Association Rules

Juliana Freire

Modified from Jeff Ullman, Jure Lescovek, Bing Liu, Jiawei Han
Association Rules: Some History

- Bar code technology allowed retailers to collect massive volumes of sales data
- Basket data: transaction date, set of items bought
- Leverage information for marketing
  - How to design coupons?
  - How to organize shelves?
- Data is very large and stored in tertiary storage
- Current (as of 1993) “database systems do not provide necessary functionality for a user interested in taking advantage of this information”
Association Rules: Some History

• The birth of data mining!

• Agrawal et al. (SIGMOD 1993) introduced the problem
  • Mining a large collection of basket data to discover association rules

• Many papers followed…
Association Rules: Some History

• Any feeling of deja vu?

• GPS technology, massive volumes of Web data, user crowds, has allowed <government, companies, …> to collect massive volumes of data

• Leverage information for marketing, to improve citizens’ lives, etc…

• The birth of Big data!
Association Rules: Impact

Scholar

About 1,700,000 results (0.06 sec)

Articles

Fast algorithms for mining association rules
R Agrawal, R Srikant - Proc. 20th Int. Conf. Very Large Data ..., 1994 - www-cgi.cs.cmu.edu
This is a very long and complicated paper about taking a set of transactions (what the paper calls basket data) and finding association rules in them. For example, a marketing firm might want to ask "What percentage of people who bought X also bought Y?" Another question ...
Cited by 13400 Related articles BL Direct All 312 versions Cite More

Mining association rules between sets of items in large databases
R Agrawal, T Imieliński, A Swami - ACM SIGMOD Record, 1993 - dl.acm.org
Abstract We are given a large database of customer transactions. Each transaction consists of items purchased by a customer in a visit. We present an efficient algorithm that generates all significant association rules between items in the database. The algorithm incorporates ...
Cited by 11894 Related articles BL Direct All 99 versions Cite

Fast discovery of association rules
R Agrawal, H Mannila, R Srikant, H Toivonen... - ... discovery and data ..., 1996 - cs.helsinki.fi
Abstract Association rules are statements of the form "98% of customers that purchase tires and automobile accessories also get automotive services." We consider the problem of discovering association rules between items in large databases. We present two new ...
Cited by 2458 Related articles All 9 versions Cite

Mining quantitative association rules in large relational tables
R Srikant, R Agrawal - ACM SIGMOD Record, 1996 - dl.acm.org
Abstract We introduce the problem of mining association rules in large relational tables
# Rakesh Agrawal

Technical Fellow, Microsoft Research

**Data Mining - Web Search - Education - Privacy**

Verified email at microsoft.com

## Citation Indices

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>Since 2007</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citations</td>
<td>67728</td>
<td>33861</td>
</tr>
<tr>
<td>h-index</td>
<td>81</td>
<td>57</td>
</tr>
<tr>
<td>i10-index</td>
<td>186</td>
<td>144</td>
</tr>
</tbody>
</table>

## Citations to My Articles

<table>
<thead>
<tr>
<th>Title / Author</th>
<th>Cited by</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast algorithms for mining association rules</td>
<td>13400</td>
<td>1994</td>
</tr>
<tr>
<td>R Agrawal, R Srikant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proc. 20th Int. Conf. Very Large Data Bases, VLDB 1215, 487-499</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mining association rules between sets of items in large databases</td>
<td>11894</td>
<td>1993</td>
</tr>
<tr>
<td>R Agrawal, T Imieliński, A Swami</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACM SIGMOD Record 22 (2), 207-216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mining sequential patterns</td>
<td>4244</td>
<td>1995</td>
</tr>
<tr>
<td>R Agrawal, R Srikant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Engineering, 1995. Proceedings of the Eleventh International Conference ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast discovery of association rules</td>
<td>2458</td>
<td>1996</td>
</tr>
<tr>
<td>R Agrawal, H Mannila, R Srikant, H Toivonen, Al Verkamo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advances in knowledge discovery and data mining 12, 307-328</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic subspace clustering of high dimensional data for data mining applications</td>
<td>2090</td>
<td>1998</td>
</tr>
<tr>
<td>R Agrawal, J Gehrke, D Gunopulos, P Raghavan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACM SIGMOD Record 27 (2), 94-105</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Association Rule Discovery

Supermarket shelf management:

• **Goal:** Identify items that are bought together by sufficiently many customers – the *frequent itemsets*
  - Items that *co-occur more frequently than would be expected were the items bought independently*
  - Bread + milk is not surprising…
  - Hot dogs + mustard is not surprising either, but supermarkets can do clever marketing: hot dogs on sale and increase the price of mustard

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Coke, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Beer, Bread</td>
</tr>
<tr>
<td>3</td>
<td>Beer, Coke, Diaper, Milk</td>
</tr>
<tr>
<td>4</td>
<td>Beer, Bread, Diaper, Milk</td>
</tr>
<tr>
<td>5</td>
<td>Coke, Diaper, Milk</td>
</tr>
</tbody>
</table>
Supermarket shelf management:
• **Goal**: Identify items that are bought together by sufficiently many customers – the *frequent itemsets*
• **Approach**: Process the sales data collected with barcode scanners to *find dependencies among items*
• **A classic rule**:  
  • If one buys diaper and milk, then she is likely to buy beer  
  • Don’t be surprised if you find six-packs next to diapers!

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

**Rules Discovered:**
{Milk} --> {Coke}
{Diaper, Milk} --> {Beer}
The Market-Basket Model

- A large set of *items*, e.g., things
  - $I = \{i_1, i_2, \ldots, i_m\}$
- A much larger set of *baskets/transactions*, e.g., the things one customer buys on one day
  - $t$ a set of items, and $t \subseteq I$.

- Transaction Database $T$: a set of transactions $T = \{t_1, t_2, \ldots, t_n\}$.

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

An average Walmart stores sells 120,000 items; Walmart processes 1M transaction every hour.

Market-Baskets and Associations

• A many-many mapping (association) between two kinds of objects.
  • Identify *connections among “items,”* not “baskets.”
  • E.g., 90% of transactions that purchase bread and butter also purchase milk

• The technology focuses on *common events,* not rare events ( “long tail” )
  • *Why is this required for bricks and mortar but not for online retailers?*
Applications – Market Baskets

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

- **Real market baskets**: Chain stores keep TBs of data about what customers buy together.

- Tells how typical customers navigate stores, lets them position tempting items
  - Place items close to each other, e.g., printer + laptop
  - Place items far apart – customer is likely to find/buy something interesting on the way

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

- High **support** needed, or no $$’s
  - Only useful if many buy diapers & beer.

- Amazon’s **people who bought X also bought Y**
Applications – Plagiarism

- **Baskets** = sentences; **items** = documents containing those sentences.
  - Items that appear together too often could represent plagiarism.
    - “I love NYC” – \{d1, d3, d5\}
    - “The subway is slow” – \{d1, d3\}
  - *Notice items do not have to be “in” baskets.*
Applications – Drugs/Side Effects

- **Baskets** = patients; **Items** = drugs & side-effects
  - Has been used to detect combinations of drugs that result in particular side-effects
  - **But requires extension:** Absence of an item needs to be observed as well as presence
Applications – Related Concepts

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

- Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.
Association Rules: Approach

• Given a set of baskets, discover association rules
  • People who bought \{a, b, c\} tend to buy \{d, e\}

• 2-step approach
  • Find frequent \textit{itemsets}
  • Generate \textit{association rules}
Why Is Frequent Pattern Mining Important?

- Finding inherent regularities in data: Discloses an intrinsic and important property of data sets
- Forms the foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Cluster analysis: frequent pattern-based clustering
Scale of the Problem

- WalMart sells 120,000 items and can store billions of baskets.
- The Web has billions of words and many billions of pages.
- A big data problem!
Outline

• Define:
  • Frequent itemsets
  • Association rules: confidence, support, interestingness

• Algorithms for finding frequent itemsets
Frequent Itemsets

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

• **Support** for itemset \( I \) = the number of baskets containing all items in \( I \).
  • Often expressed as a fraction of the total number of baskets

• Given a support threshold \( s \), sets of items that appear in at least \( s \) baskets are called **frequent itemsets**.

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

Support of \{Beer, Bread\} = 2

Support of \{Coke, Milk\} = ?
Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.
  - \( B_1 = \{m, c, b\} \)
  - \( B_2 = \{m, p, j\} \)
  - \( B_3 = \{m, b\} \)
  - \( B_4 = \{c, j\} \)
  - \( B_5 = \{m, p, b\} \)
  - \( B_6 = \{m, c, b, j\} \)
  - \( B_7 = \{c, b, j\} \)
  - \( B_8 = \{b, c\} \)
- Frequent itemsets: \{m\}-5, \{c\}-5, \{b\}-4, \{j\}-4, \{p\}-2
- How many doubletons can be frequent?
  - \{m,b\}, \{b,c\}, \{c,j\}
The Market-Basket Model

- A transaction $t$ contains $X$, a set of items (itemset) in $I$, if $X \subseteq t$.

- An itemset is a set of items.
  - E.g., $X = \{milk, bread, cereal\}$ is an itemset.

- A $k$-itemset is an itemset with $k$ items.
  - E.g., $\{milk, bread, cereal\}$ is a 3-itemset

- An association rule is an implication of the form: $X \rightarrow Y$, where $X, Y \subseteq I$, and $X \cap Y = \emptyset$
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 $i_1, \ldots, i_k$.

\[
\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}
\]
Example: Confidence

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

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

\[
\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}
\]
Interesting Association Rules

- Not all high-confidence rules are interesting
  - The rule $X \rightarrow \text{Milk}$ may have high confidence for many itemsets $X$, because milk is purchased very often (independent of $X$) and the confidence will be very high

- **Interest** of an association rule $I \rightarrow j$ is the difference between its confidence and the fraction of baskets that contain $j$

  \[
  \text{Interest} (I \rightarrow j) = \text{conf}(I \rightarrow j) - \Pr[j]
  \]

- Interesting rules are those with high positive or negative interest values
- For uninteresting rules, the fraction of baskets containing $j$ will be the same as the fraction of the subset baskets including $\{I,j\}$. So confidence will be high, but interest low
Example: Confidence and Interest

- An association rule: \( \{m, b\} \rightarrow c \).
  - Confidence = \( \frac{2}{4} = 50\% \).
  - Interest (\( \{m,b\} \rightarrow c \)) = Confidence (\( \{m,b\} \rightarrow c \)) − Pr[c] = 10.5 − 5/8 = 1/8 --- not very interesting…
  - Item c appears in 5/8 of the baskets
Example: Confidence and Interest

• \{\text{diapers}\} \rightarrow \text{beer}
  • The fraction of diaper-buyers that buy beer is significantly greater than the fraction of all customers that buy beer
    \[
    \text{interest}(\{\text{diapers}\} \rightarrow \text{beer}) = \frac{\text{support}(\{\text{diapers,beer}\})}{\text{support}(\{\text{diapers}\})} - \frac{\text{support}(\{\text{beer}\})}{\text{num\_baskets}}
    \]

• \{\text{coke}\} \rightarrow \text{pepsi}
  • Negative interest – people who buy coke are unlikely to also buy pepsi
    \[
    \text{interest}(\{\text{coke}\} \rightarrow \text{pepsi}) = \frac{\text{support}(\{\text{coke,pepsi}\})}{\text{support}(\{\text{coke}\})} - \frac{\text{support}(\{\text{pepsi}\})}{\text{num\_baskets}}
    \]
Finding Association Rules

- **Goal:** Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).
  - Support $\geq s$ and confidence $\geq c$

- **For brick-and-mortar marketing:**
  - *support of 1% is reasonably high*
  - *confidence of 50% is adequate, otherwise rule has little practical effect*

- **Key Features**
  - Completeness: find all rules.
  - Mining with data on hard disk (not in memory)
Mining Association Rules

- **Two steps:**
  1) Find all itemsets \( I \) that have minimum support (frequent itemsets, also called large itemsets).
  2) Rule generation: Use frequent itemsets to generate rules.
     - For every subset \( A \) of \( I \), generate rule \( A \rightarrow I-A \)
       - If \( I \) is frequent, then so is \( A \)
       - Perform a single pass to compute the rule confidence
         - \( \text{Conf}(A,B \rightarrow C,D) = \frac{\text{supp}(A,B,C,D)}{\text{supp}(A,B)} \)
         - Can generate bigger rules from smaller ones
     - Output rules above confidence threshold

The hard part!
Example

Min support $s = 3$, confidence $= 0.75$

1) Frequent itemsets

\{m\} – 5; \{c\} – 6; \{b\} – 6; \{n\} – 1; \{p\} – 2; \{j\} – 4
\{m,c\} – 3; \{m,b\} – 4; \{m,n\} – 1; …; \{b,c\} – 5; \{c,j\} – 3; \{m,c,b\} – 3

2) Generate rules

\begin{align*}
m \rightarrow c: & \quad c = \frac{3}{5} \\
b \rightarrow c: & \quad c = \frac{5}{6} \\
b,c \rightarrow m: & \quad c = \frac{3}{5} \\
m \rightarrow b: & \quad c = \frac{4}{5} \\
b,m \rightarrow c: & \quad c = \frac{3}{4} \\
b \rightarrow m: & \quad c = \frac{4}{6}
\end{align*}

\[
\text{conf}(I \rightarrow j) = \frac{\text{supp}(I,j)}{\text{supp}(I)}
\]
Back to finding frequent itemsets

Typically, data is kept in flat files rather than in a database system:

- Stored on disk
- Stored basket-by-basket
- Baskets are small but we have many baskets and many items
  - Expand baskets into pairs, triples, etc. as you read baskets
  - Use $k$ nested loops to generate all sets of size $k$

Note: We want to find frequent itemsets. To find them, we have to count them. To count them, we have to generate them.
Computing Itemsets

- Cost of mining is the *number of disk I/Os*
  - *Disk access costs:*
    - *Seek time (milliseconds)*
    - *Rotational latency (milliseconds)*
    - *Transfer rate (100 MB/sec)*
    - *RAM (nanoseconds): factor of $10^6$ faster than disk*

- In practice, association-rule algorithms read data in passes
- We measure the cost by the *number of passes* over the data
- Main memory bottleneck:
  - As we read the baskets, we need to count the pairs, triples, …
  - The number of different things we can count is limited by main memory
  - Swapping counts in/out is a disaster. Why?
Finding Frequent Pairs

• This is the hardest problem!
  • Often, frequent pairs are common, frequent triples are rare
  • The probability of being frequent drops with size

• We always need to generate all the itemsets

• But we would only like to count/keep track of those itemsets that in the end turn out to be frequent
Naïve Algorithm

- Read file once, counting in main memory the occurrences of each pair
  - From each basket of $n$ items, generate its $n(n-1)/2$ pairs using two nested loops
- Problem: fails if $n^2$ exceeds main memory
  - 120K (Walmart); 10B (Web pages)
Example: Counting Pairs

- Suppose 10^5 items at Walmart
- Suppose counts are 4-byte integers.
- Number of pairs of items: 10^5(10^5-1)/2 = 5*10^9 (approximately).
- Number of bytes: 4*5*10^9 = 2*10^{10} (20 gigabytes) of main memory needed.
Details of Main-Memory Counting

- **Two approaches:**
  
  1. Count all pairs, using a triangular matrix.
     - requires only 4 bytes/pair.
     
     **Note:** always assume *integers are 4 bytes*.

  2. Keep a table of triples \([i, j, c] = \) “the count of the pair of items \{i, j\} is \(c\).”
     - requires 12 bytes, but only for those pairs with count > 0.
Comparing Approaches

Method (1)

4 per pair

Method (2)

12 per occurring pair
Triangular-Matrix Approach

- $n =$ total number or items
- 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 $\{1,2\}, \{1,3\}, \ldots, \{1,n\}, \{2,3\}, \{2,4\}, \ldots, \{2,n\}, \{3,4\}, \ldots, \{3,n\}, \ldots \{n-1,n\}$.
- Pair $\{i,j\}$ is at position $(i-1)(n-i/2) + j - i$
- Total number of pairs $n(n-1)/2$; total bytes $2n^2$
**Triangular-Matrix Approach**

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

- Pair $\{i,j\}$ is at position $(i-1)(n-i/2) + j - i$
- $\{1,2\}: 0 + 2 - 1 = 1$
- $\{1,3\} = 0 + 3 - 1 = 2$
- $\{1,4\} = 0 + 4 - 1 = 3$
- $\{1,5\} = 0 + 5 - 1 = 4$
- $\{2,3\} = (2-1)*(5-2/2) + 3 - 2 = 5$
- $\{2,4\} = (2-1)*(5-2/2) + 4 - 2 = 6$
- $\{2,5\} = (2-1)*(5-2/2) + 5 - 2 = 7$
- $\{3,4\} = (3-1)*(5-3/2) + 4 - 2 = 8$
Details of Approach #2

- Total bytes used is about $12p$, where $p$ is the number of pairs that actually occur.
  - $[i, j, \text{count}]$ – 3 integers are needed – 12 bytes
  - Save space by not storing triples for pairs with count=0
  - Beats triangular matrix if at most 1/3 of possible pairs actually occur
    - Because it uses 3 times more memory per pair than matrix

- May require extra space for retrieval structure, e.g., a hash table.
  - $h(i,j) \rightarrow \text{count}$
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.

- The *downward closure* property of frequent patterns
  - *Any subset of a frequent itemset must be frequent*
  - If \{beer, diaper, nuts\} is frequent, so is \{beer, diaper\}
  - i.e., every transaction having \{beer, diaper, nuts\} also contains \{beer, diaper\}

- *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)

- Candidate generation-and-test approach

- **Apriori pruning principle**: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @ VLDB’94, Mannila, et al. @ KDD’94)
Max-Patterns

• A long pattern contains a combinatorial number of sub-patterns, e.g., \( \{a_1, \ldots, a_{100}\} \) contains \( (100^1) + (100^2) + \ldots + (1^0 \ 0^0 \ 0^0) = 2^{100} - 1 = 1.27 \times 10^{30} \) sub-patterns!

• Solution: Compact the output by mining max-patterns instead

• An itemset \( X \) is a max-pattern if \( X \) is frequent and there exists no frequent super-pattern \( Y \supset X \) (proposed by Bayardo @ SIGMOD’98)
Compacting the Output

**Maximal Frequent itemsets:**
no immediate superset is frequent

<table>
<thead>
<tr>
<th>Count</th>
<th>Maximal</th>
<th>S=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>No</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>AB</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>AC</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>BC</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>ABC</td>
<td>2</td>
<td>No</td>
</tr>
</tbody>
</table>

BC is also frequent
Frequent and its only superset ABC is not frequent
A-Priori Algorithm – (3)

• **Pass 1**: Read baskets and count in main memory the occurrences of each item.
  • Requires only memory proportional to #items n.

• Items that appear at least $s$ times are the **frequent items**.
  • Typical $s=1\%$ -- many singletons will be infrequent

• **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 (for counts) – $2m^2$ instead of $2n^2$
  • Plus a list of the frequent items (so you know what must be counted).
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>

**Sup**\(_{min} = 2\)

\(C_1\) 1st scan

- \(\{A\}\) sup 2
- \(\{B\}\) sup 3
- \(\{C\}\) sup 3
- \(\{D\}\) sup 1
- \(\{E\}\) sup 3

\(L_1\)

<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>{D}</td>
<td>1</td>
</tr>
<tr>
<td>{E}</td>
<td>3</td>
</tr>
</tbody>
</table>

\(C_2\) 2nd scan

- \(\{A, B\}\) sup 1
- \(\{A, C\}\) sup 2
- \(\{A, E\}\) sup 1
- \(\{B, C\}\) sup 2
- \(\{B, E\}\) sup 3
- \(\{C, E\}\) sup 2

\(L_2\)

<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>

\(C_3\) 3rd scan

- \{B, C, E\}

\(L_3\)

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

**Sup**\(_{min} = 2\)**
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>

Sup\(_{\text{min}} = 2\)

\(L_1\)  

<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>
</tbody>
</table>

1st scan

Should we count:  

ABC  
ACE

\(C_1\)  

<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>
</tbody>
</table>

2nd scan

\(L_2\)  

<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>

\(C_2\)  

<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>

3rd scan

\(C_3\)  

<table>
<thead>
<tr>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B, C, E}</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>
Main-Memory Picture of A-Priori

Pass 1

n

Item counts

Pass 2

Frequent items

Counts of pairs of frequent items

m < n
Detail for A-Priori

- You can use the triangular matrix method with $m = \text{number of frequent items}$.
  - May save space compared with storing triples.
- **Trick**: re-number frequent items 1, 2, … and keep a table relating new numbers to original item numbers.

---

### Item counts

Pass 1

- Main memory
- **Item counts**
- **Frequent items**
- **Old item #s**

Pass 2

- Counts of pairs of frequent items
• For each $k$, we construct two sets of $k$–tuples (sets of size $k$):
  • $C_k = \text{candidate } k$-sets = those that might be frequent sets (support $\geq s$) based on information from the pass for $k - 1$.
  • $L_k = \text{the set of truly frequent } k$-sets.
Frequent Triples, Etc.

All items

Count the items

All pairs of items from $L_1$

Count the pairs

To be explained

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 $\rightarrow C_3 \rightarrow$

Filter

Construct

Count

Construct

All pairs of items from $L_1$

Filter

Count

Construct

To be explained

First pass

Frequent items

Second pass

Frequent pairs
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 – use A-Priori property!
  - 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 $C_k$

{A,C} \hspace{1cm} sup=2
{B,C}
{B,E}
{C,E}

\rightarrow

\{A,B,C\} \hspace{1cm} X
\{A,C,E\} \hspace{1cm} X
\{B,C,E\} \hspace{1cm} \checkmark
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.
A-Priori for All Frequent Itemsets

- Many possible extensions:
  - Lower the support $s$ as itemset gets bigger
  - Association rules with intervals:
    - For example: Men over 65 have 2 cars
  - Association rules when items are in a taxonomy
    - Bread, Butter $\rightarrow$ FruitJam
    - BakedGoods, MilkProduct $\rightarrow$ PreservedGoods
Improvements to A-Priori

Modified Slides by Jeff Ullman
Association Rules: Not enough memory

- Counting for candidates C2 requires a lot of memory -- $O(n^2)$

- Can we do better?

1. **PCY**: In pass 1, there is a lot of memory left, leverage that to help with pass 2
   - Maintain a hash table with as many buckets as fit in memory
   - Keep count for each bucket into which pairs of items are hashed
     - Just the count, not the pairs!

2. **Multistage** improves PCY
Aside: Hash-Based Filtering

- **Simple problem:** Given a set $S$ of one billion strings of length 10.
  - E.g., allowed email addresses, not spam

- Need to scan a larger file $F$ of strings and output those that are in $S$.
  - I.e., filter spam

- Main memory available = 1GB = $10^9$ bytes
  - So I can’t afford to store $S$ in memory – need $10 \times 4 \times 10^9$ bytes
Solution

• Create a bit array of 8 billion bits initially all 0’s.
  • Use up the 1GB!

• Choose a hash function \( h \) with range \([0, 8*10^9]\), and hash each member of \( S \) to one of the bits, which is then set to 1.

• Filter the file \( F \) by hashing each string and outputting only those that hash to a 1.

For more details, see Mining Data Streams chapter, in Mining of Massive Datasets
Solution (cont.)

\[ h(s_1), h(s_2), \ldots, h(s_n) = 0010001011000 \]

Drop; surely not in \( S \).

To output; may be in \( S \).

False positives are possible
Solution (cont.)

- If a string is in $S$, it surely hashes to a 1, so it always gets through.

- Can repeat with another hash function and bit array to reduce the *false positives*.

- Each filter step costs one pass through the remaining file $F$ and reduces the fraction of false positives.

- Repeat passes until few false positives.

- Either accept some errors, or check the remaining strings.
  - e.g., divide surviving $F$ into chunks that fit in memory and make a pass though $S$ for each.
PCY Algorithm – An Application of Hash-Filtering

for each basket:
  for each item in basket:
    add 1 to item’s count;
  for each pair of items:
    hash pair to a bucket
    add 1 to the count for that bucket

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

2. We are not just interested in the presence of a pair, but we need to see whether it is present at least $s$ (support) times.
PCY Algorithm

- A bucket contains a *frequent pair* if its count is at least the support threshold.

- If a bucket contains a frequent pair, the bucket is surely frequent
  - Even without any frequent pair, a bucket can be frequent.
  - We cannot use the hash to eliminate any member of this bucket

- If a bucket is not frequent, no pair that hashes to that bucket could possibly be a frequent pair.
  - *Pairs that hash to this bucket can be eliminated as candidates*

- On Pass 2, we only count pairs that hash to frequent buckets.
Main-Memory: PCY

- **Pass 1**
  - Hash table for pairs

- **Pass 2**
  - Item counts
  - Frequent items
  - Bitmap
  - Counts of candidate pairs
PCY Algorithm – Before Pass 1 Organize Main Memory

- 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.
PCY Algorithm – Pass 1

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

83
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, decide which items are frequent and list them for the second pass.
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.

• Both conditions are necessary for the pair to have a chance of being frequent.
Main-Memory: PCY

- **Pass 1**
  - Item counts
  - Hash table for pairs

- **Pass 2**
  - Frequent items
  - Bitmap
  - Counts of candidate pairs
Memory Details

• Buckets require a few bytes each.
  • Note: we don’t have to count past $s$.
  • # buckets is $O$(main-memory size).

• On second pass, a table of (item, item, count) triples is essential
  • Pairs of frequent items that PCY avoids counting are placed randomly in the matrix – can’t compact the matrix
  • Thus, hash table must eliminate 2/3 of the candidate pairs for PCY to beat a-priori.
Refinement: Multistage Algorithm

• Further reduce the number of candidates to be counted
  • Remember: memory is the bottleneck
  • Still need to generate all itemsets
  • Uses several successive hash tables --- reduce the number of false positives
    • Requires more than two passes

• **Key idea:** After Pass 1 of PCY, rehash only those pairs that qualify for Pass 2 of PCY.
  • i and j are frequent, and
  • \{i,j\} hashes to a frequent bucket from Pass 1

• On middle pass, fewer pairs contribute to buckets, so fewer *false positives* — frequent buckets with no frequent pair.

• Requires 3 passes over the data
Multistage Picture

**Pass 1**
- Count items
- Hash pairs \( \{i,j\} \)

**Pass 2**
- Hash pairs \( \{i,j\} \) into Hash2 iff:
  - \( i,j \) are frequent,
  - \( \{i,j\} \) hashes to freq. bucket in B1

**Pass 3**
- Count pairs \( \{i,j\} \) iff:
  - \( i,j \) are frequent,
  - \( \{i,j\} \) hashes to freq. bucket in B1
  - \( \{i,j\} \) hashes to freq. bucket in B2
• Count only those pairs \( \{i, j\} \) that satisfy these candidate pair conditions:
  1. Both \( i \) and \( j \) are frequent items.
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is 1.
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is 1.
Important Points

1. The two hash functions have to be independent.

2. We need to check both hashes on the third pass.
   • If not, we would wind up counting pairs of frequent items that hashed first to an infrequent bucket but happened to hash second to a frequent bucket.
Multihash

• **Key idea**: use several independent hash tables on the first pass.
Multihash Picture

Main memory

Pass 1

Item counts
First hash table
Second hash table

Pass 2

Freq. items
Bitmap 1
Bitmap 2
Counts of candidate pairs
Multihash

- **Key idea**: use several independent hash tables on the first pass.

- **Risk**: halving the number of buckets doubles the average count. We have to be sure most buckets will still not reach count $s$.

- If so, we can get a benefit like multistage, but in only 2 passes.
Extensions

- Either multistage or multihash can use more than two hash functions.

- In multistage, there is a point of diminishing returns, since the bit-vectors eventually consume all of main memory.

- For multihash, the bit-vectors occupy exactly what one PCY bitmap does, but too many hash functions makes all counts $\geq s$. 
All (Or Most) Frequent Itemsets In \( \leq 2 \) 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:
  - Random sampling
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen (see textbook)
Random Sampling

- Take a random sample of the market baskets.

- Run a-priori 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.
  - Be sure you leave enough space for counts.

- 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 \( s \).
Random Sampling: Option

• Optionally, verify that your guesses are truly frequent in the entire data set by a second pass (avoid 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.
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

SON Algorithm – (1)

• Repeatedly read small subsets of the baskets into main memory and perform the first pass of the simple algorithm on each subset.
  • This is not sampling but processing the entire file in memory-sized chunks

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

- Take the union of all frequent itemsets found in one or more chunks – these are the candidate itemsets.
- On a 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.
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.
SON Mapreduce

- **Map 1**: lower support -- support = \( s \times p \)
  - Each mapper gets a fraction \( p \) of input
  - Output: (FrequentItemset, 1)

- **Reduce 1**: reducer is assigned a set of itemsets
  - Output: itemsets that appear 1 or more times

- **Map2**: all candidates and portion of input file
  - Count occurrences of candidates in file
  - Output: (CandidateSet, support)

- **Reduce 2**: sum support values for CandidateSet
Mining Various Kinds of Association Rules

- Mining multilevel association
- Mining multidimensional association
- Mining quantitative association
- Mining interesting correlation patterns

See Data Mining: Concepts and Techniques, by Han and Kamber
Mining Multiple-Level Association Rules

- Items often form hierarchies
- Flexible support settings
  - Items at the lower level are expected to have lower support
- Exploration of shared multi-level mining (Agrawal & Srikant@VLB’95, Han & Fu@VLDB’95)

**uniform support**

- Level 1
  - \( \text{min\_sup} = 5\% \)

  - \text{Milk} [support = 10\%]

- Level 2
  - \( \text{min\_sup} = 5\% \)

  - 2\% Milk [support = 6\%]
  - Skim Milk [support = 4\%]

**reduced support**

- Level 1
  - \( \text{min\_sup} = 5\% \)

- Level 2
  - \( \text{min\_sup} = 3\% \)
Mining Multi-Dimensional Association

- Single-dimensional rules:
  \[ \text{buys}(X, \text{"milk"}) \Rightarrow \text{buys}(X, \text{"bread"}) \]

- Multi-dimensional rules: \( \geq 2 \) dimensions or predicates
  - Inter-dimension assoc. rules (\textit{no repeated predicates})
    \[ \text{age}(X, \text{"19-25"}) \land \text{occupation}(X, \text{"student"}) \Rightarrow \text{buys}(X, \text{"coke"}) \]
  - Hybrid-dimension assoc. rules (\textit{repeated predicates})
    \[ \text{age}(X, \text{"19-25"}) \land \text{buys}(X, \text{"popcorn"}) \Rightarrow \text{buys}(X, \text{"coke"}) \]

- Categorical Attributes: finite number of possible values, no ordering among values—data cube approach

- Quantitative Attributes: numeric, implicit ordering among values—discretization, clustering, and gradient approaches
From Associations to Correlations

- *play basketball* ⇒ *eat cereal* [40%, 66.7%] is misleading
  - The overall % of students eating cereal is 75% > 66.7%.
  - Rules is only an estimate of the conditional probability of “eat cereal” given “play basketball”

- *play basketball* ⇒ *not eat cereal* [20%, 33.3%] is more accurate, although with lower support and confidence

- Measure of dependent/correlated events: **lift**

\[
\text{lift} = \frac{P(A \cup B)}{P(A)P(B)}
\]

\[
\text{lift}(B,C) = \frac{2000/5000}{3000/5000 \times 3750/5000} = 0.89
\]

**B is negatively correlated with C**

\[
\text{lift}(B, \neg C) = \frac{1000/5000}{3000/5000 \times 1250/5000} = 1.33
\]

**B is correlated with not C**
Summary - 1

• Market-basket model: useful for different applications – use your imagination!
  • Products and transactions: item placement, sales strategies, etc
  • Sentences and documents: plagiarism
  • Patients and drugs/side effects: drug combinations that result in particular side effects

• Frequent itemsets and association rules

• The bottleneck: counting pairs
  • Triangular matrices: save space by mapping a matrix into a 1-dimensional array
  • Triples: if fewer than 1/3 of pairs actually occur in baskets, triples are more efficient than triangular matrices
Summary - 2

- Monotonicity of frequent itemsets → allow for efficient algorithms
  - No need to count all itemsets!

- A-priory algorithm: find all pairs in 2 passes
  - Additional passes for bigger sets

- PCY algorithm: leverages spare main memory in first pass to reduce the number of pairs that need to be counted

- Multistage: multiple passes to hash pairs to different hash tables

- Multihash: use multiple hash tables in the first pass

- Randomized: user random samples instead of the full data set: may result in false positive and negatives

- SON algorithm: improvement over randomized – divide and conquer