Association Rules

1. Objectives .......................................................................................................................... 2
2. Definitions ............................................................................................................................ 2
3. Type of Association Rules .................................................................................................... 7
4. Frequent Itemset generation .................................................................................................. 9
5. Apriori Algorithm: Mining Single-Dimension Boolean AR .............................................. 13
   5.1. Join Step: ....................................................................................................................... 15
   5.2. Prune step ..................................................................................................................... 17
   5.3. Example ......................................................................................................................... 18
   5.4. Pseudo-code ................................................................................................................. 19
   5.5. Challenges .................................................................................................................... 19
   5.6. Improving the Efficiency of Apriori ........................................................................... 20
6. Mining Frequent Itemsets without Candidate Generation .............................................. 22
   6.1. Mining Frequent patterns using FP-Tree ..................................................................... 25
   6.2. Major steps to mine FP-trees ....................................................................................... 25
7. Multiple-Level Association Rules ...................................................................................... 31
   7.1. Approach ...................................................................................................................... 31
   7.2. Redundancy Filtering.................................................................................................... 36
1. Objectives

- Increase sales and reduce costs
- What products were often purchased together?
  - Beer and diapers?!
- What are the subsequent purchases after buying a PC?
- What kinds of DNA are sensitive to this new drug?
- Can we automatically classify web documents?
- Broad applications:
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis
  - Web log (click stream) analysis, DNA sequence analysis, etc.
- Example: Items frequently purchased together:
  - **Bread ➔ PeanutButter**
- Why associations:
  - Placement
  - Advertising
  - Sales
  - Coupons

2. Definitions

- Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.

- Frequent pattern: pattern (set of items, sequence, etc.) that occurs frequently in a database.

- Basic Concepts:
- A set of items: \( I = \{x_1, \ldots, x_k\} \)

- Transactions: \( D = \{t_1, t_2, \ldots, t_n\}, t_j \subseteq I \)

- A k-Itemset: \( \{I_{i1}, I_{i2}, \ldots, I_{ik}\} \subseteq I \)

- Support of an itemset: Percentage of transactions that contain that itemset.

- Large (Frequent) itemset: Itemset whose number of occurrences is above a threshold.

- Example:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>Bread, Jelly, PeanutButter</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>Bread, PeanutButter</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>Bread, Milk, PeanutButter</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>Beer, Bread</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>Beer, Milk</td>
</tr>
</tbody>
</table>

\( I = \{ \text{Beer}, \text{Bread}, \text{Jelly}, \text{Milk}, \text{PeanutButter}\} \)
Support of \( \{\text{Bread}, \text{PeanutButter}\} = \frac{3}{5} = 60\% \)
• Association Rules
  o Implication: $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$;
  o Support of AR (s) $X \Rightarrow Y$:
    ▪ Percentage of transactions that contain $X \cup Y$
    ▪ Probability that a transaction contains $X \cup Y$.
  o Confidence of AR (a) $X \Rightarrow Y$: 

<table>
<thead>
<tr>
<th>Transaction-id</th>
<th>Items bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A, B, C</td>
</tr>
<tr>
<td>20</td>
<td>A, C</td>
</tr>
<tr>
<td>30</td>
<td>A, D</td>
</tr>
<tr>
<td>40</td>
<td>B, E, F</td>
</tr>
</tbody>
</table>
- Ratio of number of transactions that contain $X \cup Y$ to the number that contain $X$
- Conditional probability that a transaction having $X$ also contains $Y$.

- Example:

<table>
<thead>
<tr>
<th>Transaction-id</th>
<th>Items bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A, B, C</td>
</tr>
<tr>
<td>20</td>
<td>A, C</td>
</tr>
<tr>
<td>30</td>
<td>A, D</td>
</tr>
<tr>
<td>40</td>
<td>B, E, F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequent pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>75%</td>
</tr>
<tr>
<td>{B}</td>
<td>50%</td>
</tr>
<tr>
<td>{C}</td>
<td>50%</td>
</tr>
<tr>
<td>{A, C}</td>
<td>50%</td>
</tr>
</tbody>
</table>

- For rule $A \Rightarrow C$:

$$\text{Support}(A \Rightarrow C) = P(A \cup C) = \text{support}(\{A\} \cup \{C\}) = 50\%$$

$$\text{confidence}(A \Rightarrow C) = P(C|A)$$
= support({A}∪{C})/support({A}) = 66.6%

- Another Example:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>Bread, Jelly, PeanutButter</td>
</tr>
<tr>
<td>$t_2$</td>
<td>Bread, PeanutButter</td>
</tr>
<tr>
<td>$t_3$</td>
<td>Bread, Milk, PeanutButter</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Beer, Bread</td>
</tr>
<tr>
<td>$t_5$</td>
<td>Beer, Milk</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$X \Rightarrow Y$</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bread $\Rightarrow$ Peanutbutter</td>
<td>$3/5% = 60%$</td>
<td>$(3/5)/(4/5)% = 75%$</td>
</tr>
<tr>
<td>Peanutbutter $\Rightarrow$ Bread</td>
<td>$60%$</td>
<td>$(3/5)/(3/5)% = 100%$</td>
</tr>
<tr>
<td>Jelly $\Rightarrow$ Milk</td>
<td>$0%$</td>
<td>$0%$</td>
</tr>
<tr>
<td>Jelly $\Rightarrow$ Peanutbutter</td>
<td>$1/5 % = 20%$</td>
<td>$(1/5)/(1/5) % = 100%$</td>
</tr>
</tbody>
</table>
• Association Rule Problem:
  o Given a set of items \( I = \{I_1, I_2, \ldots, I_m\} \) and a database of transactions \( D = \{t_1, t_2, \ldots, t_n\} \) where \( t_i = \{I_{i1}, I_{i2}, \ldots, I_{ik}\} \) and \( I_{ij} \in I \), the Association Rule Problem is to identify all association rules \( X \rightarrow Y \) with a minimum support and confidence.
  
  o NOTE: Support of \( X \rightarrow Y \) is same as support of \( X \cup Y \).

• Association Rules techniques:
  - Find all frequent itemsets.
  - Generate strong association rules from the frequent itemsets: those rules must satisfy minimum support and minimum confidence.

3. Type of Association Rules

• Boolean AR:
  o It is a rule that checks whether an item is present or absent.
  o All the examples we have seen so far are Boolean AR.

• Quantitative AR:
  o It describes associations between quantitative items or attributes.
  o Generally, quantitative values are partitioned into intervals.
  o Example:

    \[ \text{Age}(X, "30..39") \land \text{income}(X, "80K..100K") \]
buys(X, High Resolution TV)

- Single-Dimension AR:
  - It is a rule that references only one dimension.
  - Example:
    
    ```plaintext
    buys(X,"computer")
    ➔ buys(X,"financial_software")
    ```
    
    The single dimension is “buys”

  - The following rule is a multi-dimensional AR:
    
    ```plaintext
    Age(X,"30..39") ∧ income(X,"80K..100K")
    ➔ buys(X, High Resolution TV)
    ```

- Multi-level AR
  - It is a set of rules that reference different levels of abstraction.
  - Example:
    
    ```plaintext
    Age(X,"30..39") ➔ buys(X, “desktop”)
    Age(X,"20..29") ➔ buys(X, “laptop”)
    ```
    
    Laptop ➔ desktop ➔ computer
4. Frequent Itemset generation

- Given d items, there are \(2^d\) possible candidate itemsets
• Brute-force approach:
  o Each itemset in the lattice is a candidate frequent itemset
  o Count the support of each candidate by scanning the database

  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>

  List of Candidates

  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => Expensive since M = 2^d !!!

• Complexity:
  o Given d unique items:
  o Total number of itemsets = 2^d
  o Total number of possible association rules:

  \[ R = \sum_{k=1}^{d-l} \binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \]
  \[ = 3^d - 2^{d+1} + 1 \]
  o If d=6, R = 602 rules
Frequent Itemset Generation Strategies

- 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

- 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
5. Apriori Algorithm: Mining Single-Dimension Boolean AR

- It is used to mine Boolean, single-level, and single-dimension ARs.
- Apriori Principle
• Apriori algorithm:
  
  o Uses prior knowledge of frequent itemset properties.

  o It is an iterative algorithm known as level-wise search.

  o The search proceeds level-by-level as follows:
    ‧ First determine the set of frequent 1-itemset; L1
    ‧ Second determine the set of frequent 2-itemset using L1: L2
    ‧ Etc.

  o The complexity of computing Li is $O(n)$ where n is the number of transactions in the transaction database.

  o Reduction of search space:
    ‧ In the worst case what is the number of itemsets in a level Li?
    ‧ Apriori uses “Apriori Property”:

  o Apriori Property:
    ‧ It is an anti-monotone property: if a set cannot pass a test, all of its supersets will fail the same test as well.
    ‧ It is called anti-monotone because the property is monotonic in the context of failing a test.
    ‧ All nonempty subsets of a frequent itemset must also be frequent.
    ‧ An itemset I is not frequent if it does not satisfy the minimum support threshold:
P(I) < min_sup

- If an item A is added to the itemset I, then the resulting itemset I ∪ A cannot occur more frequently than I:
  I ∪ A is not frequent

Therefore, P(I ∪ A) < min_sup

• How Apriori algorithm uses “Apriori property”?
  o In the computation of the itemsets in L_k using L_{k-1}
  o It is done in two steps:
    ▪ Join
    ▪ Prune

5.1. Join Step:

• The set of candidate k-itemsets (element of L_k), C_k, is generated by joining L_{k-1} with itself:

  \[ L_{k-1} \bowtie L_{k-1} \]

• Given l_1 and l_2 of L_{k-1}

  \[ L_i = l_{i1}, l_{i2}, l_{i3}, \ldots, l_{i(k-2)}, l_{i(k-1)} \]
  \[ L_j = l_{j1}, l_{j2}, l_{j3}, \ldots, l_{j(k-2)}, l_{j(k-1)} \]

  Where L_i and L_j are sorted.

• L_i and L_j are joined if there are different (no duplicate generation). Assume the following:

  \[ l_{i1} = l_{j1}, \ l_{i2} = l_{j1}, \ldots, \ l_{i(k-2)} = l_{j(k-2)} \text{ and } l_{i(k-1)} < l_{j(k-1)} \]

• The resulting itemset is:
$l_1, l_2, l_3, \ldots, l_{i(k-1)}, l_{j(k-1)}$

- Example of Candidate-generation:

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

Self-joining: $L_3 \Join L_3$

- $abcd$ from $abc$ and $abd$
- $acde$ from $acd$ and $ace$
5.2. Prune step

- \( C_k \) is a superset of \( L_k \) ➔ some itemset in \( C_k \) may or may not be frequent.
- \( L_k \): Test each generated itemset against the database:
  - Scan the database to determine the count of each generated itemset and include those that have a count no less than the minimum support count.
  - This may require intensive computation.

- Use Apriori property to reduce the search space:
  - Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset.
  - Remove from \( C_k \) any k-itemset that has a (k-1)-subset not in \( L_{k-1} \) (itemsets that are not frequent)
  - Efficiently implemented: maintain a hash table of all frequent itemset.

- Example of Candidate-generation and Pruning:

  \[ L_3 = \{abc, abd, acd, ace, bcd\} \]

  **Self-joining:** \( L_3 \bowtie L_3 \)

  - *abcd* from abc and abd
  - *acde* from acd and ace

  **Pruning:**
  - acde is removed because ade is not in \( L_3 \)
  - \( C_4 = \{abcd\} \)
5.3. Example

Database TDB

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A, C, D</td>
</tr>
<tr>
<td>20</td>
<td>B, C, E</td>
</tr>
<tr>
<td>30</td>
<td>A, B, C, E</td>
</tr>
<tr>
<td>40</td>
<td>B, E</td>
</tr>
</tbody>
</table>

**1st scan**

$C_1$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>2</td>
</tr>
<tr>
<td>{B}</td>
<td>3</td>
</tr>
<tr>
<td>{C}</td>
<td>3</td>
</tr>
<tr>
<td>{D}</td>
<td>1</td>
</tr>
<tr>
<td>{E}</td>
<td>3</td>
</tr>
</tbody>
</table>

$L_1$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>2</td>
</tr>
<tr>
<td>{B}</td>
<td>3</td>
</tr>
<tr>
<td>{C}</td>
<td>3</td>
</tr>
<tr>
<td>{E}</td>
<td>3</td>
</tr>
</tbody>
</table>

**2nd scan**

$C_2$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B}</td>
<td>1</td>
</tr>
<tr>
<td>{A, C}</td>
<td>2</td>
</tr>
<tr>
<td>{A, E}</td>
<td>1</td>
</tr>
<tr>
<td>{B, C}</td>
<td>2</td>
</tr>
<tr>
<td>{B, E}</td>
<td>3</td>
</tr>
<tr>
<td>{C, E}</td>
<td>2</td>
</tr>
</tbody>
</table>

$L_2$

<table>
<thead>
<tr>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B}</td>
</tr>
<tr>
<td>{A, C}</td>
</tr>
<tr>
<td>{A, E}</td>
</tr>
<tr>
<td>{B, C}</td>
</tr>
<tr>
<td>{B, E}</td>
</tr>
<tr>
<td>{C, E}</td>
</tr>
</tbody>
</table>

**3rd scan**

$C_3$

<table>
<thead>
<tr>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B, C, E}</td>
</tr>
</tbody>
</table>

$L_3$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B, C, E}</td>
<td>2</td>
</tr>
</tbody>
</table>
5.4. Pseudo-code

$C_k$: Candidate itemset of size $k$
$L_k$: frequent itemset of size $k$
$L_1 = \{\text{frequent items}\}$;

for (k = 1; $L_k \neq \emptyset$; k++) do
    - $C_{k+1} =$ candidates generated from $L_k$;
    - for each transaction $t$ in database do
        increment the count of all candidates in $C_{k+1}$
        that are contained in $t$;
    endfor;
    - $L_{k+1} =$ candidates in $C_{k+1}$ with min_support
endfor;
return $\cup_k L_k$;

5.5. Challenges

- Multiple scans of transaction database
- Huge number of candidates
- Tedious workload of support counting for candidates
- Improving Apriori:
  - general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates
  - Easily parallelized
5.6. Improving the Efficiency of Apriori

- Several attempts have been introduced to improve the efficiency of Apriori:
  - Hash-based technique
    - Hashing itemset counts
    - Example:

  - Transaction DB:

    | TID | List of Transactions |
    |-----|----------------------|
    | T100 | I1, I2, I5           |
    | T200 | I2, I4               |
    | T300 | I2, I3               |
    | T400 | I1, I2, I4           |
    | T500 | I1, I3               |
    | T600 | I2, I3               |
    | T700 | I1, I3               |
    | T800 | I1, I2, I3, I5       |
    | T900 | I1, I2, I3           |

  - Create a hash table for candidate 2-itemsets:
    - Generate all 2-itemsets for each transaction in the transaction DB
    - \( H(x,y) = ((\text{order of } x) \times 10 + (\text{order of } y)) \mod 7 \)
- A 2-itemset whose corresponding bucket count is below the support threshold cannot be frequent.

<table>
<thead>
<tr>
<th>Bucket @</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucket count</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Content</td>
<td>{I1,I4}</td>
<td>{I1,I5}</td>
<td>{I2,I3}</td>
<td>{I2,I4}</td>
<td>{I2,I5}</td>
<td>{I1,I2}</td>
<td>{I1,I3}</td>
</tr>
<tr>
<td></td>
<td>{I3,I5}</td>
<td>{I1,I5}</td>
<td>{I2,I3}</td>
<td>{I2,I4}</td>
<td>{I2,I5}</td>
<td>{I1,I2}</td>
<td>{I1,I3}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{I2,I3}</td>
<td></td>
<td>{I1,I2}</td>
<td>{I1,I3}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{I2,I3}</td>
<td>{I1,I2}</td>
<td>{I1,I3}</td>
<td></td>
</tr>
</tbody>
</table>

- Remember: support(x \rightarrow y) = percentage number of transactions that contain x and y. Therefore, if the minimum support is 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in C_2.

- Transaction reduction
  - Reduce the number of transactions scanned in future iterations.
  - A transaction that does not contain any frequent k-itemsets cannot contain any frequent (k+1)-itemsets: Do not include such transaction in subsequent scans.

- Other techniques include:
  - Partitioning (partition the data to find candidate itemsets)
  - Sampling (Mining on a subset of the given data)
  - Dynamic itemset counting (Adding candidate itemsets at different points during a scan)
6. Mining Frequent Itemsets without Candidate Generation

- Objectives:
  - The bottleneck of *Apriori*: candidate generation
  - Huge candidate sets:
    - For $10^4$ frequent 1-itemset, Apriori will generate $10^7$ candidate 2-itemsets.
    - To discover a frequent pattern of size 100, e.g., \{a1, a2, \ldots, a100\}, one needs to generate $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs $(n+1)$ scans, $n$ is the length of the longest pattern.

- Principal
  - Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
    - Highly condensed, but complete for frequent pattern mining
    - Avoid costly database scans
  - Develop an efficient, FP-tree-based frequent pattern mining method
    - A divide-and-conquer methodology: decompose mining tasks into smaller ones
    - Avoid candidate generation: sub-database test only!
• Algorithm:
  1. Scan DB once, find frequent 1-itemset (single item pattern)
  2. Order frequent items in frequency descending order, called \textit{L order}: (in the example below: F(4), c(4), a(3), etc.)
  3. Scan DB again and construct FP-tree
     a. Create the root of the tree and label it null or \{\}
     b. The items in each transaction are processed in the L order (sorted according to descending support count).
     c. Create a branch for each transaction
     d. Branches share common prefixes
• Example: \( \text{min}_\text{support} = 0.5 \)

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
<th>(Ordered) frequent items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{f, a, c, d, g, i, m, p}</td>
<td>{f, c, a, m, p}</td>
</tr>
<tr>
<td>200</td>
<td>{a, b, c, f, l, m, o}</td>
<td>{f, c, a, b, m}</td>
</tr>
<tr>
<td>300</td>
<td>{b, f, h, j, o}</td>
<td>{f, b}</td>
</tr>
<tr>
<td>400</td>
<td>{b, c, k, s, p}</td>
<td>{c, b, p}</td>
</tr>
<tr>
<td>500</td>
<td>{a, f, c, e, l, p, m, n}</td>
<td>{f, c, a, m, p}</td>
</tr>
</tbody>
</table>

**Node Structure:**

<table>
<thead>
<tr>
<th>Item</th>
<th>count</th>
<th>node pointer</th>
<th>child pointers</th>
<th>parent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.1. Mining Frequent patterns using FP-Tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree
- Method
  - For each item, construct its conditional pattern-base, and then its conditional FP-tree
  - Recursion: Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

6.2. Major steps to mine FP-trees

- Main Steps:
  1. Construct conditional pattern base for each node in the FP-tree
  2. Construct conditional FP-tree from each conditional pattern-base
  3. Recursively mine conditional FP-trees and grow frequent patterns obtained so far If the conditional FP-tree contains a single path, simply enumerate all the patterns

- Step 1: From FP-tree to Conditional Pattern Base

  - Starting at the frequent header table in the FP-tree
  - Traverse the FP-tree by following the link of each frequent item, starting by the item with the highest frequency.
  - Accumulate all of transformed prefix paths of that item to form a conditional pattern base
Example:

### Header Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Supp. Count</th>
<th>Node Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

### Conditional pattern bases

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional pattern base</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>f:3</td>
</tr>
<tr>
<td>a</td>
<td>fc:3</td>
</tr>
<tr>
<td>b</td>
<td>fca:1, f:1, c:1</td>
</tr>
<tr>
<td>m</td>
<td>fca:2, fcab:1</td>
</tr>
<tr>
<td>p</td>
<td>fcam:2, cb:1</td>
</tr>
</tbody>
</table>

- Properties of FP-tree for Conditional Pattern Base Construction:
  - Node-link property
    - For any frequent item $a_i$, all the possible frequent patterns that contain $a_i$ can be obtained by following $a_i$'s node-links, starting from $a_i$'s head in the FP-tree header.
  - Prefix path property
To calculate the frequent patterns for a node $a_i$ in a path $P$, only the prefix sub-path of $a_i$ in $P$ need to be accumulated and its frequency count should carry the same count as node $a_i$.

- Step 2: Construct Conditional FP-tree
  - For each pattern-base
    - Accumulate the count for each item in the base
    - Construct the FP-tree for the frequent items of the pattern base
  - Example:
    \[ m\text{-conditional pattern base: } fca:2, fcab:1 \]

```
<table>
<thead>
<tr>
<th>Item</th>
<th>Supp. Count</th>
<th>Node Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
```

All frequent patterns concerning $m$: $m, fm, cm, am, fcm, fam, cam, fcam$
- Mining Frequent Patterns by Creating Conditional Pattern-Bases:

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional pattern-base</th>
<th>Conditional FP-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{(fcam:2), (cb:1)}</td>
<td>{(c:3)}</td>
</tr>
<tr>
<td>m</td>
<td>{(fca:2), (fcab:1)}</td>
<td>{(f:3, c:3, a:3)}</td>
</tr>
<tr>
<td>b</td>
<td>{(fca:1), (f:1), (c:1)}</td>
<td>Empty</td>
</tr>
<tr>
<td>a</td>
<td>{(fc:3)}</td>
<td>{(f:3, c:3)}</td>
</tr>
<tr>
<td>c</td>
<td>{(f:3)}</td>
<td>{(f:3)}</td>
</tr>
<tr>
<td>f</td>
<td>Empty</td>
<td>Empty</td>
</tr>
</tbody>
</table>

- Step 3: Recursively mine the conditional FP-tree

```
| {}      | Cond. pattern base of “am”: (fc:3) | f:3 |
| {}      |                                  |     |
| f:3     |                                  | c:3 |
| c:3     |                                  |     |
| a:3     | Cond. pattern base of “cm”: (fa:3) | {} |

am-conditional FP-tree

m-conditional FP-tree

| {}      | Cond. pattern base of “cam”: (f:3) | f:3 |
| {}      |                                  |     |
```

cm-conditional FP-tree

cam-conditional FP-tree
• Why is FP-Tree mining fast?
  o The performance study shows FP-growth is an order of magnitude faster than Apriori
  o Reasoning:
    ▪ No candidate generation, no candidate test
    ▪ Use compact data structure
    ▪ Eliminate repeated database scan
    ▪ Basic operation is counting and FP-tree building

• FP-Growth vs. Apriori: Scalability with the support Threshold [Jiawei Han and Micheline Kamber]
7. Multiple-Level Association Rules

- Items often form hierarchy.
- Items at the lower level are expected to have lower support.
- Rules regarding itemsets at appropriate levels could be quite useful.
- Transaction database can be encoded based on dimensions and levels
- We can explore shared multi-level mining

7.1. Approach

- A top-down, progressive deepening approach:
  - First find high-level strong rules:
    
    milk $\Rightarrow$ bread [20%, 60%].
• Then find their lower-level “weaker” rules:

\[
2\% \text{ milk } \rightarrow \text{ wheat bread } [6\%, 50\%].
\]

• Variations at mining multiple-level association rules.
  • Level-crossed association rules:

\[
2\% \text{ milk } \rightarrow \text{ Wonder wheat bread}
\]

• Association rules with multiple, alternative hierarchies:

\[
2\% \text{ milk } \rightarrow \text{ Wonder bread}
\]

• Two multiple-level mining associations strategies:
  • Uniform Support
  • Reduced support

• Uniform Support: the same minimum support for all levels

  • One minimum support threshold.
  • No need to examine itemsets containing any item whose ancestors do not have minimum support.
  • Drawback:
    o Lower level items do not occur as frequently. If support threshold too high \(\rightarrow\) miss low level associations
    too low \(\rightarrow\) generate too many high level assoc.
Reduced Support: reduced minimum support at lower levels

- There are 4 search strategies:
  - Level-by-level independent
  - Level-cross filtering by k-itemset
  - Level-cross filtering by single item
  - Controlled level-cross filtering by single item

- **Level-by-Level independent:**
  - Full-breadth search
  - No background knowledge is used.
  - Each node is examined regardless the frequency of its parent.

- **Level-cross filtering by single item:**
  - An item at the ith level is examined if and only if its parent node at the (i-1)th level is frequent.

- **Level-cross filtering by k-itemset:**
  - A k-itemset at the ith level is examined if and only if its corresponding parent k-itemset at the (i-1)th level is frequent.
This restriction is stronger than the one in level-cross filtering by single item.

They are not usually many k-itemsets that, when combined, are also frequent:

Many valuable patterns can be mined

- **Controlled level-cross filtering by single item:**
  - A variation of the level-cross filtering by single item: Relax the constraint in this approach.
  - Allow the children of items that do not satisfy the minimum support threshold to be examined if these items satisfy the level passage threshold:

  \[ level\_passage\_supp \]

  - **level\_passage\_sup** Value: It is typically set between the min\_sup value of the given level and the min\_sup of the next level.
Example:

Level 1
min_sup = 12%
level_passage_sup = 8%

Milk
[support = 10%]

Level 2
min_sup = 4%

2% Milk
[support = 6%]

Skim Milk
[support = 5%]
7.2. Redundancy Filtering

- Some rules may be redundant due to “ancestor” relationships between items.

- Definition: A rule $R_1$ is an ancestor of a rule, $R_2$, if $R_1$ can be obtained by replacing the items in $R_2$ by their ancestors in a concept hierarchy.

- Example

  $R_1$: milk $\rightarrow$ wheat bread [support = 8%, confidence = 70%]
  $R_2$: 2% milk $\rightarrow$ wheat bread [support = 2%, confidence = 72%]

  Milk in $R_1$ is an ancestor of 2% milk in $R_2$.

- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support is close to the “expected” value, based on the rule’s ancestor:
  - $R_2$ is redundant since its confidence is close to the confidence of $R_1$ (kind of expected) and its support is around 2% = (8% * ¼)
  - $R_2$ does not add any additional information.