"Association Rules"

Market Baskets
Frequent Itemsets
A-priori Algorithm

The Market-Basket Model

◆ A large set of \textit{items}, e.g., things sold in a supermarket.

◆ A large set of \textit{baskets}, each of which is a small set of the items, e.g., the things one customer buys on one day.

The Market-Basket Model

Support

◆ Simplest question: find sets of items that appear “frequently” in the baskets.

◆ \textit{Support} for itemset \( I \) = the number of baskets containing all items in \( I \).

◆ Given a support \textit{threshold} \( s \), sets of items that appear in \( \geq s \) baskets are called \textit{frequent itemsets}.

Support

Example

◆ \textit{Items}=\{milk, coke, pepsi, beer, juice\}.

◆ \textit{Support} = 3 baskets.

\begin{align*}
B_1 &= \{m, c, b\} & B_2 &= \{m, p, j\} \\
B_3 &= \{m, b\} & B_4 &= \{c, j\} \\
B_5 &= \{m, p, b\} & B_6 &= \{m, c, b, j\} \\
B_7 &= \{c, b, j\} & B_8 &= \{b, c\}
\end{align*}

◆ Frequent itemsets: \{m\}, \{c\}, \{b\}, \{j\}, \{m, b\}, \{c, b\}, \{j, c\}.

Applications --- (1)

◆ \textit{Real market baskets}: chain stores keep terabytes of information about what customers buy together.

◆ Tells how typical customers navigate stores, lets them position tempting items.

◆ Suggests tie-in “tricks,” e.g., run sale on diapers and raise the price of beer.

◆ High support needed, or no \$’s.

Applications --- (2)

◆ \textit{“Baskets”} = documents; \textit{“items”} = words in those documents.

◆ Lets us find words that appear together unusually frequently, i.e., linked concepts.

◆ \textit{“Baskets”} = sentences, \textit{“items”} = documents containing those sentences.

◆ Items that appear together too often could represent plagiarism.
Applications --- (3)

- "Baskets" = Web pages; "items" = linked pages.
  - Pairs of pages with many common references may be about the same topic.
- "Baskets" = Web pages $p$; "items" = pages that link to $p$.
  - Pages with many of the same links may be mirrors or about the same topic.

Important Point

- "Market Baskets" is an abstraction that models any many-many relationship between two concepts: "items" and "baskets."
  - Items need not be "contained" in baskets.
- The only difference is that we count co-occurrences of items related to a basket, not vice-versa.

Scale of Problem

- WalMart sells 100,000 items and can store billions of baskets.
- The Web has over 100,000,000 words and billions of pages.

Association Rules

- If-then rules about the contents of baskets.
- $\{i_1, i_2, \ldots, i_k\} \rightarrow j$ means: "if a basket contains all of $i_1, \ldots, i_k$ then it is likely to contain $j$.
- Confidence of this association rule is the probability of $j$ given $i_1, \ldots, i_k$.

Example

+ $B_1 = \{m, c, b\}$
+ $B_2 = \{m, p, j\}$
+ $B_3 = \{m, b\}$
+ $B_4 = \{c, j\}$
- $B_5 = \{m, p, b\}$
+ $B_6 = \{m, c, b, j\}$
- $B_7 = \{c, b, j\}$
+ $B_8 = \{b, c\}$

An association rule: $\{m, b\} \rightarrow c$.
  - Confidence = $2/4 = 50\%$.

Interest

- The interest of an association rule $X \rightarrow Y$ is the absolute value of the amount by which the confidence differs from the probability of $Y$. 

Example

\[ B_1 = \{\text{m, c, b}\} \quad B_2 = \{\text{m, p, j}\} \]
\[ B_3 = \{\text{m, b}\} \quad B_4 = \{\text{c, j}\} \]
\[ B_5 = \{\text{m, p, b}\} \quad B_6 = \{\text{m, c, b, j}\} \]
\[ B_7 = \{\text{c, b, j}\} \quad B_8 = \{\text{b, c}\} \]

- For association rule \( \{\text{m, b}\} \rightarrow \text{c} \), item c appears in 5/8 of the baskets.
- Interest = \( |2/4 - 5/8| = 1/8 \) --- not very interesting.

Relationships Among Measures

- Rules with high support and confidence may be useful even if they are not “interesting.”
  - We don’t care if buying bread *causes* people to buy milk, or whether simply a lot of people buy both bread and milk.
  - But high interest suggests a cause that might be worth investigating.

Finding Association Rules

- A typical question: “find all association rules with support \( \geq s \) and confidence \( \geq c \).”
  - Note: “support” of an association rule is the support of the set of items it mentions.
- Hard part: finding the high-support (frequent) itemsets.
  - Checking the confidence of association rules involving those sets is relatively easy.

Computation Model

- Typically, data is kept in a “flat file” rather than a database system.
  - Stored on disk.
  - Stored basket-by-basket.
  - Expand baskets into pairs, triples, etc. as you read baskets.

File Organization

```
  Basket 1
  Basket 2
  Basket 3
  Etc.
```

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 a disaster.

Finding Frequent Pairs

- The hardest problem often turns out to be finding the frequent pairs.
- We’ll concentrate on how to do that, then discuss extensions to finding frequent triples, etc.

Naïve Algorithm

- Read file once, counting in main memory the occurrences of each pair.
  - Expand each basket of $n$ items into its $n \cdot (n - 1)/2$ pairs.
- Fails if $n^2$ exceeds main memory.
  - Remember: #items can be 100K (Wal-Mart) or 10B (Web pages).

Details of Main-Memory Counting

- Two approaches:
  1. Count all item pairs, using a triangular matrix.
  2. Keep a table of triples $[i, j, c] = \text{the count of the pair of items } \{i, j\} \text{ is } c$.
- (1) requires only (say) 4 bytes/pair.
- (2) requires 12 bytes, but only for those pairs with count $> 0$.

Details of Approach #1

- Number items 1, 2, ...
- 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 $n \cdot (n-1)/2$; total bytes about $2n^2$. 
Details of Approach #2

- You need a hash table, with $i$ and $j$ as the key, to locate $(i, j, c)$ triples efficiently.
  - Typically, the cost of the hash structure can be neglected.
- Total bytes used is about $12p$, where $p$ is the number of pairs that actually occur.
  - Beats triangular matrix if at most 1/3 of possible pairs actually occur.

A-Priori Algorithm --- (1)

- A two-pass approach called a-priori limits the need for main memory.
- Key idea: monotonicity: if a set of items appears at least $s$ times, so does every subset.
  - Contrapositive for pairs: if item $i$ does not appear in $s$ baskets, then no pair including $i$ can appear in $s$ baskets.

A-Priori Algorithm --- (2)

- **Pass 1**: Read baskets and count in main memory the occurrences of each item.
  - Requires only memory proportional to #items.
- **Pass 2**: Read baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent.
  - Requires memory proportional to square of frequent items only.

Picture of A-Priori

![Picture of A-Priori]

Detail for A-Priori

- You can use the triangular matrix method with $n =$ number of frequent items.
  - Saves space compared with storing triples.
- Trick: number frequent items 1, 2, ..., and keep a table relating new numbers to original item numbers.

Frequent Triples, Etc.

- For each $k$, we construct two sets of $k$-tuples:
  - $C_k =$ candidate $k$-tuples = those that might be frequent sets (support $\geq s$) based on information from the pass for $k-1$.
  - $L_k =$ the set of truly frequent $k$-tuples.
A-Priori for All Frequent Itemsets

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

Frequent Itemsets --- (2)

- $C_1 =$ all items
- $L_1 =$ those counted on first pass to be frequent.
- $C_2 =$ pairs, both chosen from $L_1$.
- In general, $C_k =$ $k$-tuples, each $k-1$ of which is in $L_{k-1}$.
- $L_k =$ members of $C_k$ with support $\geq s$. 