.item_2, ..., p.item_k, q.item_k$
  
  from $L_k p, L_k q$
  
  where $p.item_1 = q.item_1, ..., p.item_{k-1} = q.item_{k-1}, p.item_k < q.item_k$

- **Step 2:** *pruning*
  
  forall *itemsets* $c$ in $C_{k+1}$ do
  
  forall *$k$-subsets* $s$ of $c$ do
  
  if ($s$ is not in $L_k$) then delete $c$ from $C_{k+1}$
Example of Candidates Generation

- $L_3=\{abc, abd, acd, ace, bcd\}$

- **Self-joining:** $L_3 \ast L_3$
  - $abcd$ from $abc$ and $abd$
  - $acde$ from $acd$ and $ace$

- **Pruning:**
  - $acde$ is removed because $ade$ is not in $L_3$

- $C_4=\{abcd\}$
The Apriori algorithm

$C_k$: Candidate itemsets of size $k$
$L_k$: frequent itemsets of size $k$

$L_1 = \{\text{frequent items}\}$;

for $(k = 1; L_k \neq \emptyset; k++)$

\[
C_{k+1} = \text{GenerateCandidates}(L_k)
\]

for each transaction $t$ in database do

increment count of candidates in $C_{k+1}$ that are contained in $t$

endfor

$L_{k+1} = \text{candidates in } C_{k+1} \text{ with support } \geq min\_sup$

endfor

return $\bigcup_k L_k$;
How to Count Supports of Candidates?

• Naive algorithm?

  – Method:
    – Candidate itemsets are stored in a hash-tree
    – *Leaf node* of hash-tree contains a list of itemsets and counts
    – *Interior node* contains a hash table
    – *Subset function*: finds all the candidates contained in a transaction
Example of the hash-tree for $C_3$

Hash function: mod 3

Hash on 1st item

Hash on 2nd item

Hash on 3rd item
Example of the hash-tree for $C_3$

Hash function: mod 3

Hash on 1st item

Hash on 2nd item

Hash on 3rd item
Example of the hash-tree for $C_3$

Hash function: mod 3

The subset function finds all the candidates contained in a transaction:
- At the root level it hashes on all items in the transaction
- At level $i$ it hashes on all items in the transaction that come after item the $i$-th item
Discussion of the Apriori algorithm

• Much faster than the Brute-force algorithm
  – It avoids checking all elements in the lattice

• The running time is in the worst case $O(2^d)$
  – Pruning really prunes in practice

• It makes multiple passes over the dataset
  – One pass for every level $k$

• Multiple passes over the dataset is inefficient when we have thousands of candidates and millions of transactions
Making a single pass over the data: the AprioriTid algorithm

- The database is **not** used for counting support after the 1\textsuperscript{st} pass!

- Instead information in data structure $C_k'$ is used for counting support in every step

  - $C_k' = \{<\text{TID}, \{X_k\}> | X_k \text{ is a potentially frequent } k\text{-itemset in transaction with id=\text{TID}>}$

  - $C_1'$: corresponds to the original database (every item $i$ is replaced by itemset $\{i\}$)

  - The member $C_k'$ corresponding to transaction $t$ is $<t.\text{TID}, \{c \in C_k | c \text{ is contained in } t}>$
The AprioriTID algorithm

- \( L_1 = \{ \text{frequent 1-itemsets} \} \)
- \( C_1' = \text{database D} \)
- \( \text{for } (k=2, L_{k-1}' \neq \text{empty}; k++) \)
  \( C_k = \text{GenerateCandidates}(L_{k-1}) \)
  \( C_k' = {} \)
  \( \text{for all entries } t \in C_{k-1}' \)
  \( C_t = \{ c \in C_k | t[c-c[k]] = 1 \text{ and } t[c-c[k-1]] = 1 \} \)
  \( \text{for all } c \in C_t \{ c.\text{count}++ \} \)
  \( \text{if } (C_t \neq \{\}) \)
    \( \text{append } C_t \text{ to } C_k' \)
  \( \text{endif} \)
  \( \text{endfor} \)
  \( L_k = \{ c \in C_k | c.\text{count} \geq \text{minsup} \} \)
  \( \text{endfor} \)
- \( \text{return } U_k L_k \)
AprioriTid Example (minsup=2)

Database D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

C′₁

<table>
<thead>
<tr>
<th>TID</th>
<th>Sets of itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{{1},{3},{4}}</td>
</tr>
<tr>
<td>200</td>
<td>{{2},{3},{5}}</td>
</tr>
<tr>
<td>300</td>
<td>{{1},{2},{3},{5}}</td>
</tr>
<tr>
<td>400</td>
<td>{{2},{5}}</td>
</tr>
</tbody>
</table>

L₁

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
</tbody>
</table>

C′₂

<table>
<thead>
<tr>
<th>TID</th>
<th>Sets of itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{{1 3}}</td>
</tr>
<tr>
<td>200</td>
<td>{{2 3},{2 5},{3 5}}</td>
</tr>
<tr>
<td>300</td>
<td>{{1 2},{1 3},{1 5},{2 3},{2 5},{3 5}}</td>
</tr>
<tr>
<td>400</td>
<td>{{2 5}}</td>
</tr>
</tbody>
</table>

L₂

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

C′₃

<table>
<thead>
<tr>
<th>TID</th>
<th>Sets of itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>{{2 3 5}}</td>
</tr>
<tr>
<td>300</td>
<td>{{2 3 5}}</td>
</tr>
</tbody>
</table>

L₃

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>
Discussion on the AprioriTID algorithm

- \( L_1 = \{\text{frequent 1-itemsets}\} \)
- \( C_1' = \text{database D} \)
- **for** (k=2, \( L_{k-1}' \neq \text{empty}; k++ \)
  - \( C_k = \text{GenerateCandidates}(L_{k-1}) \)
  - \( C_k' = \{\} \)
  - **for** all entries \( t \in C_{k-1}' \)
    - \( C_t = \{c \in C_k \mid t[c-c[k]]=1 \text{ and } t[c-c[k-1]]=1\} \)
    - **for** all \( c \in C_t \) \( \{c.\text{count}++\} \)
    - if \( (C_t \neq \{\}) \)
      - append \( C_t \) to \( C_k' \)
    - endif
  - endfor
  - \( L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\} \)
- endfor
- return \( U_k L_k \)

- One single pass over the data
- \( C_k' \) is generated from \( C_{k-1}' \)
- For small values of \( k \), \( C_k' \) could be larger than the database!
- For large values of \( k \), \( C_k' \) can be very small
Apriori vs. AprioriTID

• *Apriori* makes multiple passes over the data while *AprioriTID* makes a single pass over the data.

• *AprioriTID* needs to store additional data structures that may require more space than *Apriori*.

• Both algorithms need to check all candidates’ frequencies in every step.
Implementations

• Lots of them around

• See, for example, the web page of Bart Goethals: http://www.adrem.ua.ac.be/~goethals/software/

• Typical input format: each row lists the items (using item id's) that appear in every row
Lecture outline

• **Task 1:** Methods for finding all frequent itemsets efficiently

• **Task 2:** Methods for finding association rules efficiently
Definition: Association Rule

Let $D$ be database of transactions

- e.g.:

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>A, B, C</td>
</tr>
<tr>
<td>1000</td>
<td>A, C</td>
</tr>
<tr>
<td>4000</td>
<td>A, D</td>
</tr>
<tr>
<td>5000</td>
<td>B, E, F</td>
</tr>
</tbody>
</table>

- Let $I$ be the set of items that appear in the database, e.g., $I=\{A,B,C,D,E,F\}$

- A rule is defined by $X \rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \emptyset$
  - e.g.: $\{B,C\} \rightarrow \{A\}$ is a rule
Definition: Association Rule

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where $X$ and $Y$ are non-overlapping itemsets
  - Example:
    $$\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$$

- **Rule Evaluation Metrics**
  - **Support ($s$)**
    - Fraction of transactions that contain both $X$ and $Y$
  - **Confidence ($c$)**
    - Measures how often items in $Y$ appear in transactions that contain $X$

### Example:

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$
Rule Measures: Support and Confidence

Find all the rules $X \rightarrow Y$ with minimum confidence and support

- **Support**, $s$, probability that a transaction contains $\{X \cup Y\}$
- **Confidence**, $c$, *conditional probability* that a transaction having $X$ also contains $Y$

Let minimum support 50%, and minimum confidence 50%, we have

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>A, B, C</td>
</tr>
<tr>
<td>200</td>
<td>A, C</td>
</tr>
<tr>
<td>300</td>
<td>A, D</td>
</tr>
<tr>
<td>400</td>
<td>B, E, F</td>
</tr>
</tbody>
</table>
What is the support and confidence of the rule: \{B, D\} \rightarrow \{A\}

- **Support:**
  - percentage of tuples that contain \{A, B, D\} = 75%

- **Confidence:**
  \[
  \frac{\text{number of tuples that contain } \{A, B, D\}}{\text{number of tuples that contain } \{B, D\}} = 100\%
  \]
Association-rule mining task

• Given a set of transactions $D$, the goal of association rule mining is to find all rules having
  – support $\geq \text{minsup}$ threshold
  – confidence $\geq \text{minconf}$ threshold
Brute-force algorithm for association-rule mining

• List all possible association rules
• Compute the support and confidence for each rule
• Prune rules that fail the $\text{minsup}$ and $\text{minconf}$ thresholds

• $\Rightarrow$ Computationally prohibitive!
Computational Complexity

- Given \( d \) unique items in \( I \):
  - Total number of itemsets = \( 2^d \)
  - Total number of possible association rules:
    \[
    R = \sum_{k=1}^{d-1} \left( \binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right) = 3^d - 2^{d+1} + 1
    \]
Mining Association Rules

Example of Rules:

\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\} \ (s=0.4, \ c=0.67)
\{\text{Milk, Beer}\} \rightarrow \{\text{Diaper}\} \ (s=0.4, \ c=1.0)
\{\text{Diaper, Beer}\} \rightarrow \{\text{Milk}\} \ (s=0.4, \ c=0.67)
\{\text{Beer}\} \rightarrow \{\text{Milk, Diaper}\} \ (s=0.4, \ c=0.67)
\{\text{Diaper}\} \rightarrow \{\text{Milk, Beer}\} \ (s=0.4, \ c=0.5)
\{\text{Milk}\} \rightarrow \{\text{Diaper, Beer}\} \ (s=0.4, \ c=0.5)

Observations:

• All the above rules are binary partitions of the same itemset: \{\text{Milk, Diaper, Beer}\}
• Rules originating from the same itemset have identical support but can have different confidence
• Thus, we may decouple the support and confidence requirements
Mining Association Rules

• Two-step approach:
  – **Frequent Itemset Generation**
    – Generate all itemsets whose support $\geq$ minsup
  – **Rule Generation**
    – Generate high confidence rules from each frequent itemset, where each rule is a binary partition of a frequent itemset
Rule Generation – Naive algorithm

• Given a frequent itemset \( X \), find all non-empty subsets \( y \subset X \) such that \( y \rightarrow X - y \) satisfies the minimum confidence requirement

  – If \( \{A,B,C,D\} \) is a frequent itemset, candidate rules:
    
    \[
    \begin{align*}
    ABC \rightarrow D, & \quad ABD \rightarrow C, & \quad ACD \rightarrow B, & \quad BCD \rightarrow A, \\
    A \rightarrow BCD, & \quad B \rightarrow ACD, & \quad C \rightarrow ABD, & \quad D \rightarrow ABC \\
    AB \rightarrow CD, & \quad AC \rightarrow BD, & \quad AD \rightarrow BC, & \quad BC \rightarrow AD, \\
    BD \rightarrow AC, & \quad CD \rightarrow AB, & & \\
    \end{align*}
    \]

• If \( |X| = k \), then there are \( 2^k - 2 \) candidate association rules (ignoring \( L \rightarrow \emptyset \) and \( \emptyset \rightarrow L \))
Efficient rule generation

• How to efficiently generate rules from frequent itemsets?
  – In general, confidence does not have an anti-monotone property
    \[ c(ABC \rightarrow D) \] can be larger or smaller than \[ c(AB \rightarrow D) \]
  – But confidence of rules generated from the same itemset has an anti-monotone property
  – Example: \( X = \{A,B,C,D\} \):
    \[ c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD) \]
  – Why?

Confidence is anti-monotone w.r.t. number of items on the RHS of the rule
Rule Generation for Apriori Algorithm

Lattice of rules

Low Confidence Rule

Pruned Rules
Apriori algorithm for rule generation

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent.

- \textit{join}(CD \rightarrow AB, BD \rightarrow AC) would produce the candidate rule \( D \rightarrow ABC \).

- \textit{Prune} rule \( D \rightarrow ABC \) if there exists a subset (e.g., \( AD \rightarrow BC \)) that does not have high confidence.