Frequent Itemsets and Association Rule Mining

Vinay Setty
vinay.j.setty@uis.no

Slides credit: http://www.mmds.org/
Association Rule Discovery

Supermarket shelf management – Market-basket model:

- **Goal:** Identify items that are bought together by sufficiently many customers

- **Approach:** Process the sales data collected with barcode scanners to find dependencies among items

- **A classic rule:**
  - If someone buys diaper and milk, then he/she is likely to buy beer
  - Don’t be surprised if you find six-packs next to diapers!
The Market-Basket Model

- A large set of **items**
  - e.g., things sold in a supermarket
- A **large set** of **baskets**
- Each basket is a **small subset of items**
  - e.g., the things one customer buys on one day
- Want to discover **association rules**
  - People who bought \{x,y,z\} tend to buy \{v,w\}
  - Amazon!

### Input:

<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>

### Output:

**Rules Discovered:**

- \{Milk\} --> \{Coke\}
- \{Diaper, Milk\} --> \{Beer\}
Applications – (I)

- **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
  - Suggests tie-in “tricks”, e.g., run sale on diapers and raise the price of beer
  - Need the rule to occur frequently, or no $$’s

- **Amazon’s people who bought X also bought Y**
Applications – (2)

- **Baskets** = sentences; **Items** = documents containing those sentences
  - Items that appear together too often could represent plagiarism
  - Notice items do not have to be “in” baskets

- **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
More generally

- A general many-to-many mapping (association) between two kinds of things
  - But we ask about connections among “items”, not “baskets”

- For example:
  - Finding communities in graphs (e.g., Twitter)
Example:

- **Finding communities in graphs (e.g., Twitter)**
- **Baskets** = nodes; **Items** = outgoing neighbors
  - Searching for complete bipartite subgraphs $K_{s,t}$ of a big graph

- **How?**
  - View each node $i$ as a basket $B_i$ of nodes $i$ it points to
  - $K_{s,t} = \text{a set } Y \text{ of size } t \text{ that occurs in } s \text{ buckets } B_i$
  - Looking for $K_{s,t} \rightarrow \text{set of support } s \text{ and look at layer } t$ – all frequent sets of size $t$
Outline

First: Define

Frequent itemsets

Association rules:

Confidence, Support, Interestingness

Then: Algorithms for finding frequent itemsets

Finding frequent pairs

A-Priori algorithm
Frequent Itemsets

- **Simplest question:** Find sets of items that appear together “frequently” in baskets

- **Support** for itemset \( I \): Number of baskets containing all items in \( I \)
  
  - (Often expressed as a fraction of the total number of baskets)

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

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>Support of {Beer, Bread} = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Coke, Milk</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Beer, Bread</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Beer, Coke, Diaper, Milk</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Beer, Bread, Diaper, Milk</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Coke, Diaper, Milk</td>
<td></td>
</tr>
</tbody>
</table>
Example: Frequent Itemsets

- **Items** = \{milk, coke, pepsi, beer, juice\}
- **Support threshold** = 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\}, \{b, c\}, \{c, j\}. 
Association Rules

Association Rules:
If-then rules about the contents of baskets

\{i_1, i_2, \ldots, i_k\} \rightarrow j \quad \text{means: “if a basket contains all of } i_1, \ldots, i_k \text{ then it is likely to contain } j”

In practice there are many rules, want to find significant/interesting ones!

Confidence of this association rule is the probability of \( j \) given \( I = \{i_1, \ldots, i_k\} \)

\[
\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}
\]
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 just purchased very often (independent of $X$) and the confidence will be high.

- **Interest** of an association rule $I \rightarrow j$: 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 (usually above 0.5).
Example: Confidence and Interest

<table>
<thead>
<tr>
<th>$B_1$ = {m, c, b}</th>
<th>$B_2$ = {m, p, j}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_3$ = {m, b}</td>
<td>$B_4$ = {c, j}</td>
</tr>
<tr>
<td>$B_5$ = {m, p, b}</td>
<td>$B_6$ = {m, c, b, j}</td>
</tr>
<tr>
<td>$B_7$ = {c, b, j}</td>
<td>$B_8$ = {b, c}</td>
</tr>
</tbody>
</table>

- Association rule: \{m, b\} $\rightarrow$ c
  - **Confidence** = 2/4 = 0.5
  - **Interest** = $|0.5 - 5/8| = 1/8$
  - Item c appears in 5/8 of the baskets
  - Rule is not very interesting!
Finding Association Rules

- **Problem**: 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 on the left side

- **Hard part**: Finding the frequent itemsets!
  - If $\{i_1, i_2, \ldots, i_k\} \rightarrow j$ has high support and confidence, then both $\{i_1, i_2, \ldots, i_k\}$ and $\{i_1, i_2, \ldots, i_k, j\}$ will be “frequent”

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

- **Step 1:** Find all frequent itemsets $I$
  - (we will explain this next)
- **Step 2:** Rule generation
  - For every subset $A$ of $I$, generate a rule $A \rightarrow I \setminus A$
    - Since $I$ is frequent, $A$ is also frequent
  - **Variant 1:** Single pass to compute the rule confidence
    - confidence($A, B \rightarrow C, D$) = support($A, B, C, D$) / support($A, B$)
  - **Variant 2:**
    - **Observation:** If $A, B, C \rightarrow D$ is below confidence, so is $A, B \rightarrow C, D$
    - Can generate “bigger” rules from smaller ones!
- **Output the rules above the confidence threshold**
Example

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

- **Support threshold** \( s = 3 \), **confidence** \( c = 0.75 \)

- 1) Frequent itemsets:
  - \( \{b,m\} \quad \{b,c\} \quad \{c,m\} \quad \{c,j\} \quad \{m,c,b\} \)

- 2) Generate rules:
  - \( b \rightarrow m: c=4/6 \quad b \rightarrow c: c=5/6 \quad b,c \rightarrow m: c=3/5 \)
  - \( m \rightarrow b: c=4/5 \quad \ldots \quad b,m \rightarrow c: c=3/4 \)
  - \( b \rightarrow c, m: c=3/6 \)
Compacting the Output

› To reduce the number of rules we can post-process them and only output:

› **Maximal frequent itemsets:**
  No immediate superset is frequent
  
  › Gives more pruning

or

› **Closed itemsets:**
  No immediate superset has the same count (> 0)
  
  › Stores not only frequent information, but exact counts
### Example: Maximal/Closed

<table>
<thead>
<tr>
<th>Support</th>
<th>Support Value</th>
<th>Maximal(s=3)</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AB</td>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AC</td>
<td>2</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>BC</td>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ABC</td>
<td>2</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Frequent, but superset BC also frequent.
- Frequent, and its only superset, ABC, not frequent.
- Superset BC has same count.
- Its only superset, ABC, has smaller count.
A-Priori Algorithm
A-Priori Algorithm – (I)

- A **two-pass** approach called **A-Priori** limits the need for main memory

- **Key idea:** monotonicity
  - If a set of items $I$ appears at least $s$ times, so does every subset $J$ of $I$

- **Contrapositive for pairs:**
  - If item $i$ does not appear in $s$ baskets, then no pair including $i$ can appear in $s$ baskets

- So, how does A-Priori find freq. pairs?
A-Priori Algorithm – (2)

‣ **Pass 1:** Read baskets and count in main memory the occurrences of each **individual item**
  ‣ Requires only memory proportional to \#items

‣ **Items that appear \( \geq s \) times are the **frequent** items**

‣ **Pass 2:** Read baskets again and count in main memory **only** those pairs where both elements are frequent (from Pass 1)
  ‣ Requires memory proportional to square of **frequent** items only (for counts)
  ‣ Plus a list of the frequent items (so you know what must be counted)
Main-Memory: Picture of A-Priori

- **Pass 1**
  - Item counts

- **Pass 2**
  - Frequent items
    - Counts of pairs of frequent items (candidate pairs)
You can use the triangular matrix method with $n =$ 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
Frequent Triples, Etc.

- **For each** $k$, **we construct two sets of** *k-tuples* (sets of size $k$):
  - $C_k = \textit{candidate k-tuples} =$ those that might be frequent sets (support $\geq s$) based on information from the pass for $k-1$
  - $L_k = \textit{the set of truly frequent k-tuples}
Example

- Hypothetical steps of the A-Priori algorithm
  - $C_1 = \{ \{b\} \{c\} \{j\} \{m\} \{n\} \{p\} \}$
  - Count the support of itemsets in $C_1$
  - Prune non-frequent: $L_1 = \{ b, c, j, m \}$
  - Generate $C_2 = \{ \{b,c\} \{b,j\} \{b,m\} \{c,j\} \{c,m\} \{j,m\} \}$
  - Count the support of itemsets in $C_2$
  - Prune non-frequent: $L_2 = \{ \{b,m\} \{b,c\} \{c,m\} \{c,j\} \}$
  - Generate $C_3 = \{ \{b,c,m\} \{b,c,j\} \{b,m,j\} \{c,m,j\} \}$
  - Count the support of itemsets in $C_3$
  - Prune non-frequent: $L_3 = \{ \{b,c,m\} \}$

** Note here we generate new candidates by generating $C_k$ from $L_{k-1}$ and $L_1$. But that one can be more careful with candidate generation. For example, in $C_3$ we know $\{b,m,j\}$ cannot be frequent since $\{m,j\}$ is not frequent.
Generating Candidates – Full Example

Database D

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

\[
\text{TID}\quad\text{Items}
\]

\| 100  | 1 3 4 |
\| 200  | 2 3 5 |
\| 300  | 1 2 3 5 |
\| 400  | 2 5 |

\[
\text{C}_1 \quad \text{minsup} = 2
\]

\[
\begin{array}{|c|c|}
\hline
\text{itemset} & \text{sup.} \\
\hline
\{1\} & 2 \\
\{2\} & 3 \\
\{3\} & 3 \\
\{4\} & 1 \\
\{5\} & 3 \\
\hline
\end{array}
\]

\[
\text{L}_1
\]

\[
\begin{array}{|c|c|}
\hline
\text{itemset} & \text{sup.} \\
\hline
\{1\} & 2 \\
\{2\} & 3 \\
\{3\} & 3 \\
\{5\} & 3 \\
\hline
\end{array}
\]

C1 Scan D

\[
\text{C}_2
\]

\[
\begin{array}{|c|c|}
\hline
\text{itemset} & \text{sup.} \\
\hline
\{1 2\} & 1 \\
\{1 3\} & 2 \\
\{1 5\} & 1 \\
\{2 3\} & 2 \\
\{2 5\} & 2 \\
\{3 5\} & 3 \\
\hline
\end{array}
\]

\[
\text{L}_2
\]

\[
\begin{array}{|c|c|}
\hline
\text{itemset} & \text{sup.} \\
\hline
\{1 3\} & 2 \\
\{2 3\} & 2 \\
\{2 5\} & 3 \\
\{3 5\} & 2 \\
\hline
\end{array}
\]

\[
\text{C}_3
\]

\[
\text{C}_4 \text{ is empty}
\]

\[
\text{L}_3
\]

\[
\begin{array}{|c|c|}
\hline
\text{itemset} & \text{sup.} \\
\hline
\{2 3 5\} & 2 \\
\hline
\end{array}
\]

\[
\text{C}_3 \text{ Scan D}
\]

\[
\text{C}_4 \text{ Scan D}
\]

\[
\text{C}_3 \text{ Scan D}
\]

\[
\text{C}_4 \text{ Scan D}
\]

\[
\text{C}_3 \text{ Scan D}
\]

\[
\text{C}_4 \text{ is empty}
\]
Pruning Step

For an itemset of size $k$, check if all the itemsets of size $k-1$ are also frequent.

If any of the $k-1$ sized itemsets are not frequent, prune the itemset of size $k$.

Check to see if all these itemsets are frequent:

- $\{1 \ 3\}$
- $\{2 \ 3\}$
- $\{2 \ 5\}$
- $\{3 \ 5\}$
- $\{1 \ 3 \ 5\}$
- $\{2 \ 3 \ 5\}$

Not frequent!
A-Priori for All Frequent Itemsets

- One pass for each $k$ (itemset size)
- 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

- Many possible extensions:
  - 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
  - Lower the support $s$ as itemset gets bigger
Frequent Itemsets in $\leq 2$ Passes
Frequent Itemsets in $\leq 2$ Passes

- A-Priori takes $k$ passes to find frequent itemsets of size $k$

- **Can we use fewer passes?**

- Use 2 or fewer passes for all sizes, but may miss some frequent itemsets
  - Random sampling
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen (see textbook)
Random Sampling (I)

- Take a random sample of the market baskets

- Run a-priori or one of its improvements in main memory
  - So we don’t pay for disk I/O each time we increase the size of itemsets
  - Reduce support threshold proportionally to match the sample size
Random Sampling (2)

- Optionally, verify that the candidate pairs 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
  - Smaller threshold, e.g., $s/125$, helps catch more truly frequent itemsets
    - But requires more space
Repeatly read small subsets of the baskets into main memory and run an in-memory algorithm to find all frequent itemsets.

- Note: we are 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.
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 Summary

- **Pass 1 – Batch Processing**
  - Scan data on disk
  - Repeatedly fill memory with new batch of data
  - Run sampling algorithm on each batch
  - Generate candidate frequent itemsets

- **Candidate Itemsets** – if frequent in some batch

- **Pass 2 – Validate candidate itemsets**

- **Monotonicity Property**
  
  Itemset $X$ is frequent overall $\rightarrow$ frequent in at least one batch
SON – Distributed Version

- SON lends itself to distributed data mining

- Baskets distributed among many nodes
  - Compute frequent itemsets at each node
  - Distribute candidates to all nodes
  - Accumulate the counts of all candidates
SON: Map/Reduce

- **Phase 1:** Find candidate itemsets
  - Map?
  - Reduce?

- **Phase 2:** Find true frequent itemsets
  - Map?
  - Reduce?
PCY (Park-Chen-Chen-Yu) Algorithm
(Park-Chen-Yu) PCY Idea

- **Improvement** upon A-Priori
- **Observe** – during Pass 1, memory mostly idle

**Idea**

- Use idle memory for hash-table $H$
- **Pass 1** – hash pairs from $b$ into $H$
  - Increment counter at hash location
- **At end** – bitmap of high-frequency hash locations
- **Pass 2** – bitmap extra condition for candidate pairs
Memory Usage PCY

Candidate Items

Hash Table

Frequent Items

Bitmap

Candidate Pairs

Pass 1

Pass 2
PCY Algorithm

- **Pass 1**
  - m counters and hash-table $T$
  - Linear scan of baskets $b$
  - Increment counters for each item in $b$
  - Increment hash-table counter for each item-pair in $b$
  - Mark as frequent, $f$ items of count at least $s$
  - Summarize $T$ as bitmap (count > $s \rightarrow$ bit = 1)

- **Pass 2**
  - Counter only for F qualified pairs ($X_i, X_j$):
    - both are frequent
    - pair hashes to frequent bucket (bit=1)
  - Linear scan of baskets $b$
  - Increment counters for candidate qualified pairs of items in $b$
Multi-Stage PCY

- **Problem** – False positives from hashing
- **New Idea**
  - Multiple rounds of hashing
  - After Pass 1, get list of qualified pairs
  - In Pass 2, hash only qualified pairs
  - Fewer pairs hash to buckets → less false positives
    (buckets with count >s, yet no pair of count >s)
  - In Pass 3, less likely to qualify infrequent pairs
- **Repetition** – reduce memory, but more passes
- **Failure** – memory < $\mathcal{O}(f+F)$
Multi-Stage PCY Memory

Candidate Items

Hash Table 1

Pass 1

Frequent Items

Bitmap

Hash Table 2

Pass 2

Frequent Items

Bitmap 1

Bitmap 2

Candidate Pairs
Mining of Massive Datasets Jure Leskovec, Anand Rajaraman, Jeff Ullman, Chapter 6

http://mmds.org