DATA MINING
LECTURE 3
Frequent Itemsets and Association Rules
This is how it all started…

- Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami: Mining Association Rules between Sets of Items in Large Databases. SIGMOD Conference 1993: 207-216

- These two papers are credited with the birth of Data Mining
- For a long time people were fascinated with Association Rules and Frequent Itemsets
  - Some people (in industry and academia) still are.
Market-Basket Data

- A large set of **items**, e.g., things sold in a supermarket.
- A large set of **baskets**, each of which is a small subset of the items, e.g., the things one customer buys on one day.

**Baskets**: 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>
Frequent itemsets

- Goal: find combinations of items (itemsets) that occur frequently
  - Called Frequent Itemsets

<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>

Support $s(I)$: number of transactions that contain itemset $I$

Examples of frequent itemsets $s(I) \geq 3$

- $\{\text{Bread}\}$: 4
- $\{\text{Milk}\}$: 4
- $\{\text{Diaper}\}$: 4
- $\{\text{Beer}\}$: 3
- $\{\text{Diaper, Beer}\}$: 3
- $\{\text{Milk, Bread}\}$: 3
Market-Baskets – (2)

• Really, a general many-to-many mapping (association) between two kinds of things, where the one (the baskets) is a set of the other (the items)
  • But we ask about connections among “items,” not “baskets.”

• The technology focuses on common/frequent events, not rare events (“long tail”).
Applications – (1)

• **Items** = products; **baskets** = sets of products someone bought in one trip to the store.

• **Example application**: given that many people buy beer and diapers together:
  • Run a sale on diapers; raise price of beer.
  • Only useful if many buy diapers & beer.
Applications – (2)

• **Baskets** = Web pages; **items** = words.

• **Example application**: Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.
Applications – (3)

- **Baskets** = sentences; **items** = documents containing those sentences.

- **Example application:** Items that appear together too often could represent plagiarism.
- Notice items do not have to be “in” baskets.
Definitions

• **Itemset**
  • A collection of one or more items
    • Example: {Milk, Bread, Diaper}
  • **k-itemset**
    • An itemset that contains $k$ items

• **Support (s)**
  • **Count**: Frequency of occurrence of an itemset
    • E.g. $s(\{\text{Milk, Bread, Diaper}\}) = 2$
  • **Fraction**: Fraction of transactions that contain an itemset
    • E.g. $s(\{\text{Milk, Bread, Diaper}\}) = 40\%$

• **Frequent Itemset**
  • An itemset $I$ whose support is greater than or equal to a $\textit{minsop}$ threshold, $s(I) \geq \text{minsop}$

<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>
Mining Frequent Itemsets task

- **Input:** Market basket data, threshold *minsup*
- **Output:** All frequent itemsets with support \( \geq \text{minsup} \)

**Problem parameters:**
- **N (size):** number of transactions
  - Walmart: billions of baskets per year
  - Web: billions of pages
- **d (dimension):** number of (distinct) items
  - Walmart sells more than 100,000 items
  - Web: billions of words
- **w:** max size of a basket
- **M:** Number of possible itemsets.
  \[ M = 2^d \]
The itemset lattice

Given $d$ items, there are $2^d$ possible itemsets.

Representation of all possible itemsets and their relationships
A Naïve Algorithm

- **Brute-force** approach: Every itemset is a candidate:
  - Consider all itemsets in the lattice, and scan the data for each candidate to compute the support
  - Time Complexity $\sim O(NMw)$, Space Complexity $\sim O(d)$

- OR
  - Scan the data, and for each transaction generate all possible itemsets. Keep a count for each itemset in the data.
  - Time Complexity $\sim O(N2^w)$, Space Complexity $\sim O(M)$

- Expensive since $M = 2^d$ !!!
  - No solution that considers all candidates is acceptable!
Computation Model

• Typically, data is kept in flat files rather than in a database system.
  • Stored on disk.
  • Stored basket-by-basket.
  • We can expand a basket into pairs, triples, etc. as we read the data.
    • Use $k$ nested loops, or recursion to generate all itemsets of size $k$.

• Data is too large to be loaded in memory.
Example file: retail

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32
33 34 35
36 37 38 39 40 41 42 43 44 45 46
38 39 47 48
38 39 48 49 50 51 52 53 54 55 56 57 58
32 41 59 60 61 62
3 39 48
63 64 65 66 67 68
32 69
48 70 71 72
39 73 74 75 76 77 78 79
36 38 39 41 48 79 80 81
82 83 84
41 85 86 87 88
39 48 89 90 91 92 93 94 95 96 97 98 99 100 101
36 38 39 48 89
39 41 102 103 104 105 106 107 108
38 39 41 109 110
39 111 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
48 134 135 136
39 48 137 138 139 140 141 142 143 144 145 146 147 148 149
39 150 151 152
38 39 56 153 154 155
```

Example: items are positive integers, and each basket corresponds to a line in the file of space-separated integers.
Computation Model – (2)

• The true cost of mining disk-resident data is usually the number of disk I/O’s.
• In practice, association-rule algorithms read the data in passes – all baskets read in turn.

• Thus, we measure the cost by the number of passes an algorithm takes.
Main-Memory Bottleneck

• For many frequent-itemset algorithms, **main memory** is the critical resource.
  • As we read baskets, we need to count something, e.g., occurrences of pairs.
  • The number of different things we can count is limited by main memory.
  • Swapping counts in/out is too slow
The Apriori Principle

- **Apriori** principle (Main observation):
  - If an itemset is frequent, then all of its subsets must also be frequent
  - If an itemset is not frequent, then all of its supersets cannot be frequent
  - The support of an itemset never exceeds the support of its subsets

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

- This is known as the anti-monotone property of support
Illustration of the Apriori principle

Figure 6.3. An illustration of the Apriori principle. If \( \{c, d, e\} \) is frequent, then all subsets of this itemset are frequent.
Illustration of the Apriori principle

Found to be Infrequent

Infrequent supersets

Pruned
The Apriori algorithm

Level-wise approach

\( C_k \) = candidate itemsets of size \( k \)
\( L_k \) = frequent itemsets of size \( k \)

1. \( k = 1, C_1 = \text{all items} \)
2. While \( C_k \) not empty

3. Scan the database to find which itemsets in \( C_k \) are frequent and put them into \( L_k \)

4. Generate the candidate itemsets \( C_{k+1} \) of size \( k+1 \) using \( L_k \)

5. \( k = k+1 \)

Candidate Generation

• Apriori principle:
  • An itemset of size \( k+1 \) is candidate to be frequent only if all of its subsets of size \( k \) are known to be frequent

Candidate generation:
• Construct a candidate of size \( k+1 \) by combining frequent itemsets of size \( k \)
  • If \( k = 1 \), take the all pairs of frequent items
  • If \( k > 1 \), join pairs of itemsets that differ by just one item
  • For each generated candidate itemset ensure that all subsets of size \( k \) are frequent.
Generate Candidates $C_{k+1}$

- Assumption: The items in an itemset are ordered
  - Integers ordered in increasing order, strings ordered lexicographically
  - The order ensures that if item $y > x$ appears before $x$, then $x$ is not in the itemset
- The itemsets in $L_k$ are also ordered

Create a candidate itemset of size $k+1$, by joining two itemsets of size $k$, that share the first $k-1$ items

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Generate Candidates $C_{k+1}$

- Assumption: The items in an itemset are ordered
  - Integers ordered in increasing order, strings ordered in lexicographically
  - The order ensures that if item $y > x$ appears before $x$, then $x$ is not in the itemset
- The itemsets in $L_k$ are also ordered

Create a candidate itemset of size $k+1$, by joining two itemsets of size $k$, that *share* the *first* $k-1$ items

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ \{1, 2, 3, 5\} \]
Generate Candidates $C_{k+1}$

- Assumption: The items in an itemset are ordered
  - Integers ordered in increasing order, strings ordered in lexicographically
  - The order ensures that if item $y > x$ appears before $x$, then $x$ is not in the itemset
- The itemsets in $L_k$ are also ordered

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Are we missing something? What about this candidate?
Generating Candidates $C_{k+1}$ in SQL

- **self-join $L_k$**

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

- \( L_3 = \{abc, abd, acd, ace, bcd\} \)
- Generating candidate set \( C_4 \)
- Self-join: \( L_3 \times L_3 \)

\[
p.item_1 = q.item_1, p.item_2 = q.item_2, p.item_3 < q.item_3
\]
Example

- $L_3\{\text{abc, abd, acd, ace, bcd}\}$
- Generating candidate set $C_4$
- Self-join: $L_3 \times L_3$

\[
p_{\text{item}_1} = q_{\text{item}_1}, p_{\text{item}_2} = q_{\text{item}_2}, p_{\text{item}_3}^{} < q_{\text{item}_3}
\]
**Example**

- \( L_3 = \{abc, abd, acd, ace, bcd\} \)
- Generating candidate set \( C_4 \)
- Self-join: \( L_3 \times L_3 \)

\[
\begin{align*}
\text{item1} & \quad \text{item2} & \quad \text{item3} \\
a & \quad b & \quad c \\
a & \quad b & \quad d \\
a & \quad c & \quad d \\
a & \quad c & \quad e \\
b & \quad c & \quad d \\
\end{align*}
\]

\[
\begin{align*}
\text{item1} & \quad \text{item2} & \quad \text{item3} \\
a & \quad b & \quad c \\
a & \quad b & \quad d \\
a & \quad c & \quad d \\
a & \quad c & \quad e \\
b & \quad c & \quad d \\
\end{align*}
\]

\[
\begin{align*}
\{a, b, c\} & & \{a, b, d\} \\
\{a, b, c, d\} & & \\
\end{align*}
\]

\[
p.item_1 = q.item_1, p.item_2 = q.item_2, \quad p.item_3 < q.item_3
\]
Example

- \( L_3 = \{abc, abd, acd, ace, bcd\} \)
- Generating candidate set \( C_4 \)
- Self-join: \( L_3 \times L_3 \)

**Example**

\[ p.item_1 = q.item_1, p.item_2 = q.item_2, p.item_3 < q.item_3 \]
**Illustration of the Apriori principle**

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

Items (1-itemsets)

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

Pairs (2-itemsets)

<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>

Only this triplet has all subsets to be frequent
But it is below the minsup threshold

Triplets (3-itemsets)

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

If every subset is considered,
\[
\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41
\]

With support-based pruning,
\[
\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13
\]
Generate Candidates $C_{k+1}$

- Are we done? Are all the candidates valid?

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Is this a valid candidate?

No. Subsets (1,3,5) and (2,3,5) should also be frequent

- Pruning step:
  - For each candidate (k+1)-itemset create all subset k-itemsets
  - Remove a candidate if it contains a subset k-itemset that is not frequent
Example

- \( L_3 = \{abc, abd, acd, ace, bcd\} \)
- **Self-joining:** \( L_3 \times L_3 \)
  - abcd from abc and abd
  - acde from acd and ace
- \( C_4 = \{abcd, acde\} \)
- **Pruning:**
  - abcd is kept since all subset itemsets are in \( L_3 \)
  - acde is removed because ade is not in \( L_3 \)
- \( C_4 = \{abcd\} \)
### Example II

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

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

- \{Bread,Diaper,Milk\}
Generate Candidates $C_{k+1}$

- We have all frequent $k$-itemsets $L_k$
- **Step 1:** self-join $L_k$
  - Create set $C_{k+1}$ by joining frequent $k$-itemsets that share the first $k-1$ items
- **Step 2:** prune
  - Remove from $C_{k+1}$ the itemsets that contain a subset $k$-itemset that is not frequent
Given the set of candidate itemsets $C_k$, we need to compute the support and find the frequent itemsets $L_k$.

Scan the data, and use a hash structure to keep a counter for each candidate itemset that appears in the data.
A simple hash structure

- Create a dictionary (hash table) that stores the candidate itemsets as keys, and the number of appearances as the value.
  - Initialize with zero
- Increment the counter for each itemset that you see in the data
Example

Suppose you have 15 candidate itemsets of length 3:

$$C_3 = \{ 
\{1 4 5\}, \{1 2 4\}, \{4 5 7\}, \{1 2 5\}, \{4 5 8\}, \\
\{1 5 9\}, \{1 3 6\}, \{2 3 4\}, \{5 6 7\}, \{3 4 5\}, \\
\{3 5 6\}, \{3 5 7\}, \{6 8 9\}, \{3 6 7\}, \{3 6 8\} 
\}$$

Hash table stores the counts of the candidate itemsets as they have been computed so far:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3 6 7}</td>
<td>0</td>
</tr>
<tr>
<td>{3 4 5}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3 6}</td>
<td>3</td>
</tr>
<tr>
<td>{1 4 5}</td>
<td>5</td>
</tr>
<tr>
<td>{2 3 4}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5 9}</td>
<td>1</td>
</tr>
<tr>
<td>{3 6 8}</td>
<td>0</td>
</tr>
<tr>
<td>{4 5 7}</td>
<td>2</td>
</tr>
<tr>
<td>{6 8 9}</td>
<td>0</td>
</tr>
<tr>
<td>{5 6 7}</td>
<td>3</td>
</tr>
<tr>
<td>{1 2 4}</td>
<td>8</td>
</tr>
<tr>
<td>{3 5 7}</td>
<td>1</td>
</tr>
<tr>
<td>{1 2 5}</td>
<td>0</td>
</tr>
<tr>
<td>{3 5 6}</td>
<td>1</td>
</tr>
<tr>
<td>{4 5 8}</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

A new tuple \{1,2,3,5,6\} generates the following itemsets of length 3:

\{1 2 3\}, \{1 2 5\}, \{1 2 6\}, \{1 3 5\}, \{1 3 6\},
\{1 5 6\}, \{2 3 5\}, \{2 3 6\}, \{3 5 6\},

Increment the counters for the itemsets in the dictionary

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3 6 7}</td>
<td>0</td>
</tr>
<tr>
<td>{3 4 5}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3 6}</td>
<td>3</td>
</tr>
<tr>
<td>{1 4 5}</td>
<td>5</td>
</tr>
<tr>
<td>{2 3 4}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5 9}</td>
<td>1</td>
</tr>
<tr>
<td>{3 6 8}</td>
<td>0</td>
</tr>
<tr>
<td>{4 5 7}</td>
<td>2</td>
</tr>
<tr>
<td>{6 8 9}</td>
<td>0</td>
</tr>
<tr>
<td>{5 6 7}</td>
<td>3</td>
</tr>
<tr>
<td>{1 2 4}</td>
<td>8</td>
</tr>
<tr>
<td>{3 5 7}</td>
<td>1</td>
</tr>
<tr>
<td>{1 2 5}</td>
<td>0</td>
</tr>
<tr>
<td>{3 5 6}</td>
<td>1</td>
</tr>
<tr>
<td>{4 5 8}</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

A new tuple \(\{1,2,3,5,6\}\) generates the following itemsets of length 3:

\[
\{1\ 2\ 3\}, \ \{1\ 2\ 5\}, \ \{1\ 2\ 6\}, \ \{1\ 3\ 5\}, \ \{1\ 3\ 6\}, \\
\{1\ 5\ 6\}, \ \{2\ 3\ 5\}, \ \{2\ 3\ 6\}, \ \{3\ 5\ 6\},
\]

Increment the counters for the itemsets in the dictionary

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3 6 7}</td>
<td>0</td>
</tr>
<tr>
<td>{3 4 5}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3 6}</td>
<td>4</td>
</tr>
<tr>
<td>{1 4 5}</td>
<td>5</td>
</tr>
<tr>
<td>{2 3 4}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5 9}</td>
<td>1</td>
</tr>
<tr>
<td>{3 6 8}</td>
<td>0</td>
</tr>
<tr>
<td>{4 5 7}</td>
<td>2</td>
</tr>
<tr>
<td>{6 8 9}</td>
<td>0</td>
</tr>
<tr>
<td>{5 6 7}</td>
<td>3</td>
</tr>
<tr>
<td>{1 2 4}</td>
<td>8</td>
</tr>
<tr>
<td>{3 5 7}</td>
<td>1</td>
</tr>
<tr>
<td>{1 2 5}</td>
<td>1</td>
</tr>
<tr>
<td>{3 5 6}</td>
<td>2</td>
</tr>
<tr>
<td>{4 5 8}</td>
<td>0</td>
</tr>
</tbody>
</table>
The frequent itemset algorithm

- **All items**
- **Count the items**
- **All pairs of items from $L_1$**
- **Count the pairs**

- **First pass**
- **Frequent items**

- **Second pass**
- **Frequent pairs**

C_1 \rightarrow Filter \rightarrow L_1 \rightarrow Construct \rightarrow C_2 \rightarrow Filter \rightarrow L_2 \rightarrow Construct \rightarrow C_3
A-Priori for All Frequent Itemsets

• One pass for each $k$.
• Needs room in main memory to count each candidate $k$-set.
• For typical market-basket data and reasonable support (e.g., 1%), $k = 2$ requires the most memory.
Picture of A-Priori

Pass 1

Item counts

Pass 2

Frequent items

Counts of pairs of frequent items
Details of Main-Memory Counting

• Two approaches:
  1. Count all pairs, using a “triangular matrix” = one dimensional array that stores the lower diagonal.
  2. Keep a table of triples \([i, j, c]\) = “the count of the pair of items \(\{i, j\}\) is \(c\)”

• (1) requires only 4 bytes/pair.
  • Note: always assume integers are 4 bytes.

• (2) requires 12 bytes/pair, but only for those pairs with count > 0.
Method (1)

4 per pair

Method (2)

12 per occurring pair
Triangular-Matrix Approach

- Number items 1, 2,…
  - Requires table of size $O(n)$ to convert item names to consecutive integers.
- Count $\{i, j\}$ only if $i < j$.
- Keep pairs in the order $\{1,2\}, \{1,3\}, \ldots, \{1,n\}, \{2,3\}, \{2,4\}, \ldots, \{2,n\}, \{3,4\}, \ldots, \{3,n\}, \ldots, \{n-1,n\}$.
- Find pair $\{i, j\}$ at the position 
  $$(i-1)(n-i/2) + j - i.$$ 
- Total number of pairs $n(n-1)/2$; total bytes about $2n^2$. 
A-Priori Using Triangular Matrix for Counts

Pass 1: Item counts

Pass 2: Frequent items, Old item #'s, Counts of pairs of frequent items
Details of Approach #2

• Total bytes used is about $12p$, where $p$ is the number of pairs that actually occur.
  • Beats triangular matrix if no more than $1/3$ of possible pairs actually occur.

• May require extra space for retrieval structure, e.g., a hash table.
ASSOCIATION RULES
Association Rule Mining

• Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

Market-Basket transactions

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

Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs, Coke},
{Beer, Bread} → {Milk},

Implication means co-occurrence, not causality!
Mining Association Rules

• Association Rule
  – An implication expression of the form \( X \rightarrow Y \), where \( X \) and \( Y \) are itemsets
  – \( \{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\} \)

• Rule Evaluation Metrics
  – Support \((s)\)
    – Fraction of transactions that contain both \( X \) and \( Y = \text{the probability } P(X,Y) \) that \( X \) and \( Y \) occur together
  – Confidence \((c)\)
    – How often \( Y \) appears in transactions that contain \( X = \text{the conditional probability } P(Y|X) \) that \( Y \) occurs given that \( X \) has occurred.

• Problem Definition
  – Input: Market-basket data, \texttt{minsup}, \texttt{minconf} values
  – Output: All rules with items in \( I \) having \( s \geq \texttt{minsup} \) and \( c \geq \texttt{minconf} \)

\[ TID \quad | \quad Items \]
\[ \begin{array}{|c|c|}
\hline
1 & \text{Bread, Milk} \\
2 & \text{Bread, Diaper, Beer, Eggs} \\
3 & \text{Milk, Diaper, Beer, Coke} \\
4 & \text{Bread, Milk, Diaper, Beer} \\
5 & \text{Bread, Milk, Diaper, Coke} \\
\hline
\end{array} \]

Example:
\( \{\text{Milk, Diaper}\} \Rightarrow \text{Beer} \)

\[ s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4 \]
\[ c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67 \]
Mining Association Rules

- Two-step approach:
  1. **Frequent Itemset Generation**
     - Generate all itemsets whose support \( \geq \text{minsup} \)
  2. **Rule Generation**
     - Generate high confidence rules from each frequent itemset, where each rule is a partitioning of a frequent itemset into Left-Hand-Side (LHS) and Right-Hand-Side (RHS)

  Frequent itemset: \{A,B,C,D\}
  
  E.g., Rule: \( AB \rightarrow CD \)

All Candidate rules:

\[
\begin{align*}
\text{BCD} & \rightarrow A, \quad \text{ACD} \rightarrow B, \quad \text{ABD} \rightarrow C, \quad \text{ABC} \rightarrow D, \\
\text{CD} & \rightarrow \text{AB}, \quad \text{BD} \rightarrow \text{AC}, \quad \text{BC} \rightarrow \text{AD}, \quad \text{AD} \rightarrow \text{BC}, \quad \text{AB} \rightarrow \text{CD}, \quad \text{AC} \rightarrow \text{BD}, \\
\text{D} & \rightarrow \text{ABC}, \quad \text{C} \rightarrow \text{ABD}, \quad \text{B} \rightarrow \text{ACD}, \quad \text{A} \rightarrow \text{BCD}
\end{align*}
\]
Association Rule anti-monotonicity

- In general, confidence does not have an anti-monotone property with respect to the size of the itemset:
  \[ c(ABC \rightarrow D) \text{ can be larger or smaller than } c(AB \rightarrow D) \]

- But confidence is anti-monotone w.r.t. number of items on the RHS of the rule (or monotone with respect to the LHS of the rule)

- e.g., \( L = \{A,B,C,D\} \):
  \[ c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD) \]
Rule Generation for Apriori Algorithm

Lattice of rules created by the RHS

Low Confidence Rule

Pruned Rules
Rule Generation for A Priori Algorithm

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

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

- Prune rule \( D \rightarrow ABC \) if its subset \( AD \rightarrow BC \) does not have high confidence.

- Essentially we are doing A Priori on the RHS.
RESULT
POST-PROCESSING
Compact Representation of Frequent Itemsets

- Some itemsets are redundant because they have identical support as their supersets

| TID | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | B10 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 |
|-----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|
| 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 2   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 3   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 4   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 5   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 6   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 7   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 8   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 9   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 10  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 11  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 12  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 13  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 14  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 15  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

- Number of frequent itemsets $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$
- Need a compact representation
Maximal Frequent Itemsets

An itemset is **maximal** frequent if none of its immediate **supersets** is frequent

Maximal Itemsets

Infrequent Itemsets

Maximal: no superset has this property

Maximal itemsets = positive border
Negative Border

Itemsets that are not frequent, but all their immediate subsets are frequent.

**Minimal**: no subset has this property
Border

• **Border** = Positive Border + Negative Border
  • Itemsets such that all their immediate subsets are frequent and all their immediate supersets are infrequent.

• Either the positive, or the negative border is sufficient to **summarize** all frequent itemsets.
Closed Itemsets

- An itemset is closed if none of its immediate supersets has the same support as the itemset.

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{A,B}</td>
<td>{A}</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>{B,C,D}</td>
<td>{B}</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>{A,B,C,D}</td>
<td>{C}</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>{A,B,D}</td>
<td>{D}</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>{A,B,C,D}</td>
<td>{A,B}</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A,C}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A,D}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{B,C}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{B,D}</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{C,D}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A,B,C}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A,B,D}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A,C,D}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{B,C,D}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{A,B,C,D}</td>
<td>2</td>
</tr>
</tbody>
</table>
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>BCE</td>
</tr>
<tr>
<td>4</td>
<td>ACDE</td>
</tr>
<tr>
<td>5</td>
<td>DE</td>
</tr>
</tbody>
</table>

Not supported by any transactions
Maximal vs Closed Frequent Itemsets

Minimum support = 2

Closed but not maximal

Closed and maximal

# Closed = 9
# Maximal = 4
Maximal vs Closed Itemsets
Pattern Evaluation

• Association rule algorithms tend to produce too many rules but many of them are uninteresting or redundant
  • Redundant if \{A, B, C\} \rightarrow \{D\} and \{A, B\} \rightarrow \{D\} have same support & confidence
    • Summarization techniques
  • Uninteresting, if the pattern that is revealed does not offer useful information.
    • Interestingness measures: a hard problem to define

• Interestingness measures can be used to prune/rank the derived patterns
  • Subjective measures: require human analyst
  • Objective measures: rely on the data.

• In the original formulation of association rules, support & confidence are the only measures used
Computing Interestingness Measure

- Given a rule $X \rightarrow Y$, information needed to compute rule interestingness can be obtained from a contingency table.

Contingency table for $X \rightarrow Y$

<table>
<thead>
<tr>
<th></th>
<th>$Y$</th>
<th>$\bar{Y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$f_{11}$</td>
<td>$f_{10}$</td>
</tr>
<tr>
<td>$\bar{X}$</td>
<td>$f_{01}$</td>
<td>$f_{00}$</td>
</tr>
<tr>
<td></td>
<td>$f_{+1}$</td>
<td>$f_{+0}$</td>
</tr>
</tbody>
</table>

- $f_{11}$: support of $X$ and $Y$
- $f_{10}$: support of $X$ and $\bar{Y}$
- $f_{01}$: support of $\bar{X}$ and $Y$
- $f_{00}$: support of $\bar{X}$ and $\bar{Y}$

$X$: itemset $X$ appears in tuple
$Y$: itemset $Y$ appears in tuple
$\bar{X}$: itemset $X$ does not appear in tuple
$\bar{Y}$: itemset $Y$ does not appear in tuple

Used to define various measures:
- support, confidence, lift, Gini, J-measure, etc.
**Drawback of Confidence**

<table>
<thead>
<tr>
<th></th>
<th>Coffee</th>
<th>Coffee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tea</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Tea</td>
<td>75</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>10</td>
</tr>
</tbody>
</table>

**Association Rule:** Tea → Coffee

Confidence = \( P(\text{Coffee}|\text{Tea}) = \frac{15}{20} = 0.75 \)

Although confidence is high, rule is misleading

- \( P(\text{Coffee}) = \frac{90}{100} = 0.9 \)
- \( P(\text{Coffee}|\overline{\text{Tea}}) = 0.9375 \)
Statistical Independence

• Population of 1000 students
  • 600 students know how to swim (S)
  • 700 students know how to bike (B)
  • 420 students know how to swim and bike (S,B)

  • \( P(S,B) = \frac{420}{1000} = 0.42 \)
  • \( P(S) \times P(B) = 0.6 \times 0.7 = 0.42 \)

  • \( P(S,B) = P(S) \times P(B) \Rightarrow \text{Statistical independence} \)
Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 500 students know how to swim and bike (S,B)

- \( P(S,B) = \frac{500}{1000} = 0.5 \)
- \( P(S) \times P(B) = 0.6 \times 0.7 = 0.42 \)

- \( P(S,B) > P(S) \times P(B) \Rightarrow \) Positively correlated
Statistical Independence

- Population of 1000 students
  - 600 students know how to swim \((S)\)
  - 700 students know how to bike \((B)\)
  - 300 students know how to swim and bike \((S, B)\)

- \(P(S, B) = \frac{300}{1000} = 0.3\)
- \(P(S) \times P(B) = 0.6 \times 0.7 = 0.42\)

- \(P(S, B) < P(S) \times P(B) \Rightarrow\) Negatively correlated
Statistical-based Measures

- Measures that take into account statistical dependence
  - Lift/Interest/PMI
    \[
    \text{Lift} = \frac{P(Y|X)}{P(Y)} = \frac{P(X,Y)}{P(X)P(Y)} = \text{Interest}
    \]
    In text mining it is called: Pointwise Mutual Information
  - Piatesky-Shapiro
    \[
    \text{PS} = P(X,Y) - P(X)P(Y)
    \]
- All these measures measure deviation from independence
  - The higher, the better (why?)
Example: Lift/Interest

<table>
<thead>
<tr>
<th></th>
<th>Coffee</th>
<th>Coffee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tea</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Tea</td>
<td>75</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

Association Rule: Tea → Coffee

Confidence = P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

⇒ Lift = 0.75/0.9 = 0.8333 (< 1, therefore is negatively associated)

= 0.15/(0.9*0.2)
### Another Example

<table>
<thead>
<tr>
<th>Fraction of documents</th>
<th>of</th>
<th>the</th>
<th>of, the</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.9</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

If I was creating a document by picking words randomly, *(of, the)* have more or less the same probability of appearing together by chance. Therefore, **No correlation**

<table>
<thead>
<tr>
<th>Fraction of documents</th>
<th>hong</th>
<th>kong</th>
<th>hong, kong</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
<td>0.2</td>
<td>0.19</td>
</tr>
</tbody>
</table>

*(hong, kong)* have much lower probability to appear together by chance. The two words appear almost always only together. Therefore, **Positive correlation**

<table>
<thead>
<tr>
<th>Fraction of documents</th>
<th>obama</th>
<th>karagounis</th>
<th>obama, karagounis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
<td>0.2</td>
<td>0.001</td>
</tr>
</tbody>
</table>

*(obama, karagounis)* have much higher probability to appear together by chance. The two words appear almost never together. Therefore, **Negative correlation**
Drawbacks of Lift/Interest/Mutual Information

<table>
<thead>
<tr>
<th></th>
<th>honk</th>
<th>konk</th>
<th>honk, konk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction of documents</td>
<td>0.0001</td>
<td>0.0001</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

\[
MI(honk, konk) = \frac{0.0001}{0.0001 \times 0.0001} = 10000
\]

<table>
<thead>
<tr>
<th></th>
<th>hong</th>
<th>kong</th>
<th>hong, kong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction of documents</td>
<td>0.2</td>
<td>0.2</td>
<td>0.19</td>
</tr>
</tbody>
</table>

\[
MI(hong, kong) = \frac{0.19}{0.2 \times 0.2} = 4.75
\]

Rare co-occurrences are deemed more interesting. But this is not always what we want.
ALTERNATIVE FREQUENT ITEMSET COMPUTATION

Slides taken from Mining Massive Datasets course by Anand Rajaraman and Jeff Ullman.
Finding the frequent pairs is usually the most expensive operation.
Picture of A-Priori

- Item counts
- Frequent items
  - Counts of pairs of frequent items

Pass 1

Pass 2
PCY Algorithm

- During Pass 1 (computing frequent items) of Apriori, most memory is idle.

- Use that memory to keep a hash table where pairs of items are hashed.

- The hash table keeps just counts of the number of pairs hashed in each bucket, not the pairs themselves.
Needed Extensions

1. Pairs of items need to be generated from the input file; they are not present in the file.

2. Memory organization:
   • Space to count each item.
     • One (typically) 4-byte integer per item.
   • Use the rest of the space for as many integers, representing buckets, as we can.
Picture of PCY

Pass 1

Item counts

Hash table
Picture of PCY

Item counts

Pass 1

Bucket Counts
PCY Algorithm – Pass 1

FOR (each basket) {
    FOR (each item in the basket)
        add 1 to item’s count;
    FOR (each pair of items in the basket)
    {
        hash the pair to a bucket;
        add 1 to the count for that bucket
    }
}
Observations About Buckets

- A bucket is frequent if its count is at least the support threshold.

- A bucket that a frequent pair hashes to is surely frequent.
  - We cannot use the hash table to eliminate any member of this bucket.

- Even without any frequent pair, a bucket can be frequent.
  - Again, nothing in the bucket can be eliminated.

- But in the best case, the count for a bucket is less than the support $s$.
  - Now, all pairs that hash to this bucket can be eliminated as candidates, even if the pair consists of two frequent items.

- On Pass 2 (frequent pairs), we only count pairs that hash to frequent buckets.
PCY Algorithm – Between Passes

- Replace the buckets by a bit-vector:
  - 1 means the bucket is frequent; 0 means it is not.

- 4-byte integers are replaced by bits, so the bit-vector requires $1/32$ of memory.

- Also, find which items are frequent and list them for the second pass.
  - Same as with Apriori
Picture of PCY

- Item counts
- Hash table
  - Pass 1

- Frequent items
- Bitmap
  - Counts of candidate pairs
  - Pass 2
PCY Algorithm – Pass 2

- Count all pairs \( \{i, j\} \) that meet the conditions for being a candidate pair:
  1. Both \( i \) and \( j \) are frequent items.
  2. The pair \( \{i, j\} \), hashes to a bucket number whose bit in the bit vector is 1.

- Notice both these conditions are necessary for the pair to have a chance of being frequent.
All (Or Most) Frequent Itemsets in less than 2 Passes

- A-Priori, PCY, etc., take $k$ passes to find frequent itemsets of size $k$.
- Other techniques use 2 or fewer passes for all sizes:
  - Simple sampling algorithm.
  - SON (Savasere, Omiecinski, and Navathe).
  - Toivonen.
Simple Sampling Algorithm – (1)

- Take a random sample of the market baskets.

- Run Apriori or one of its improvements (for sets of all sizes, not just pairs) in main memory, so you don’t pay for disk I/O each time you increase the size of itemsets.
  - Make sure the sample is such that there is enough space for counts.
Main-Memory Picture

- Copy of sample baskets
- Space for counts
Simple Algorithm – (2)

- Use as your support threshold a suitable, scaled-back number.
  - E.g., if your sample is $1/100$ of the baskets, use $s/100$ as your support threshold instead of $s$.

- You could stop here (single pass)
  - What could be the problem?
Simple Algorithm – Option

• Optionally, verify that your guesses are truly frequent in the entire data set by a second pass (eliminate false positives)

• But you don’t catch sets frequent in the whole but not in the sample. (false negatives)
  • Smaller threshold, e.g., $s/125$, helps catch more truly frequent itemsets.
    • But requires more space.
SON Algorithm – (1)

• **First pass**: Break the data into **chunks** that can be processed in main memory.

• Read one chunk at the time
  • Find all frequent itemsets for each chunk.
  • Threshold = \( s / \text{number of chunks} \)

• An itemset becomes a **candidate** if it is found to be frequent in **any** one or more chunks of the baskets.
SON Algorithm – (2)

• **Second pass**: count all the candidate itemsets and determine which are frequent in the entire set.

• **Key “monotonicity” idea**: an itemset cannot be frequent in the entire set of baskets unless it is frequent in at least one subset.
  • Why?
SON Algorithm – Distributed Version

• This idea lends itself to distributed data mining.
• If baskets are distributed among many nodes, compute frequent itemsets at each node, then distribute the candidates from each node.
• Finally, accumulate the counts of all candidates.
Toivonen’s Algorithm – (1)

- Start as in the simple sampling algorithm, but lower the threshold slightly for the sample.
  - **Example**: if the sample is 1% of the baskets, use $s/125$ as the support threshold rather than $s/100$.
  - Goal is to avoid missing any itemset that is frequent in the full set of baskets.
Toivonen’s Algorithm – (2)

• Add to the itemsets that are frequent in the sample the **negative border** of these itemsets.
• An itemset is in the negative border if it is **not** deemed frequent in the sample, but **all** its **immediate subsets** are.
Reminder: Negative Border

- Itemset $ABCD$ is in the negative border if and only if:
  1. It is not frequent in the sample, but
  2. All of $ABC$, $BCD$, $ACD$, and $ABD$ are.

- Item $A$ is in the negative border if and only if it is not frequent in the sample.
  - Because the empty set is always frequent.
    - Unless there are fewer baskets than the support threshold (silly case).
Picture of Negative Border

... triples pairs singletons

Negative Border

Frequent Itemsets from Sample
Toivonen’s Algorithm – (3)

• In a second pass, compute the support for all candidate frequent itemsets from the first pass, and also for their negative border.

• If no itemset from the negative border turns out to be frequent, then the candidates found to be frequent in the whole data are exactly the frequent itemsets.
Toivonen’s Algorithm – (4)

- What if we find that something in the negative border is actually frequent?
  - We must start over again!

- Try to choose the support threshold so the probability of failure is low, while the number of itemsets checked on the second pass fits in main-memory.
If Something in the Negative Border is Frequent . . .

We broke through the negative border. How far does the problem go?

... tripletons

doubletons

singletons

Frequent Itemsets from Sample
Theorem:

- If there is an itemset that is frequent in the whole, but not frequent in the sample, then there is a member of the negative border for the sample that is frequent in the whole.
Proof: Suppose not; i.e.;

1. There is an itemset $S$ frequent in the whole but not frequent in the sample, and
2. Nothing in the negative border is frequent in the whole.

- Let $T$ be a smallest subset of $S$ that is not frequent in the sample.
- $T$ is frequent in the whole ($S$ is frequent + monotonicity).
- $T$ is in the negative border (else not “smallest”).
Example
FREQUENT ITEMSET RESEARCH
Figure 6.31. A summary of the various research activities in association analysis.