Chapter 2: Mining Association Rules

Dr. Osmar R. Zaïane

University of Alberta

Chapter 6 Objectives

Understand association analysis in large datasets and get a brief introduction to the different types of association rule mining.

What Is Association Rule Mining?

- Association rule mining searches for relationships between items in a dataset:
  - aims at discovering associations between items in a transactional database.

- Rule form: "Body → Head [support, confidence]"

- "buys(x, "bread") → buys(x, "milk") [0.6%, 65%]
- "major(x, "CS") ∧ takes(x, "DB") → grade(x, "A") [1%, 75%]"
Transactional Databases

Transaction Frequent itemset Rule

\{bread, milk, Pop, \ldots\} \rightarrow (Bread, milk) \rightarrow Bread \rightarrow milk

\{term_1, term_2, \ldots, term_n\} \rightarrow (term_2, term_{25}) \rightarrow term_2 \rightarrow term_{25}

\{f_1, f_2, \ldots, f_a\} \rightarrow (f_3, f_5, f_a) \rightarrow f_3^a \rightarrow f_a

Lecture Outline

Part I: Concepts (30 minutes)
- Basic concepts
  - Support and Confidence
- Naïve approach

Part II: The Apriori Algorithm (30 minutes)
- Principles
- Algorithm
- Running Example

Part III: The FP-Growth Algorithm (30 minutes)
- FP-tree structure
- Running Example

Part IV: More Advanced Concepts (30 minutes)
- Database layout and space search approach
- Other types of patterns and constraints

Finding Rules in Transaction Data Set

- 6 transactions
- 5 items: \{Pop, Bread, Jelly, Milk, PeanutButter\}

<table>
<thead>
<tr>
<th>Transactions</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Bread, Jelly, PeanutButter</td>
</tr>
<tr>
<td>T2</td>
<td>Bread, PeanutButter</td>
</tr>
<tr>
<td>T3</td>
<td>Bread, Milk, PeanutButter</td>
</tr>
<tr>
<td>T4</td>
<td>Pop, Bread</td>
</tr>
<tr>
<td>T5</td>
<td>Pop, Milk</td>
</tr>
<tr>
<td>T6</td>
<td>Bread, Milk</td>
</tr>
</tbody>
</table>

- Searching for rules of the form X \rightarrow Y, where X and Y are sets of items
  - e.g. Bread \rightarrow Jelly; Bread, Jelly \rightarrow PeanutButter

- Design an efficient algorithm for mining association rules in large data sets
- Develop an effective approach for distinguishing interesting rules from irrelevant ones
Basic Concepts

A transaction is a set of items: \( T=\{i_1, i_2, \ldots, i_t\} \)

\( T \subseteq I \), where \( I \) is the set of all possible items \( \{i_1, i_2, \ldots, i_d\} \)

\( D \), the task relevant data, is a set of transactions \( D=\{T_1, T_2, \ldots, T_n\} \).

An association rule is of the form: \( P \Rightarrow Q \), where \( P \subseteq I, Q \subseteq I \), and \( P \cap Q = \emptyset \).

Support of an Itemset

- Support of \( P = P_1 \land P_2 \land \ldots \land P_k \) in \( D \): \( \sigma(P/D) \) is the probability that \( P \) occurs in \( D \): it is the percentage of transactions \( T \) in \( D \) satisfying \( P \).

- I.e. the support of an item (or itemset) \( X \) is the percentage of transactions in which that item (or items) occurs: \( \text{number of } T \text{ by cardinality of } D \).

\[
\text{support}(X) = \frac{\#X}{n}
\]

Support and Confidence of an Association Rule

- The support of an association rule \( X \Rightarrow Y \) is the percentage of transactions that contain \( X \cup Y \)

\[
\text{support}(X \Rightarrow Y) = \frac{\#(X \cup Y)}{n}
\]

- The confidence of an association rule \( X \Rightarrow Y \) is the ratio of the number of transactions that contain \( X \cup Y \) to the number of transactions that contain \( X \)

\[
\text{confidence}(X \Rightarrow Y) = \frac{\#(X \cup Y)}{\#X}
\]

- Confidence of a rule \( P \Rightarrow Q \) in database \( D \): \( \varphi(P \Rightarrow Q/D) \) is the ratio \( \sigma((P \land Q)/D) \) by \( \sigma(P/D) \)

\[
\text{confidence}(X \Rightarrow Y) = \frac{\text{support}(X \Rightarrow Y)}{\text{support}(X)}
\]
Support and Confidence – cont.

- What is the support and confidence of the following rules?
  - Pop $\Rightarrow$ Bread
  - {Bread, PeanutButter} $\Rightarrow$ Jelly

- Support and confidence for some association rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bread $\Rightarrow$ PeanutButter</td>
<td>50%</td>
<td>60%</td>
</tr>
<tr>
<td>PeanutButter $\Rightarrow$ Bread</td>
<td>50%</td>
<td>100%</td>
</tr>
<tr>
<td>Pop $\Rightarrow$ Bread</td>
<td>16%</td>
<td>50%</td>
</tr>
<tr>
<td>PeanutButter $\Rightarrow$ Jelly</td>
<td>16%</td>
<td>33%</td>
</tr>
<tr>
<td>Jelly $\Rightarrow$ PeanutButter</td>
<td>16%</td>
<td>100%</td>
</tr>
<tr>
<td>Jelly $\Rightarrow$ Milk</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>(Bread, PeanutButter) $\Rightarrow$ Jelly</td>
<td>16%</td>
<td>33%</td>
</tr>
</tbody>
</table>

Why the difference?

- Support measures how often the rule occurs in the database.
- Confidence measures the strength of the rule.

Frequent Itemsets and Strong Rules

Support and Confidence are bound by Thresholds:

- minimum support $\sigma'$
- minimum confidence $\varphi'$

A **Frequent (or large) itemset** $I$ in $D$ is an itemset with a support larger than the minimum support;

A **strong rule** $X \Rightarrow Y$ is a rule that is frequent (i.e. support higher than minimum support) and its confidence is higher than the minimum confidence threshold.

Association Rule Problem Definition

- Given $I = \{i_1, i_2, \ldots, i_m\}$, $D = \{t_1, t_2, \ldots, t_n\}$ and the support and confidence thresholds, the **association rule mining problem** is to identify all strong association rules $X \Rightarrow Y$.

Naïve Approach to Generate Association Rules

- Enumerate all possible rules and select those of them that satisfy the minimum support and confidence thresholds
- Not practical for large databases
  - For a given dataset with $m$ items, the total number of possible rules is $3^m - 2^{m+1} + 1$
  - For our example: $3^5 - 2^6 + 1 = 180$
  - More than 80% of these rules are discarded if $\sigma' = 0.2$ and $\varphi' = 0.5$
- We need a strategy for rule generation - generate only the promising rules

Better Approach

- Find the **frequent itemsets**: the sets of items that have minimum support
- Use the frequent itemsets to generate association rules. Keep only strong rules.
Generating Association Rules from Frequent Itemsets

- Only strong association rules are generated.
- Frequent itemsets satisfy minimum support threshold.
- Strong AR satisfy minimum confidence threshold.

Confidence($A \rightarrow B$) = \( \frac{\text{Support}(A \cup B)}{\text{Support}(A)} \)

For each frequent itemset, \( f \), generate all non-empty subsets of \( f \).
For every non-empty subset \( s \) of \( f \) do
- output rule \( s \rightarrow (f-s) \) if \( \text{support}(f)/\text{support}(s) \geq \text{min\_confidence} \)
end

Naïve Frequent Itemset Generation

- Brute-force approach (Basic approach):
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

<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, Water, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Water, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Water</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Transactions
List of Candidates

N

M

w

Matching each transaction against every candidate
Complexity \( \sim O(NMw) \) => Expensive since \( M = 2^d \) !!!

Lecture Outline

Part I: Concepts (30 minutes)
- Basic concepts
  - Support and Confidence
- Naïve approach

Part II: The Apriori Algorithm (30 minutes)
- Principles
- Algorithm
- Running Example

Part III: The FP-Growth Algorithm (30 minutes)
- FP-tree structure
- Running Example

Part IV: More Advanced Concepts (30 minutes)
- Database layout and space search approach
- Other types of patterns and constraints

An Influential Mining Methodology — The Apriori Algorithm

- The Apriori method:
  - Proposed by Agrawal & Srikant 1994
  - A similar level-wise algorithm by Mannila et al. 1994

- Major idea (Apriori Principle):
  - A subset of a frequent itemset must be frequent
    - E.g., if \{Pop, diaper, nuts\} is frequent, \{Pop, diaper\} must be.
      - Any itemset that is infrequent, its superset cannot be frequent!
  - A powerful, scalable candidate set pruning technique:
    - It reduces candidate k-itemsets dramatically (for \( k > 2 \))
**Apriori Algorithm**

- **Apriori principle:**
  - *A subset of any frequent (large) itemset is also frequent*
  - This also implies that *if an itemset is not frequent (small), a superset of it is also not frequent*
  
  - If we know that an itemset is infrequent, we need not generate any subsets of it as they will be infrequent.
  
  - Lines represent “subset” relationship
  - If ACD is frequent, than AC, AD, CD, A, C, D are also frequent, i.e. if an itemset is frequent than any set in a path above it is also frequent.
  - If AB is infrequent, than ABC, ABD, ABCD will also be infrequent, i.e. if an itemset is infrequent than any set in the path below is also infrequent.
  - If any of A, C, D, AC, AD, CD, is infrequent than ACD is infrequent (no need to check).

**Mining Association rules: the Key Steps**

1. **Find the frequent itemsets:** the sets of items that have minimum support
   - A subset of a frequent itemset must also be a frequent itemset, i.e., if \{AB\} is a frequent itemset, both \{A\} and \{B\} should be frequent itemsets.
   - Iteratively find frequent itemsets with cardinality from 1 to \(k\) (\(k\)-itemsets).

2. **Use the frequent itemsets to generate strong association rules.**

**The Apriori Algorithm**

\(C_k\): Candidate itemset of size \(k\)
\(L_k\): frequent itemset of size \(k\)

\[L_1 = \{\text{frequent items}\};\]

\[
\text{for } (k = 1; \ L_k \neq \emptyset; \ k++) \ \text{do begin} \\
\ C_{k+1} = \text{candidates generated from } L_k; \\
\ \text{for each transaction } t \text{ in database do} \\
\ \ \text{increment the count of all candidates in } C_{k+1} \text{ that are contained in } t \\
\ L_{k+1} = \text{candidates in } C_{k+1} \text{ with min_support} \\
\ \text{end} \\
\text{return } \bigcup_k L_k; \]
The Apriori Algorithm -- Example

Database D

TID Items
100 1 3 4
200 2 3 5
300 1 2 3 5
400 2 5

C1

{1} 2
{2} 3
{3} 3
{4} 1
{5} 3

L1

{1} 2
{2} 3
{3} 3
{4} 1
{5} 3

C2

{12} 1
{13} 2
{15} 1
{23} 2
{25} 3
{35} 2

L2

{12} 1
{13} 2
{15} 1
{23} 2
{25} 3
{35} 2

C3

{235} 2

L3

{235} 2

Note: \{1,2,3\} and \{2,3,5\} not in C3

Support > 1

Principles of Knowledge Discovery in Data
University of Alberta
© Dr. Osmar R. Zaïane, 1999, 2007

Apriori-Gen Algorithm – Clothing Example

Given: 20 clothing transactions; \( s=20\% \), \( c=50\% \)

Generate association rules using the Apriori algorithm

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items</th>
<th>Transaction</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Blouse, Jeans, Shoes, Shorts, Skirt, TShirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>Blouse, Jeans, Shoes, Skirt, TShirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>Jeans, TShirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td>Jeans, Shoes, TShirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td>Jeans, Shorts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t6</td>
<td>Shoes, TShirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td>Jeans, Skirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t8</td>
<td>Jeans, Shoes, Shorts, TShirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t9</td>
<td>Jeans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t10</td>
<td>Jeans, Shoes, Shorts, TShirt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t11</td>
<td>TShirt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scan1: Find all 1-itemsets. Identify the frequent ones.
Candidates: Blouse, Jeans, Shoes, Shorts, Skirt, TShirt
Support: 3/20 14/20 10/20 5/20 6/20 14/20
Frequent (Large): Jeans, Shoes, Shorts, Skirt, Tshirt
Join the frequent items – combine items with each other to generate candidate pairs

Clothing Example – cont.1

Scan2: 10 candidate 2-itemsets were generated. Find the frequent ones.

<table>
<thead>
<tr>
<th>Scan</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{Blouse, Jeans, (Shoes, Shorts, Skirt, TShirt)}</td>
</tr>
<tr>
<td>2</td>
<td>{Jeans, Shoes, Shorts, Skirt, TShirt, Skirt, TShirt}</td>
</tr>
<tr>
<td>3</td>
<td>{Jeans, Shoes, Shorts, Skirt, TShirt, Skirt, TShirt}</td>
</tr>
<tr>
<td>4</td>
<td>{Jeans, Shoes, Shorts, Skirt, TShirt, Skirt, TShirt}</td>
</tr>
</tbody>
</table>

7 frequent itemsets are found out of 10.

Clothing Example – cont.2

The next step is to use the large itemsets and generate association rules

\( c=50\% \)

The set of large itemsets is

\( L=\{\text{Jeans}, \text{Shoes}, \text{Shorts}, \text{Skirt}, \text{TShirt}, \text{Jeans, Shoes}, \text{Jeans, Shorts}, \text{Jeans, TShirt}, \text{Shoes, Shorts}, \text{Shoes, TShirt}, \text{Shorts, TShirt}, \text{Skirt, TShirt}, \text{Jeans, Shoes, Shorts}, \text{Jeans, Shoes, TShirt}, \text{Jeans, Shorts, TShirt}, \text{Shoes, Shorts, TShirt}, \text{Jeans, Shoes, Shorts, TShirt}\} \)

We ignore the first 5 as they do not consist of 2 nonempty subsets of large itemsets. We test all the others, e.g.:

\[
\text{confidence} (\text{Jeans} \rightarrow \text{Shoes}) = \left( \frac{\text{support}(\text{Jeans}, \text{Shoes})}{\text{support}(\text{Jeans})} \right) = \left( \frac{7/20}{14/20} \right) = 50\% \geq c
\]

etc.

See Slide 17
Lecture Outline

Part I: Concepts (30 minutes)
  • Basic concepts
  • Support and Confidence
  • Naïve approach

Part II: The Apriori Algorithm (30 minutes)
  • Principles
  • Algorithm
  • Running Example

Part III: The FP-Growth Algorithm (30 minutes)
  • FP-tree structure
  • Running Example

Part IV: More Advanced Concepts (30 minutes)
  • Database layout and space search approach
  • Other types of patterns and constraints

Problems with Apriori

• Generation of candidate itemsets are expensive
  (Huge candidate sets)
  • $10^4$ frequent 1-itemset will generate $10^7$ candidate 2-itemsets
  • To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.

• High number of data scans

Frequent Pattern Growth

• First algorithm that allows frequent pattern mining without generating candidate sets
• Requires Frequent Pattern Tree

FP-Growth

• Grow long patterns from short ones using local frequent items
  – “abc” is a frequent pattern
  – Get all transactions having “abc”: DB|abc
  – “d” is a local frequent item in DB|abc → abcd is a frequent pattern

Frequent Pattern Tree

• Prefix tree.
• Each node contains the item name, frequency and pointer to another node of the same kind.
• Frequent item header that contains item names and pointer to the first node in FP tree.
Database Compression Using FP-tree (on T10I4D100k)

Frequent Pattern Tree

Original Transaction | Ordered frequent items
--- | ---
B, F, H, J, O | F, B
A, F, C, E, L, P, M, N | C, B, P
F, M, C, B, A | F, C, A, M
F, B, D | F, B

F:5, C:5, A:4, B:4, M:4, P:3  Required Support: 3

Frequent Pattern Tree

Required Support: 3
Frequent Pattern Tree

- F, C, A, M, P
- F, C, A, B, M
- F, B
- C, B, P
- F, C, A, M, P
- C, A, M
- F, B

F:4
C:3
M:3
A:3
B:3
P:3

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B

F:4
C:3
M:2
A:3
B:1
P:2

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B

F:4
C:3
M:1
A:1
B:1
P:1

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B

F:4
C:3
M:2
A:3
B:1
P:2

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B
Frequent Pattern Tree

Mining Frequent Patterns with FP-Tree

3 Major Steps

Starting the processing from the end of list L:

Step 1:
Construct conditional pattern base for each item in the header table

Step 2:
Construct conditional FP-tree from each conditional pattern base

Step 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.

Frequent Pattern Growth

Recursively build the A, C and F conditional trees.
Another Example: Construct FP-tree from a Transaction Database

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, F-List
3. Scan DB again, construct FP-tree

**Header Table**

<table>
<thead>
<tr>
<th>Item</th>
<th>Frequency</th>
<th>Head</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**

- p  → fc:2, cb:1
- m  → fc:2, fc:1
- a  → fc:3
- c  → fc:3

**Properties of Step 1**

- 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 1: Construct Conditional Pattern Base**

- Starting at the frequent-item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of transformed prefix paths of that item to form a conditional pattern base

**Step 2: Construct Conditional FP-tree**

- For each pattern base
  - Accumulate the count for each item in the base
  - Construct the conditional FP-tree for the frequent items of the pattern base
### Conditional Pattern Bases and Conditional FP-Tree

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

**order of L**

### Principles of FP-Growth

- **Pattern growth property**
  - Let \( \alpha \) be a frequent itemset in DB, \( B \) be \( \alpha \)'s conditional pattern base, and \( \beta \) be an itemset in \( B \). Then \( \alpha \cup \beta \) is a frequent itemset in DB iff \( \beta \) is frequent in \( B \).
- **Is “fca bm” a frequent pattern?**
  - “fca” is a branch of \( m \)'s conditional pattern base
  - “b” is NOT frequent in transactions containing “fca”
  - “bm” is NOT a frequent itemset.

### Single FP-tree Path Generation

- Suppose an FP-tree \( T \) has a single path \( P \). The complete set of frequent pattern of \( T \) can be generated by enumeration of all the combinations of the sub-paths of \( P \)

\[
\text{m-conditional FP-tree}
\]

- All frequent patterns concerning \( m \)
  - \( m \),\( fn, cm, am, fcm, fam, cam, fcam \)

- \( m \)-conditional FP-tree
Discussion (1/2)

- Association rules are typically sought for very large databases → efficient algorithms are needed
- The Apriori algorithm makes 1 pass through the dataset for each different itemset size
  - The maximum number of database scans is k+1, where k is the cardinality of the largest large itemset (4 in the clothing ex.)
  - Potentially large number of scans – weakness of Apriori
- Sometimes the database is too big to be kept in memory and must be kept on disk
- The amount of computation also depends on the min.support; the confidence has less impact as it does not affect the number of passes
- Variations
  - Using sampling of the database
  - Using partitioning of the database
  - Generation of incremental rules

Discussion (2/2)

- Choice of minimum support threshold
  - Lowering support threshold results in more frequent itemsets
  - This may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - More space is needed to store support count of each item
  - If number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - Since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - Transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Lecture Outline

Part I: Concepts  (30 minutes)
- Basic concepts
  - Support and Confidence
- Naïve approach

Part II: The Apriori Algorithm  (30 minutes)
- Principles
- Algorithm
- Running Example

Part III: The FP-Growth Algorithm  (30 minutes)
- FP-tree structure
- Running Example

Part IV: More Advanced Concepts  (30 minutes)
- Database layout and space search approach
- Other types of patterns and constraints

Other Frequent Patterns

- Frequent pattern \( \{a_1, \ldots, a_{100}\} \rightarrow (100)^1 + (100)^2 + \ldots + (100)^{100} = 2^{100} - 1 = 1.27 \times 10^{30} \) frequent sub-patterns!

- Frequent Closed Patterns
- Frequent Maximal Patterns
- All Frequent Patterns

Maximal frequent itemsets ⊆ Closed frequent itemsets ⊆ All frequent itemset
**Frequent Closed Patterns**

- For frequent itemset $X$, if there exists no item $y$ such that every transaction containing $X$ also contains $y$, then $X$ is a frequent closed pattern.
- In other words, frequent itemset $X$ is closed if there is no item $y$, not already in $X$, that always accompanies $X$ in all transactions where $X$ occurs.
- Concise representation of frequent patterns. Can generate all frequent patterns with their support from frequent closed ones.
- Reduce number of patterns and rules
- N. Pasquier et al. In ICDT’99

**Frequent Maximal Patterns**

- Frequent itemset $X$ is maximal if there is no other frequent itemset $Y$ that is a superset of $X$.
- In other words, there is no other frequent pattern that would include a maximal pattern.
- More concise representation of frequent patterns but the information about supports is lost.
- Can generate all frequent patterns from frequent maximal ones but without their respective support.
- R. Bayardo. In SIGMOD’98

**Maximal vs. Closed Itemsets**

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
</tr>
<tr>
<td>2</td>
<td>ABCD</td>
</tr>
<tr>
<td>3</td>
<td>ABC</td>
</tr>
<tr>
<td>4</td>
<td>ACDE</td>
</tr>
<tr>
<td>5</td>
<td>DE</td>
</tr>
</tbody>
</table>

Set of transaction Ids

- Not supported by any transaction
- Closed and maximal
- Closed but not maximal
- Frequent Pattern Border
- Minimum support = 2

**Frequent Closed Patterns**

- $\{abcd\}$
- $\{abc\}$
- $\{bd\}$

Transactions
- Support = 2

**Frequent Maximal Patterns**

- $\{abcd\}$
- $\{abc\}$
- $\{bd\}$

Transactions
- Support = 2
Mining the Pattern Lattice

- **Breadth-First**
  - It uses current items at level \( k \) to generate items of level \( k+1 \) (many database scans)

- **Depth-First**
  - It uses a current item at level \( k \) to generate all its supersets (favored when mining long frequent patterns)

- **Hybrid approach**
  - It mines using both direction at the same time

- **Leap traversal approach**
  - Jumps to selected nodes

There is also the notion of:
- **Top-down** (level \( k \) then level \( k+1 \))
- **Bottom-up** (level \( k+1 \) then level \( k \))
**Leap Traversal Example**

**Constraint-based Data Mining**

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!
- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface)
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
  - System optimization: explores such constraints for efficient mining—constraint-based mining

**Restricting Association Rules**

- Useful for interactive and ad-hoc mining
- Reduces the set of association rules discovered and confines them to more relevant rules.
- **Before mining**
  - Knowledge type constraints: classification, etc.
  - Data constraints: SQL-like queries (DMQL)
  - Dimension/level constraints: relevance to some dimensions and some concept levels.
- **While mining**
  - Rule constraints: form, size, and content.
  - Interestingness constraints: support, confidence, correlation.
- **After mining**
  - Querying association rules

**Constrained Frequent Pattern Mining: A Mining Query Optimization Problem**

- Given a frequent pattern mining query with a set of constraints C, the algorithm should be
  - sound: it only finds frequent sets that satisfy the given constraints C
  - complete: all frequent sets satisfying the given constraints C are found
- A naive solution
  - First find all frequent sets, and then test them for constraint satisfaction
- More efficient approaches:
  - Analyze the properties of constraints comprehensively
  - Push them as deeply as possible inside the frequent pattern computation.
Rule Constraints in Association Mining

- Two kinds of rule constraints:
  - Rule form constraints: meta-rule guided mining.
    - \( P(x, y) \land Q(x, w) \Rightarrow \text{takes}(x, \text{“database systems”}) \).
  - Rule content constraint: constraint-based query optimization (where and having clauses) (Ng, et al., SIGMOD’98).
    - \( \sum(\text{LHS}) < 100 \land \min(\text{LHS}) > 20 \land \text{count}(\text{LHS}) > 3 \land \sum(\text{RHS}) > 1000 \)

- 1-variable vs. 2-variable constraints
  (Lakshmanan, et al. SIGMOD’99):
    - 1-var: A constraint confining only one side (L/R) of the rule, e.g., as shown above.
    - 2-var: A constraint confining both sides (L and R).
      - \( \sum(\text{LHS}) < \min(\text{RHS}) \land \max(\text{RHS}) < 5 \ast \sum(\text{LHS}) \)

Anti-Monotonicity in Constraint-Based Mining

- Anti-monotonicity
  - When an itemset \( S \) violates the constraint, so does any of its supersets
    - \( \sum(\text{S.Price}) \leq v \) is anti-monotone
    - \( \sum(\text{S.Price}) \geq v \) is not anti-monotone

- Example. C: \( \text{range(} S\text{.profit}) \leq 15 \) is anti-monotone
  - Itemset \( ab \) violates C
  - So does every superset of \( ab \)

Monotonicity in Constraint-Based Mining

- Monotonicity
  - When an itemset \( S \) satisfies the constraint, so does any of its supersets
    - \( \sum(\text{S.Price}) \geq v \) is monotone
    - \( \min(\text{S.Price}) \leq v \) is monotone

- Example. C: \( \text{range(} S\text{.profit}) \geq 15 \) is monotone
  - Itemset \( ab \) satisfies C
  - So does every superset of \( ab \)

Which Constraints Are Monotone or Anti-Monotone?

<table>
<thead>
<tr>
<th>SQL-based Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v \in S )</td>
</tr>
<tr>
<td>( S \supseteq V )</td>
</tr>
<tr>
<td>( S \subseteq V )</td>
</tr>
<tr>
<td>( \min(S) \geq v )</td>
</tr>
<tr>
<td>( \min(S) \leq v )</td>
</tr>
<tr>
<td>( \max(S) \geq v )</td>
</tr>
<tr>
<td>( \max(S) \leq v )</td>
</tr>
<tr>
<td>( \text{count}(S) \geq v )</td>
</tr>
<tr>
<td>( \text{count}(S) \leq v )</td>
</tr>
<tr>
<td>( \sum(S) \geq (a \in S, a \geq 0) )</td>
</tr>
<tr>
<td>( \sum(S) \leq (a \in S, a \geq 0) )</td>
</tr>
<tr>
<td>( \text{range}(S) \geq v )</td>
</tr>
<tr>
<td>( \text{range}(S) \leq v )</td>
</tr>
<tr>
<td>( \text{support}(S) \geq \xi )</td>
</tr>
<tr>
<td>( \text{support}(S) \leq \xi )</td>
</tr>
</tbody>
</table>
State Of The Art

Constraint pushing techniques have been proven to be effective in reducing the explored portion of the search space in constrained frequent pattern mining tasks.

Anti-monotone constraints:
- Easy to push …
- Always profitable to do …

Monotone constraints:
- Hard to push …
- Should we push them, or not?

Finding Maximal using leap traversal approach

Minimum support = 2

TID | Items
--- | ---
1,3 | ABC
2 | ABCD
3 | ABC
4 | ACDE
5 | DE

Intersect ABCD with ACDE

Step 1: Define actual paths (Mark paths)

Step 2: Intersect non frequent marked paths with each others
Finding Maximal using leap traversal approach

Step3: Remove non frequent paths, or frequent paths that have superset of other frequent paths

Minimum support = 2

Empirical Tests

Support Size of Largest | Total Candidates Created
--- | ---
Connect database (long Frequent Patterns) | 0.00 2.00 4.00 6.00 8.00 T10I4D100K (Short Frequent Patterns) | 0.01 0.0075 0.005 0.0025 0.001 0.0005
Chess (Mixed Length Frequent Patterns) | 0.00 2.00 4.00 6.00 8.00

Total Candidates Created

| Connect | Chess | T10I4D100K
--- | --- | ---
95 | 3 | 375 385 20 10 29
90 | 4 | 463 561 262 124 291
500 | 5 | 569 1073 1492 731 1546250 8 717 7703 36994 36309 26666100 10 797 27532 264645 458275 200113
50 | 6 | 839 53385 1129779 4923723 1067050
95 | 7 | 13 10 1296 27/89 1590 1197
90 | 8 | 16 2692 25871 9681 4319
500 | 9 | 17 1712 7703 38050 200112
100 | 10 | 19 2132 29485 456275 200113
50 | 10 | 20 2692 25871 9681 4319

When to use a Given Strategy

- Breadth First
  - Suitable for short frequent patterns
  - Unsuitable for long frequent patterns
- Depth First
  - Suitable for long frequent patterns
  - In general not scalable when long candidate patterns are not frequent
- Leap Traversal
  - Suitable for cases having short and long frequent patterns simultaneously
Transactional Layouts

- **Horizontal Layout**
  Each transaction is recorded as a list of items
  
  Candidacy generation can be removed (FP-Growth)
  
  Superfluous Processing

- **Vertical Layout**
  Tid-list is kept for each item
  
  Minimize Superfluous Processing
  
  Candidacy generation is required

- **Inverted Matrix Layout**
  Minimize Superfluous Processing
  
  Candidacy generation can be reduced
  
  Appropriate for Interactive Mining

**El-Hajj and Zaiane, ACM SIGKDD’03**
Why The Matrix Layout?

Interactive mining

Changing the support level means expensive steps (whole process is redone)

Why The Matrix Layout?

Repetitive tasks, (I/O) readings (Superfluous Processing)

Support > 4

Frequent 1-itemsets (A, B, C, D, E, F)
Non frequent items (G, H, I, J, K, L, M, N, O, P, Q, R)

Support > 9

Frequent 1-itemsets (A, B, C)
Non frequent items (D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R)

Transactional Layouts

• Inverted Matrix Layout

Support > 4

Border Support
## The Algorithms (State of the Art)

### All
- Apriori, FP-Growth, COFI*, ECLAT, Leap

### Closed
- CHARM, CLOSET+, COFI-CLOSED, Leap

### Maximal
- MaxMiner, MAFIA, GENMAX, COFI-MAX, Leap

---

### All-Apriori

### Apriori

- Repetitive I/O scans
- Huge Computation to generate candidate items

---

R. Agrawal, R. Srikant, VLDB'94
All-Apriori

Problems with Apriori

• Generation of candidate itemsets are expensive (Huge candidate sets)
  • $10^4$ frequent 1-itemset will generate $10^7$ candidate 2-itemsets
  • To discover a frequent pattern of size 100, e.g., \{a_1, a_2, \ldots, a_{100}\}, one needs to generate $2^{100} \approx 10^{30}$ candidates.

• High number of data scans

Frequent Pattern Growth

• First algorithm that allows frequent pattern mining without generating candidate sets
• Requires Frequent Pattern Tree

All-FP-Growth

FP-Growth

2 I/O scans
Reduced candidacy generation
High memory requirements
Claims to be 1 order of magnitude faster than Apriori

FP-Tree Recursive conditional trees and FP-Trees

J. Han, J. Pei, Y. Yin, SIGMOD'00

All-COFI

COFI algorithm big picture

COFI

2 I/O scans
reduced candidacy generation
Small memory footprint

FP-Tree

Patterns

El-Hajj and Zaïane, DaWak’03

All-COFI

Co-Ocurrences Frequent Item tree

Start with item P:
Find Locally frequent items with respect to P: C3

PC:3 frequent-path-base
All subsets of PC:3 are frequent and have the same support

Support
Participation

El-Hajj and Zaïane, DaWak’03
Then with item M: Find Locally frequent items with respect to M:
A:4, C:4, F:3

All-COFI

Co-Occurrences Frequent Item tree

How to mine frequent-path-bases

Three approaches:
1: Bottom-Up

Support of any pattern is the summation of the supports of its supersets of frequent-path-bases
**All-COFI**

**Co-Occurrences Frequent Item tree**

How to mine frequent-path-bases

Three approaches:

2: Top-down

Support of any pattern is the summation of the supports of its superset of frequent-path-bases

FCA: 3

CA: 1

---

**ECLAT**

- For each item, store a list of transaction ids (tids)

**Horizontal Data Layout**

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A,B,E</td>
</tr>
<tr>
<td>2</td>
<td>B,C,D</td>
</tr>
<tr>
<td>3</td>
<td>C,E</td>
</tr>
<tr>
<td>4</td>
<td>A,C,D</td>
</tr>
<tr>
<td>5</td>
<td>A,B,C,D</td>
</tr>
<tr>
<td>6</td>
<td>A,E</td>
</tr>
<tr>
<td>7</td>
<td>A,B</td>
</tr>
<tr>
<td>8</td>
<td>A,B,C</td>
</tr>
<tr>
<td>9</td>
<td>A,C,D</td>
</tr>
<tr>
<td>10</td>
<td>B</td>
</tr>
</tbody>
</table>

**Vertical Data Layout**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TID-list**

J.M. Zaki, IEEE Transactions on Knowledge and Data Engineering, 00

---

**All-COFI**

**Co-Occurrences Frequent Item tree**

How to mine frequent-path-bases

Three approaches:

3: Leap-Traversal

1) Intersect non frequent path bases

FCA: 3 \( \cap \) CA: 1 = CA

2) Find subsets of the only frequent paths (sure to be frequent)

3) Find the support of each pattern

Support of any pattern is the summation of the supports of its superset of frequent-path-bases

---

**ECLAT**

- Determine support of any k-itemset by intersecting tid-lists of two of its (k-1) subsets.
ECLAT

Find all frequent patterns with respect to item A
AB, AC, ..., ABC, ABD, ACD, ABCD, ....

Then it finds all frequent patterns with respect to item B
BC, BD, ..., BCD, BDE, BCDE, ....

- 3 traversal approaches:
  - top-down, bottom-up and hybrid
- Advantage: very fast support counting, Few scans of database (best case 2)
- Disadvantage: intermediate tid-lists may become too large for memory

All-ECLAT
Find all frequent patterns with respect to item A
AB, AC, ..., ABC, ABD, ACD, ABCD, ....

Then it finds all frequent patterns with respect to item B
BC, BD, ..., BCD, BDE, BCDE, ....

Other Algorithms for Other Patterns

Algorithms for Closed Patterns and Maximal Patterns will be discussed in class with paper presentations.

Which algorithm is the winner?

Not clear yet

With relatively small datasets we can find different winners

1. By using different datasets
2. By changing the support level
3. By changing the implementations

Which algorithm is the winner?

What about Extremely large datasets (hundreds of millions of transactions)?

Most of the existing algorithms do not run on such sizes

Vertical approaches and Bitmaps approaches cannot load the transactions in Main Memory

Repetitive approaches cannot keep scanning these huge databases many times

Requirements: We need algorithms that

1) do not require multiple scans of the database
2) Leave small footprint in Main Memory at any given time