Acknowledgements

- Some parts of these slides are modified from
  - C. Clifton & W. Aref, Purdue University
Outline

- Association Rule Mining – Basic Concepts
- Association Rule Mining Algorithms:
  - Single-dimensional Boolean associations
  - Multi-level associations
  - Multi-dimensional associations
- Association vs. Correlation
- Adding constraints
- Applications/extensions of frequent pattern mining
- Summary

Motivation

Finding regularities/patterns in data

- Market basket analysis:
  - What products were often purchased together?
  - What are the subsequent purchases after buying a PC?
- Bioinformatic:
  - What kinds of DNA are sensitive to this new drug?
- Web mining:
  - Which web documents are similar to a given one?
Association Rule Mining

Finds interesting relationships among data items

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

**Form:** Itemset-A \( \rightarrow \) Itemset-B \[ \% \text{support, } \% \text{confidence} \]

E.g., Bread \( \Rightarrow \) Milk [support = 2\%, confidence = 70\%]

- 2\% of all transaction databases, bread and milk are bought together
- 70\% of customers who bought bread also bought milk

- Rules are interesting (or strong) if they satisfy
  - A min support threshold
  - A minimum confidence threshold

Importance

- Foundation for many essential data mining tasks
  - Association, correlation, causality
  - Sequential patterns, temporal association, partial periodicity, spatial and multimedia association
  - Associative classification, cluster analysis, iceberg cube, fascicles (semantic data compression)

- Broad applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis
  - Web log (click stream) analysis,
  - DNA sequence analysis, etc.
Basic Concepts

- $I$ = set of all possible items
- A transaction, $T$ is set of items. $T \subset I$
- $D = \text{data set of all transactions}$
- Association Rule: $A \Rightarrow B \,[\%\text{support, } \%\text{confidence}]$, where $A \subset I$, $B \subset I$ and $A \cap B = \emptyset$
  - Support($A \Rightarrow B$) – prob. that a transaction contains both $A$ and $B$
    $$= P(A \cap B) = \frac{|A \cap B|}{|D|}$$
  - Confidence($A \Rightarrow B$) – prob. that a trans. having $A$ also contains $B$
    $$= P(B|A) = \frac{|A \cap B|}{|A|} = \frac{\text{Support}(A \Rightarrow B)}{P(A)} = \frac{\text{Support-count}(A \cap B)}{\text{support-count}(A)}$$

Example

- $I = \{A, B, C, D, E, F\}$
- Min-support = 50%
- Min-confidence = 50%
- $A \Rightarrow C \ [50\%, \ 66.7\%]$
  - Support($A \Rightarrow C$) = 2/4
  - Confidence($A \Rightarrow C$) = 2/3
- $C \Rightarrow A \ [50\%, \ 100\%]$
  - Support($C \Rightarrow A$) = 2/4
  - Confidence($C \Rightarrow A$) = 2/2
- What about $A \Rightarrow B$?
  \{$A, C\} \Rightarrow B$, etc.?
Types of association rules

- Single vs. multi-dimensional rules
  - $\text{Buys}(X, \{\text{bread, egg}\}) \Rightarrow \text{Buys}(X, \text{milk})$
  - $\text{Buys}(X, \text{beer}) \Rightarrow \text{Age}(X, \text{over-20})$
  - (Bread, Egg) $\Rightarrow$ (milk)

- Boolean vs. quantitative rules
  - $\text{Buys-bread-and-egg} \Rightarrow \text{Buys-milk}$
  - $\text{Age}(X, 1..15) \Rightarrow \text{Income}(X, 0..2K)$

- Single vs. multi-level rules
  - $\text{Buys}(X, \text{bread}) \Rightarrow \text{Buys}(X, \text{milk})$
  - $\text{Buys}(X, \text{bread}) \Rightarrow \text{Buys}(X, \text{diary-product})$
    (given milk is a kind of diary-product)

Outline

- Association Rule Mining – Basic Concepts
- Association Rule Mining Algorithms:
  - Single-dimensional Boolean associations
  - Multi-level associations
  - Multi-dimensional associations
- Association vs. Correlation
- Adding constraints
- Applications/extensions of frequent pattern mining
- Summary
Mining single dimensional Boolean rules

- **Input:**
  - A database of transactions
  - Each transaction is a list of items
- **Output:**
  - Rules that associate the presence of one itemset with the other (i.e., LHS & RHS of rule)
  - No restrictions on number of items in both sets

### Example

<table>
<thead>
<tr>
<th>Trans-id</th>
<th>Items bought</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>

- I = \{A, B, C, D, E, F\}
- Min-support = 50%
- Min-confidence = 50%
- \(A \Rightarrow C\) [50%, 66.7%]
  - Support(\(A \Rightarrow C\)) = 2/4
  - Confidence(\(A \Rightarrow C\)) = 2/3
- \(C \Rightarrow A\) [50%, 100%]
  - Support(\(C \Rightarrow A\)) = 2/4
  - Confidence(\(C \Rightarrow A\)) = 2/2
- What about \(A \Rightarrow B\)?
  - \(\{A, C\} \Rightarrow B\), etc.?
Frequent pattern

Frequent pattern
- pattern (itemset, sequence, etc.) that occurs frequently in a database [AIS93]
- I.e., those with % of occurrences > a minimum support

<table>
<thead>
<tr>
<th>Transaction-id</th>
<th>Items bought</th>
<th>Frequent pattern</th>
<th>Sup</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>A, B, C</td>
<td>{A}</td>
<td>3</td>
<td>75%</td>
</tr>
<tr>
<td>200</td>
<td>A, C</td>
<td>{B}</td>
<td>2</td>
<td>50%</td>
</tr>
<tr>
<td>300</td>
<td>A, D</td>
<td>{C}</td>
<td>2</td>
<td>50%</td>
</tr>
<tr>
<td>400</td>
<td>B, E, F</td>
<td>{A, C}</td>
<td>2</td>
<td>50%</td>
</tr>
</tbody>
</table>

Min. support 50%

Note: Omit {A, B}, {B, C}, {B, E} etc.

What we really need for mining

- Goal: Rules with high support/confidence
- How to compute?
  - Support: Find sets of items that occur frequently
  - Confidence: Find frequency of subsets of supported itemsets
- If we have all frequently occurring sets of items (frequent itemsets), we can compute support and confidence!

How can we compute frequent itemsets efficiently?
The Apriori Algorithm

- Approach – level-wise generate and test
- Basic steps:
  - Iterate with increasing size of itemsets
    - Generate candidates – possible itemsets
    - Test for “large” candidates – itemsets that satisfy min support, i.e., frequent itemsets
  - Use the frequent itemsets to generate association rules

Apriori pruning principle

- Any subset of a frequent itemset must be frequent
- A super set of an infrequent itemset is infrequent

Why?
- E.g., \# \{A and B and C\} transactions ≤ \# \{A and B\} transactions
- \{A, B, C\} is frequent → \{A, B\} must be frequent
  - \{A, B\} is infrequent → \{A, B, C\} is infrequent
The Apriori Algorithm (Cont)

- Method:
  - generate length (k+1) candidate itemsets, $C_{k+1}$ from length k frequent itemsets, $L_k$, and
  - test candidates to obtain $L_{k+1}$

- Assume itemsets are in lexicographical order


---

The Apriori Algorithm—An Example (Min Sup 50%)

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>

$C_2$

Note: {A, B, C}, {A, B, E}, {A, C, E} are not in $C_3$
Generating rules

- For each frequent itemset \( I \)
  - For each non-empty proper subset \( S \) of \( I \)
    
    Output rule: \( S \Rightarrow (I - S) \) if its confidence > threshold

E.g., consider \( L = \{B, BCE\} \) (write “AB” for \( \{A, B\} \))

Assume min thresholds are satisfied, from a frequent item set BCE, output

\[
\begin{align*}
B & \Rightarrow CE \\
C & \Rightarrow BE \\
E & \Rightarrow BC \\
BC & \Rightarrow E \\
BE & \Rightarrow C \\
CE & \Rightarrow B
\end{align*}
\]

No rule generated from frequent item set B since B has no non-empty proper subset.

The Apriori Algorithm—An 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>

C1

<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>

1st scan

L1

<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

C2

<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>

3rd scan

L3

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

C3

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

C4 is empty

Note: \{A, B, C\}, \{A, B, E\}, \{A, C, E\} are not in C3
The Apriori Algorithm—An 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>

Min-Support 50%
Min-Confidence 100%

$L_1$

<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>

$L_2$

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

$L_3$

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

Min-Support 50%, Min-Confidence 100%:

| A à C  | 2/4%, 2/2% |
| C à A  | 2/4%, 2/3% |
| B à C  | 2/4%, 2/3% |
| C à B  | 2/4%, 2/3% |
| B à E  | 3/4%, 3/3% |
| E à B  | 3/4%, 3/3% |
| C à E  | 2/4%, 2/3% |
| E à C  | 2/4%, 2/3% |
| B Æ CE | 2/4%, 2/3% |
| C Æ BE | 2/4%, 2/3% |
| E Æ BC | 2/4%, 2/3% |
| BC Æ E | 2/4%, 2/3% |
| CE Æ B | 2/4%, 2/3% |
| BE Æ C | 2/4%, 2/3% |
The Apriori Algorithm

Pseudo-code:

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

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

\[ \text{for} \ (k = 1; \ L_k \neq \emptyset; \ k++) \ \text{do begin} \]

\[ C_{k+1} = \text{candidates generated from } L_k; \]

\[ \text{for each transaction } t \text{ in database do} \]

\[ \text{increment the count of all candidates in } C_{k+1} \]

\[ \text{that are contained in } t \]

\[ L_{k+1} = \text{candidates in } C_{k+1} \text{ with min_support} \]

\[ \text{end} \]

\[ \text{return } \bigcup_k L_k; \] What is the maximum number of scans of the DB?

Important details

- How to generate candidates?
- How to count supports of candidates?
Candidate generation

To obtain $C_{k+1}$ from $L_k$

- **Self joining** $L_k$
  
  Example: $L_3=\{ABC, ABD, ACD, ACE, BCD\}$
  
  Self-joining: $L_3 \times L_3$
  
  - $ABC$ and $ABD \implies ABCD$
  - $ACD$ and $ACE \implies ACDE$
  - No more, thus potential $C_4 = \{ABCD, ACDE\}$

- **Pruning**
  
  Example: remove $ACDE$ since $CDE$ is not in $L_3$
  
  $C_4 = \{ABCD\}$

Candidate generation (cont)

- Assume the items in $L_k$ are listed in an order

- **Step 1: self-joining** $L_k$ (requires sharing first $k-1$ items)
  
  insert into $C_{k+1}$
  
  select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_1, q.item_2, \ldots, q.item_k$
  
  from $L_k p, L_k q$
  
  where $p.item_i=q.item_i, \ldots, p.item_{k-1}=q.item_{k-1}, p.item_k < q.item_k$

- **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 $C_{k+1}$
The Apriori Algorithm—An 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

$L_1$

Itemset  sup
(A) 2
(B) 3
(C) 3
(D) 1
(E) 3

$L_2$

Itemset  sup
(A, C) 2
(B, C) 2
(B, E) 3
(C, E) 2

$L_3$

Itemset  sup
(B, C, E) 2

2nd scan

$L_2$

Itemset  sup
(A, B) 1
(A, C) 2
(A, E) 1
(B, C) 2
(B, E) 3
(C, E) 2

3rd scan

$L_3$

Itemset  sup
(A, C) 2
(B, C) 2
(B, E) 3
(C, E) 2

Note: {A, B, C}, {A, B, E}, {A, C, E} are not in $C_3$

Self joining $L_1$ requires sharing of first 0 item appended with two elements, each from the joined element $\rightarrow L_1 \times L_1$ gives $C_2$

Self joining $L_2$ requires sharing of first 1 item
- Only middle two sets gives $C_3$

Self joining $L_3$ requires sharing of first 2 items
- None is possible since $L_3$ has one entry

No pruning in any step

$L_4$ is empty

Note: {A, B, C}, {A, B, E}, {A, C, E} are not in $C_3$
Counting Supports of Candidates

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates

- 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*: provides hashing addresses

- Example: given a hash tree representing itemsets found so far and a transaction, find all the candidates contained in the transaction (and increment their counts)

---

Example

Leaves are itemsets of size three found so far

Transaction: 1 2 3 5 6

**Goal** is to scan thru the transaction and update support count of the itemsets
Example

Subtree function
1, 4, 7
3, 6, 9
2, 5, 8

Leaves are itemsets found so far
Transaction: 1 2 3 5 6

Example

Subtree function
1, 4, 7
3, 6, 9
2, 5, 8

Leaves are itemsets found so far
Transaction: 1 2 3 5 6
Apriori in SQL

- Hard to get good performance out of pure (SQL-92) based approaches alone
- Make use of object-relational extensions
  - Get orders of magnitude improvement

Efficiency issues

- Challenges:
  - Multiple scans of “large” transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
Improving Efficiency of Apriori

General ideas:
- Reduce number of candidates
- Reduce number of database scans/transactions
- Facilitate support counting of candidates

Reduce Candidates

- A hash-based technique to reduce $|C_k|$  
  - Idea: A $k$-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Example: $k = 2$
    - Items A, B, C, D, E are ordered as 1, 2, 3, 4, 5, respectively
    - For 2-item set $\{X,Y\}$, define a hash function,
      $$h(X, Y) = o(X) \times 10 + o(Y) \mod 7,$$
      where $o(X) = \text{order of } X$
### An 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>

During the 1\textsuperscript{st} scan of TDB to determine \( L_1 \) from \( C_1 \),
- Create a hash table
- omit 2-itemset in a bucket whose count < threshold 2

Items A, B, C, D, E are ordered as 1, 2, 3, 4, 5

\( h(X, Y) = o(X) \times 10 + o(Y) \mod 7 \)

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Bucket} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\text{Content} & AD & CE & BC & BE & AE & AC & \text{AC, CD} \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{Count} & 3 & 1 & 2 & 0 & 3 & 1 & 3 \\
\end{array}
\]

### The Apriori Algorithm—An 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>

During the 1\textsuperscript{st} scan to determine \( L_1 \) from \( C_1 \),
- Create a hash table
- omit 2-itemset in a bucket whose count < threshold 2

Items A, B, C, D, E are ordered as 1, 2, 3, 4, 5

\( h(X, Y) = o(X) \times 10 + o(Y) \mod 7 \)

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Itemset} & \{A\} & \{B\} & \{C\} & \{D\} & \{E\} \\
\hline
\text{sup} & 2 & 3 & 3 & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Itemset} & \{A, B\} & \{A, C\} & \{A, E\} & \{B, C\} & \{B, E\} & \{C, E\} \\
\hline
\text{sup} & 1 & 2 & 1 & 2 & 2 & 2 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Itemset} & \{A, B, C\} & \{A, B, E\} & \{A, C, E\} & \{B, C, E\} \\
\hline
\text{sup} & 2 \\
\end{array}
\]

\( C_2 \) is empty

Note: \{A, B, C\}, \{A, B, E\}, \{A, C, E\} are not in \( C_3 \)
Exploring Vertical Data Format

- Use tid-list, the list of transaction-ids containing an itemset
- Compression of tid-lists
  - Itemset A: t1, t2, t3, sup(A)=3
  - Itemset B: t2, t3, t4, sup(B)=3
  - Itemset AB: t2, t3, sup(AB)=2
- Major operation: intersection of tid-lists
- M. Zaki et al. New algorithms for fast discovery of association rules. In KDD’97
- P. Shenoy et al. Turbo-charging vertical mining of large databases. In SIGMOD’00

Improving Efficiency of Apriori

General ideas:
- Reduce number of candidates
- Reduce number of database scans/transactions
- Facilitate support counting of candidates
DIC: Reduce number of scans

- Once both A and D are determined frequent, the counting of AD begins
- Once all length-2 subsets of BCD are determined frequent, the counting of BCD begins
- Dynamic - The counting and adding of new candidates at any start point

**Itemset lattice**

S. Brin R. Motwani, J. Ullman, and S. Tsur. [Dynamic itemset counting](#) and implication rules for market basket data. In *SIGMOD’97*

---

Partition: Scan Database Only Twice

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns
Sampling for Frequent Patterns

- H. Toivonen. Sampling large databases for association rules. In VLDB ’96
- Select a sample of original database, mine frequent patterns within sample using Apriori
- Scan database once to verify frequent itemsets found in sample
- Scan database again to find missed frequent patterns

Improving Efficiency of Apriori

General ideas:
- Reduce number of candidates
- Reduce number of database scans/transactions
- Facilitate support counting of candidates
Bottleneck of Apriori

- Sources of computational cost
  - Multiple database scans
  - Long patterns → many scans and → huge number of candidates
  - To find frequent itemset $i_1i_2\ldots i_{100}$
    - # of scans: 100
    - # of Candidates $\sim 2^{100-1} = 1.27 \times 10^{30}$!

- Bottleneck: candidate-generation-and-test

- Can we avoid candidate generation?

Frequent Pattern Growth (FP-Growth)

- Method of mining frequent itemsets without candidate generation

- Ideas:
  - Use compact data structure: FP-tree to represent frequent itemsets from transactions
  - Grow long patterns from short ones using local frequent items
Construct FP-tree

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

\[
\text{min}\_\text{support} = 3
\]

\[
\text{F-list} = \{f, c, a, b, m, p\}
\]
Finding conditional pattern bases

- For each frequent item X, starting at X’s header table, follow the link to traverse the FP-tree.
- Accumulate all of transformed prefix paths of item X to form X’s conditional pattern base (with count of X).

<table>
<thead>
<tr>
<th>Item</th>
<th>cond. 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>

Item count:
- f: 4
- c: 4
- a: 3
- b: 3
- m: 3
- p: 3

min_support = 3

Finding 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Cond. pattern base</th>
<th>m-cond FP-tree</th>
<th>m-frequent patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>f:3</td>
<td>{}</td>
<td>fm:3, cm:3, am:3, fc:3, fam:3, cam:3, fcam:3</td>
</tr>
<tr>
<td>a</td>
<td>fc:3</td>
<td>f:2:3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>fca:1, f:1, c:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>fca:2, fcab:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>fcam:2, cb:1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Min. support = 3
## Finding 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Cond. pattern base</th>
<th>c-cond FP-tree</th>
<th>a-cond FP-tree</th>
<th>c-frequent patterns</th>
<th>a-frequent patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>f:3</td>
<td>{}</td>
<td>{}</td>
<td>fc:3</td>
<td>fa:3, ca:3</td>
</tr>
<tr>
<td>a</td>
<td>fc:3</td>
<td>f:3</td>
<td>f:3</td>
<td></td>
<td>fc:3</td>
</tr>
<tr>
<td>b</td>
<td>fc:1, f:1, c:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>fc:2, fcab:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>fcam:2, cb:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Min. support = 3

---

## Finding 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Cond. pattern base</th>
<th>b-cond FP-tree</th>
<th>b-frequent patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>f:3</td>
<td>{}</td>
<td>none</td>
</tr>
<tr>
<td>a</td>
<td>fc:3</td>
<td>f:3 c:1</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>fc:1, f:1, c:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>fc:2, fcab:1</td>
<td>c:1</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>fcam:2, cb:1</td>
<td>a:1</td>
<td></td>
</tr>
</tbody>
</table>

Min. support = 3
## Finding 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Cond. pattern base</th>
<th>p-cond FP-tree</th>
<th>p-frequent patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>f:3</td>
<td>{}</td>
<td>cp:3</td>
</tr>
<tr>
<td>a</td>
<td>fc:3</td>
<td>f:2</td>
<td>c:1</td>
</tr>
<tr>
<td>b</td>
<td>fca:1, f:1, c:1</td>
<td>c:2</td>
<td>b:1</td>
</tr>
<tr>
<td>m</td>
<td>fca:2, fcab:1</td>
<td>a:2</td>
<td>m:2</td>
</tr>
</tbody>
</table>

Min. support = 3

---

## Frequent patterns from FP-tree

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
<th>(ordered) frequent items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{a, b, c, d, 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, w}</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>

Min. support = 3

<table>
<thead>
<tr>
<th>Item</th>
<th>Cond. pattern base</th>
<th>Frequent patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>f:3</td>
<td>fc:3</td>
</tr>
<tr>
<td>a</td>
<td>fc:3</td>
<td>fa:3, ca:3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fca:3</td>
</tr>
<tr>
<td>b</td>
<td>fca:1, f:1, c:1</td>
<td>none</td>
</tr>
<tr>
<td>m</td>
<td>fca:2, fcab:1</td>
<td>fm:3, cm:3, am:3,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fam:3, fam:3, cam:3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fcam:3</td>
</tr>
<tr>
<td>p</td>
<td>fcam:2, cb:1</td>
<td>cp:3</td>
</tr>
</tbody>
</table>
Mining FP-trees

- Idea: Frequent pattern growth
  - Recursively grow frequent pattern by pattern and database partition

- Method (sketched, see the FP-growth algorithm in the text)
  - For each frequent pattern, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the above on each newly created conditional FP-tree
  - Single path will generate all the combinations of its sub-paths, each of which is a frequent pattern
  - Until the resulting FP-tree is empty, or it contains only a single path

Recursion on conditional FP-trees

```
\[
\begin{array}{l}
\text{Cond. pattern base of “am”: (fc:3)} \\
\text{f:3} \\
\text{c:3} \\
\text{a:3} \\
\text{m-conditional FP-tree}
\end{array}
\]

\[
\begin{array}{l}
\text{am-conditional FP-tree} \\
\text{f:3} \\
\text{c:3}
\end{array}
\]

\[
\begin{array}{l}
\text{Cond. pattern base of “cm”: (f:3)} \\
\text{f:3} \\
\text{c:3} \\
\text{m-conditional FP-tree}
\end{array}
\]

\[
\begin{array}{l}
\text{cm-conditional FP-tree} \\
\text{f:3}
\end{array}
\]

\[
\begin{array}{l}
\text{Cond. pattern base of “cam”: (f:3)} \\
\text{f:3} \\
\text{am-conditional FP-tree}
\end{array}
\]

\[
\begin{array}{l}
\text{cam-conditional FP-tree} \\
\text{f:3}
\end{array}
\]
### Building Block

- Suppose a (conditional) FP-tree $T$ has a shared single prefix-path
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node (e.g., $r_i$)
  - Concatenation of the mining results of the two parts

![Diagram showing the construction of $r_1 = \{\} \times \{a_jn_1, a_2n_2, a_3n_3\}$]

### Benefits of the FP-tree Structure

- **Complete**
  - Retain complete information for frequent patterns
  - Never break a long pattern of any transaction
- **Compact**
  - Non-redundant
    - Frequent patterns can be partitioned into subsets of F-list
    - E.g., for F-list=f-c-a-b-m-p, we can have patterns containing p, having m but no p, ...having c but no a nor b, m, p etc.
  - Reduce irrelevant info—in frequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the count field)
Scaling FP-growth

- FP-tree cannot fit in memory?
  \[ \rightarrow \text{DB projection} \]
  - First partition a database into a set of projected DBs
  - Then construct and mine FP-tree for each projected DB
- Parallel projection vs. Partition projection
  - Parallel projection is space costly

FP-Growth vs. Apriori

Data set T25I20D10K

![Run time graph](image)
Why Is FP-Growth the Winner?

- **Divide-and-conquer:**
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases

- **Other factors**
  - no candidate generation, no candidate test
  - compressed database: FP-tree structure
  - no repeated scan of entire database
  - basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

Outline

- Association Rule Mining – Basic Concepts
- Association Rule Mining Algorithms:
  - Single-dimensional Boolean associations
  - Multi-level associations
  - Multi-dimensional associations
- Association vs. Correlation
- Adding constraints
- Applications/extensions of frequent pattern mining
- Summary