• **Association rule mining**

• Mining (single-dimensional) association rules in transactional databases

• Mining Frequent Patterns Without Candidate Generation
Association rule mining

- Association analysis discovers **association rules** showing attribute-value conditions that occur frequently together in a set of data, e.g. market basket

- A rule has the form **body ⇒ head**
  1. Single-dimensional rules:
     \[
     \text{buys}(X, \text{“milk”}) \Rightarrow \text{buys}(X, \text{“sugar”})
     \]
  2. Multi-dimensional rules: ≥ 2 dimensions or predicates
     \[
     \text{age}(X, \text{“19-25”}) \land \text{income}(X, \text{“20K…29K”}) \Rightarrow \text{buys}(X, \text{“TV”})
     \]

First proposed by Agrawal, Imielinski and Swami [SIGMOD'93]

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

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

- **Motivation**: finding regularities in data
  - What products were often purchased together?— Milk and Sugar?!
  - 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?
Measures: Support and Confidence

- Itemset $X = \{x_1, \ldots, x_k\}$
- Find all the rules $X \rightarrow Y$ with min confidence and support
  - **support**, $s$, probability that a transaction contains $X \cup Y$
    \[
    \text{support}(X \rightarrow Y) = P(X \cup Y)
    \]
  - **confidence**, $c$, conditional probability that a transaction having $X$ also contains $Y$
    \[
    \text{confidence}(X \rightarrow Y) = P(Y/X) = \frac{\text{support}(\{X, Y\})}{\text{support}(\{X\})}
    \]

Let $\text{min\_support} = 50\%$, $\text{min\_conf} = 50\%$

- $A \rightarrow C$ (50%, 66.7%)  
- $C \rightarrow A$ (50%, 100%)

### Mining Association Rules–an Example

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items bought</th>
<th>Min. support 50%</th>
<th>Min. confidence 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>A, B, C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>A, C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>A, D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>B, E, F</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frequent pattern</td>
<td>Support</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A}</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{B}</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{C}</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A, C}</td>
<td>50%</td>
</tr>
</tbody>
</table>

For $A \Rightarrow C$:

\[
\text{support} = \text{support}(\{A, C\}) = 50% \\
\text{confidence} = \frac{\text{support}(\{A, C\})}{\text{support}(\{A\})} = \frac{2}{3} = 66.6%
\]
• Association rule mining

• Mining (single-dimensional) association rules in transactional databases

• Mining Frequent Patterns Without Candidate Generation

Associations extraction: Apriori

• Principal
  1. Any subset of a frequent itemset must be frequent
     • if \{bread, milk, sugar\} is frequent, so is \{bread, milk\}
     • every transaction having \{bread, milk, sugar\} also contains \{bread, milk\}
  2. Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated
     • if \{bread\} is infrequent then \{bread, milk\} should not be generated

• Method
  – generate (k+1) candidate itemsets from length k frequent itemsets and test the candidates against DB
  – use frequent itemsets to generate association rules
The Apriori Algorithm-An Example

**min_support=2**

**base D**

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

**Scan D**

**C_1**

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{4}</td>
<td>1</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
</tbody>
</table>

**L_1**

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

**Scan D**

**C_2**

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

**L_2**

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

**Scan D**

**C_3**

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

**L_3**

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

The Apriori Algorithm (1)

- \(C_k\): 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 begin**

- \(C_{k+1}\) = candidates generated from \(L_k\); \(\text{// self join of } L_k\)
- **for each** transaction \(t\) in database do
  - increment the count of all candidates in \(C_{k+1}\) that are contained in \(t\) \(\text{// computation of support}\)
- \(L_{k+1}\) = candidates in \(C_{k+1}\) with (at least) min_support

**end**

**return** \(\bigcup_k L_k\)
The Apriori Algorithm (2)

- How to generate candidates?
  - Step 1: self-joining $L_k$
  - Step 2: pruning
- Example of Candidate-generation
  - $L_3 = \{abc, abd, acd, ace, bcd\}$
  - Self-joining: $L_3 \times L_3$
    - $abcd$ from $abc$ and $abd$
    - $acde$ from $acd$ and $ace$
  - Pruning:
    - $acde$ is removed because $ade$ is not in $L_3$
  - $C_4 = \{abcd\}$

How to Generate Candidates?

- Suppose the items in $L_{k-1}$ are listed in an order
- Step 1: self-joining $L_{k-1}$
  
  insert into $C_k$
  select $p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}$
  from $L_{k-1} p, L_{k-1} q$
  where $p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}$
  
  $p.item_{k-1} < q.item_{k-1}$
- Step 2: pruning
  
  forall itemsets $c$ in $C_k$ do
    
    forall $(k-1)$-subsets $s$ of $c$ do
      
      if ($s$ is not in $L_{k-1}$) then delete $c$ from $C_k$
Generating Association rules from frequent itemsets

- For every large itemset \( l \), we find all non-empty subsets of \( l \)
- For every such subset \( s \), we output a rule of the form
  \[ s \Rightarrow (l - s) \] if the ratio of \( \text{support}(l)/\text{support}(s) \geq \minconf \)
- We consider all subsets of \( l \) to generate rules

- Example
  - \( l = \{2, 3, 5\} \)
  - Non empty subsets are \{2 3\} \{2 5\} \{3 5\} \{2\} \{3\} \{5\}
  - \( c \) is a confidence
    \[
    \begin{align*}
    &2 \text{ and } 3 \Rightarrow 5 \quad c=2/2=100\% \quad 2 \Rightarrow 3 \text{ and } 5 \quad c=2/3=67\% \\
    &2 \text{ and } 5 \Rightarrow 3 \quad c=2/3=67\% \quad 3 \Rightarrow 2 \text{ and } 5 \quad c=2/3=67\% \\
    &3 \text{ and } 5 \Rightarrow 2 \quad c=2/2=100\% \quad 5 \Rightarrow 2 \text{ and } 3 \quad c=2/3=67\%
    \end{align*}
    \]
  - If the minimum confidence threshold is 70% then only the rules with confidence 100% are kept.

Training set (at home)

- Run Apriori algorithm on the following transactional database \((\text{min}_\text{support}=2)\)

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>200</td>
<td>I2, I4</td>
</tr>
<tr>
<td>300</td>
<td>I2, I3</td>
</tr>
<tr>
<td>400</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>500</td>
<td>I1, I3</td>
</tr>
<tr>
<td>600</td>
<td>I2, I3</td>
</tr>
<tr>
<td>700</td>
<td>I1, I3</td>
</tr>
<tr>
<td>800</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>900</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>
Bottleneck of Frequent-pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1i_2...i_{100}$
    - # of scans: 100
    - $C_k$ consists of $\binom{k}{100}$ k-itemset
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + ... + \binom{1}{0} = 2^{100} - 1 = 1.27\times10^{30}$
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation? Yes 😊

Improving Apriori: general ideas

- Reduce passes of transaction database scans
- Hash-based itemset counting
  - a useful technique for accelerating the count of 2-itemset.
- Facilitate support counting of candidates
• Association rule mining
• Mining (single-dimensional) association rules in transactional databases
• **Mining Frequent Patterns Without Candidate Generation**

**Mining Frequent Patterns Without Candidate Generation**

• Compress database base representing frequent items into Frequent-Pattern tree (FP-tree)
  - compressed representation
  - avoid multiple scans of database which is costly

• FP-tree based on divide-and-conquer strategy
  - divide the compressed FP-tree into a set of conditional DBs each associated with one frequent item
  - Mine each such DB separately
Algorithm (FP-tree construction)

**Input:** A transaction database DB and a minimum support threshold  
**Output:** Its frequent pattern tree, FP-tree  
**Method:**

1. Scan the transaction database DB once.  
   Collect the set of frequent items F and their supports.  
   Sort F in support descending order as L, the list of frequent items.  
2. Create the root T of an FP-tree and label it as “empty”.  
   For each transaction Trans in DB do the following.  
   Select and sort the frequent items in Trans according to the order of L.  
   Let the item list in Trans be [p|P], where p is the first element and P the remaining.  
   Call `insert tree([p|P], T)`.  

   **function** `insert tree([p|P], T)`
   
   If T has a child N such that N.item-name= p.item-name, then  
   increment N's count by 1;  
   else  
   create a new node N, and let its count be1, its parent link be linked to T,  
   and its node-link be linked to the nodes with the same item-name via the  
   node-link structure.  
   If P is nonempty, call `insert tree(P; N)` recursively.

---

**FP-Tree: Example (1)**

<table>
<thead>
<tr>
<th>TID</th>
<th>T100</th>
<th>T200</th>
<th>T300</th>
<th>T400</th>
<th>T500</th>
<th>T600</th>
<th>T700</th>
<th>T800</th>
<th>T900</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>I1, I2, I5</td>
<td>I2, I4</td>
<td>I2, I3</td>
<td>I1, I2, I4</td>
<td>I1, I3</td>
<td>I1, I2, I3, I5</td>
<td>I1, I2, I3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted Items</td>
<td>I2, I1, I5</td>
<td>I2, I4</td>
<td>I2, I3</td>
<td>I2, I3</td>
<td>I1, I3</td>
<td>I2, I3</td>
<td>I2, I3, I5</td>
<td>I2, I1, I3</td>
<td></td>
</tr>
</tbody>
</table>

- min-support=2
- \( L = \{I2:7, I1:6, I3:6, I4:2, I5:2\} \) a sorted list on descending support count  
  *(like in Apriori we compute 1-itemset)*
- Scan database a second time. The items in each transaction are processed in  
  \( L \) order. The scan of T100 in \( L \) order leads to the following tree
FP-Tree: Example (2)

- The prefix <I2> will be shared

<table>
<thead>
<tr>
<th>TID</th>
<th>T100</th>
<th>T200</th>
<th>T300</th>
<th>T400</th>
<th>T500</th>
<th>T600</th>
<th>T700</th>
<th>T800</th>
<th>T900</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>I1, I2, I5</td>
<td>I2, I4</td>
<td>I2, I3</td>
<td>I2, I4, I5</td>
<td>I1, I3</td>
<td>I2, I3</td>
<td>I1, I3</td>
<td>I2, I3, I5, I1, I3</td>
<td></td>
</tr>
<tr>
<td>Sorted Items</td>
<td>I2, I1, I5</td>
<td>I2, I4</td>
<td>I2, I3</td>
<td>I2, I4, I5</td>
<td>I1, I3</td>
<td>I2, I3</td>
<td>I1, I3</td>
<td>I2, I3, I5, I1, I3</td>
<td></td>
</tr>
</tbody>
</table>

FP-Tree: Example (3)

FP-Tree: Example (4)

<table>
<thead>
<tr>
<th>TID</th>
<th>T100</th>
<th>T200</th>
<th>T300</th>
<th>T400</th>
<th>T500</th>
<th>T600</th>
<th>T700</th>
<th>T800</th>
<th>T900</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>I1, I2, I5</td>
<td>I2, I4</td>
<td>I2, I13</td>
<td>I1, I2, I4</td>
<td>I1, I3</td>
<td>I2, I3</td>
<td>I1, I3</td>
<td>I2, I13, I5</td>
<td>I1, I2, I3</td>
</tr>
<tr>
<td>Sorted Items</td>
<td>I2, I1, I5</td>
<td>I2, I4</td>
<td>I2, I13</td>
<td>I2, I14</td>
<td>I1, I3</td>
<td>I2, I3</td>
<td>I1, I3</td>
<td>I2, I13, I5</td>
<td>I2, I13</td>
</tr>
</tbody>
</table>

- Finally the FP_tree is, $L = \{I2:7, I1:6, I3:6, I4:2, I5:2\}$

- Exploring
  - I5 appears in 2 branches.
    - $\langle I2, I1, I5:1 \rangle$
    - $\langle I2, I1, I3, I5:1 \rangle$

  - Therefore, for suffix I5, its 2 prefix paths are:
    - $\langle I2, I1:1 \rangle$ and $\langle I2, I1, I3:1 \rangle$
    - which forms its conditional pattern base

  - Conditional FP-tree of I5 contains path $\langle I2, 2, I1:2 \rangle$
    - I3 is discarded because its support $1 < 2$

  - Frequent itemset generation
    - The path $\langle I2, 2, I1:2 \rangle$ generates all combinations of I5 with I2 and I1,
      - i.e., $\{I2, I5\}:2$, $\{I1, I5\}:2$, $\{I2, I1, I5\}:2$
We push the support 1 to every node up the chain to the root.

I3:1 is infrequent since min_support = 2.

The conditional FP-tree of I5 is \{ (I2:2, I1:2) \}.
The conditional FP-tree of I4 is \{(I2:2)\}
The conditional FP-tree of I3 is \{(I2:4,I1:2), (I1:2)\}


**Conditional FP-tree of I1**

![Conditional FP-tree of I1](image)

The conditional FP-tree of I1 is \{ (I2:4) \}

---

**Itemsets generation**

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional Pattern Base</th>
<th>Conditional FP-tree</th>
<th>Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>I5:2</td>
<td>{(I2, I1:1), (I2, I1, I3:1)}</td>
<td>{(I2:2, I1:2)}</td>
<td>{I2, I5}:2, {I1, I5}:2, {I2, I1, I5}:2</td>
</tr>
<tr>
<td>I4:2</td>
<td>{(I2, I1:1), (I2:1)}</td>
<td>{(I2:2)}</td>
<td>{I2, I4}:2</td>
</tr>
<tr>
<td>I3:6</td>
<td>{(I2, I1:2), (I2:2), (I1:2)}</td>
<td>{(I2:4, I1:2), (I1:2)}</td>
<td>{I2, I3}:4, {I1, I3}:4, {I2, I1, I3}:2</td>
</tr>
<tr>
<td>I1:6</td>
<td>{(I2:4)}</td>
<td>{(I2:4)}</td>
<td>{I2, I1}:4</td>
</tr>
<tr>
<td>I2:7</td>
<td>empty</td>
<td>empty</td>
<td>empty</td>
</tr>
</tbody>
</table>

- 1-itemset L1 = \{I2:7, I1:6, I3:6, I4:2, I5:2\}
- 2-itemset L2 = \{I2 I5:2, I1 I5:2, I2 I4:2, I2 I3:4, I1 I3:4, I2 I1:4\}
- 3-itemset L3 = \{I2 I1 I5:2, I2 I1 I3:2\}
FP-Tree vs. Apriori

Training set (At home)

- Run FP-tree algorithm on the following transactional database (min_support=3)

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