Association Rules

CS345a: Data Mining
Jure Leskovec and Anand Rajaraman
Stanford University

Slides adapted from lectures by Jeff Ullman
The Market-Basket Model

- A large set of *items*
  - e.g., things sold in a supermarket
- A large set of *baskets*, each of which is a small set of the items
  - e.g., the things one customer buys on one day
- Can be used to model any many-many relationship, not just in the retail setting
- Find “interesting” connections between items
Frequent Itemsets

- Simplest question: Find sets of items that appear together “frequently” in baskets
- **Support** for itemset $I = \text{the number of baskets containing all items in } I$
  - Often expressed as a fraction of the total number of baskets
- Given a **support thresholds**, sets of items that appear in at least $s$ baskets are called **frequent itemsets**
Example: Frequent Itemsets

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

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

- Frequent itemsets: \{m\}, \{c\}, \{b\}, \{j\},
  \{m,b\}, \{b,c\}, \{c,j\}.
Applications – (1)

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

- Suppose 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** = 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** = Web pages; **items** = words.
  - Co-occurrence of relatively rare words, e.g., “Brad” and “Angelina,” may indicate an interesting relationship
Applications – (3)

- **Baskets** = patients; **items** = drugs and side-effects

- **Baskets** = movies; **items** = Oscar nominations and wins in different categories
  - Does being nominated in certain categories boost win likelihood in other categories?
If-then rules about the contents of baskets.

\[ \{i_1, i_2, ..., i_k\} \rightarrow j \] means: “if a basket contains all of \(i_1, ..., i_k\) then it is likely to contain \(j\).”

**Confidence** of this association rule is the probability of \(j \) given \(l = \{i_1, ..., i_k\}\).
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 $X$)

Interesting rules are those with high positive or negative interest values
Example: Confidence and Interest

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

- Association rule: \( \{m, b\} \rightarrow c \).
  - Confidence = \( \frac{2}{4} = 0.5 \)
  - Interest = \( 0.5 - \frac{4}{8} = 0 \)
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.
- 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.”
Computation Model

- Typically, data is kept in flat files rather than in a database system.
  - Stored on disk.
  - Stored basket-by-basket.
  - Expand baskets into pairs, triples, etc. as you read baskets.
    - Use $k$ nested loops to generate all sets of size $k$. 
Example: items are positive integers, and boundaries between baskets are $-1$. 

Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
I
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.

We measure the cost by the number of passes an algorithm takes.
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 (why?)
The hardest problem often turns out to be finding the frequent pairs.
- Often frequent pairs are common, frequent triples are rare.
- Probability of being frequent drops exponentially with size; number of sets grows more slowly with size.
- We’ll concentrate on pairs, then extend to larger sets.
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 by two nested loops

- Fails if \((\#\text{items})^2\) exceeds main memory.
  - \#items can be 100K (Wal-Mart) or 10B (Web pages).
Approach 1: Store triples \([i, j, c]\) where \(\text{count}(i, j) = c\)

- If integers and item ids are 4 bytes, needs approximately 12 bytes for pairs with count \(> 0\)
- Plus some additional overhead for a hashtable

What if most pairs occur, even if infrequently?
Approach 2: Count all pairs

- Number items 1, 2, 3, ...
- Count \{i, j\} only if i < j.

Keep pair counts in lexicographic order:
- \{1,2\}, \{1,3\},..., \{1,n\}, \{2,3\}, \{2,4\},...,\{2,n \}, \{3,4\},...
- Pair \{i, j\} is at position \((i - 1)(n - i/2) + j - i\)

Total number of pairs \(n(n - 1)/2\); total bytes about \(2n^2\)
Comparing approaches

4 bytes per pair

Triangular Matrix

12 per occurring pair

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

Items that appear at least $s$ times are the frequent 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 (for counts)
- Plus a list of the frequent items (so you know what must be counted).
Picture of A-Priori

- Item counts
- Frequent items

Pass 1

Counts of pairs of frequent items

Pass 2
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.
For each $k$, we construct two sets of $k$-sets (sets of size $k$):

- $C_k = \text{candidate } k$-sets = those that might be frequent sets based on information from the pass for $k - 1$.
- $L_k = \text{the set of truly frequent } k$-sets.
Frequent Itemsets – (1)

- **All items**
  - Count the items
  - First pass
  - Frequent items

- **All pairs of items from \( L_1 \)**
  - Count the pairs
  - Second pass
  - Frequent pairs

- **To be explained**
  - Filter
  - Construct
  - Filter
  - Construct

First pass:
- Count the items from \( C_1 \)
- Filter
- Construct

Second pass:
- Count the pairs from \( L_1 \)
- Filter
- Construct

Frequent items:
- \( C_1 \)
- \( L_1 \)

Frequent pairs:
- \( C_2 \)
- \( L_2 \)

To be explained:
- \( C_3 \)
Frequent Itemsets – (2)

- $C_1 =$ all items
- $L_k =$ members of $C_k$ with support $\geq s$.
- $C_{k+1} =$ $(k + 1)$-sets, each $k$ of which is in $L_k$.
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.
PCY Algorithm

- Observation: In pass 1 of a-priori, most memory is idle
  - We store only individual item counts
  - Can we use the idle memory to reduce memory required in pass 2?
- Pass 1 of PCY: In addition to item counts, maintain a hash table with as many buckets as will fit in memory
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
    }
}
1. For a bucket with total count less than $s$, none of its pairs can be frequent.
2. A bucket that a frequent pair hashes to is surely frequent.
3. Even without any frequent pair, a bucket can be frequent.

We can surely eliminate all pairs that hash into buckets of Type (1).
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.
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 all these conditions are necessary for the pair to have a chance of being frequent.
Picture of PCY

- Item counts
  - Hash table
  - Pass 1
- Frequent items
  - Bitmap
  - Counts of candidate pairs
  - Pass 2
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 (why?)

- Hash table must eliminate approx. 2/3 of the candidate pairs for PCY to beat a-priori.
Key idea: After Pass 1 of PCY, rehash only those pairs that qualify for Pass 2 of PCY

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

- **Item counts**
  - First hash table
  - Pass 1

- **Freq. items**
  - Bitmap 1
  - Second hash table
  - Pass 2

- **Freq. items**
  - Bitmap 1
  - Bitmap 2
  - Counts of candidate pairs
  - Pass 3

Pass 1 → Pass 2 → Pass 3
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.
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 counts.
- If so, we can get a benefit like multistage, but in only 2 passes.
Multihash Picture

- Item counts
  - First hash table
  - Second hash table
- Freq. items
  - Bitmap 1
  - Bitmap 2
- Counts of candidate pairs
- Pass 1
- Pass 2

Pass 1: Compute item counts and hash them into the first hash table.
Pass 2: For each item, hash it into the second hash table and add to the bitmap.

References:
- 413/4/2010 Jure Leskovec & Anand Rajaraman, Stanford CS345a: Data Mining
A-Priori, PCY, etc., take $k$ passes to find frequent itemsets of size $k$.

Other techniques 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 (1)

- 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 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)

- May miss some frequent itemsets
  - Smaller threshold helps catch more truly frequent itemsets.
SON Algorithm – (1)

- Repeatedly 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 Algorithm – 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?
1. **Maximal Frequent itemsets**: no immediate superset is frequent

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

Frequent, but superset BC also frequent.

Frequent, and its only superset, ABC, not freq.

Superset BC has same count.

Its only superset, ABC, has smaller count.