Association Rules
Homework

- **Gradiance:**
  - Create an account at [http://www.gradiance.com/services](http://www.gradiance.com/services)
  - The class token is: D8C4D42B
    - Enter the token at the bottom of the homepage
  - First quiz (4 questions) on MapReduce is up
  - It is due in a week (January 11)

- HW1 will be out soon – check the website

- **HW Q&A website:**
Recitations

- **Friday 1/7 5-7pm:** review of basic concepts of linear algebra, probability and statistics
- **Tuesday 1/11 5-7pm:** Hadoop Q&A session
- We will post the location and on the website and the mailing list soon
Supermarket shelf management – Market-basket model:

- **Goal:** To 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 one buys diaper and milk, then he 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 sold in a supermarket

- A large set of *baskets*, each is a small subset of 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

<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>
Given a set of baskets

Want to discover association rules:
- People who bought \{x,y,z\} tend to buy \{v,w\}
  - Amazon!

2 step approach
- 1) Find frequent \textbf{itemsets}
- 2) Generate the association rules

\begin{tabular}{|c|c|}
\hline
\textbf{TID} & \textbf{Items} \\
\hline
1 & Bread, Coke, Milk \\
2 & Beer, Bread \\
3 & Beer, Coke, Diaper, Milk \\
4 & Beer, Bread, Diaper, Milk \\
5 & Coke, Diaper, Milk \\
\hline
\end{tabular}

Rules Discovered:
- \{Milk\} $\rightarrow$ \{Coke\}
- \{Diaper, Milk\} $\rightarrow$ \{Beer\}
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**
**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\}.  

1/5/2011

Jure Leskovec, Stanford C246: Mining Massive Datasets
Applications – (1)

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

- **Real market baskets**: chain stores keep terabytes 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

- High support needed, 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.
Applications – (3)

- Finding communities in large graphs (e.g., web)
- Baskets = nodes; items = outgoing neighbors
  - Searching for complete bipartite subgraphs $K_{s,t}$ of a big graph
    - How?
      - View each node $i$ as a bucket $B_i$ 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 $K_{s,t} \rightarrow \text{set of support } s \text{ and look at layer } t \text{ – all frequent sets of size } t$

Use this to define topics:
What the same people on the left talk about on the right
Define:

- Frequent Itemsets
- Association rules:
  - Confidence, Support, Interestingness

2 algorithms for finding frequent itemsets:

- A-priori algorithm
- PCY algorithm +2 refinements:
  - Multistage Algorithm
  - Multihash Algorithm
- Random sampling and SON algorithms
**Association Rules**

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

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

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

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

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

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}

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

- Checking the confidence of association rules involving those sets is relatively easy
Mining Association Rules

- **Step 1:** Find all frequent itemsets
  - We explain this next
  - Let itemset \( I = \{i_1, i_2, \ldots, i_k\} \) be frequent
- **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
    - **Variant 2:** \( \text{conf}(AB \rightarrow CD) = \frac{\text{supp}(ABCD)}{\text{supp}(AB)} \)
      - **Observation:** If \( ABC \rightarrow D \) is below confidence, so is \( AB \rightarrow CD \)
      - Can generate “bigger” rules from smaller ones!
  - Output the rules above the confidence threshold
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 $k$
File Organization

Example: items are positive integers, and boundaries between baskets are $-1$.  

<table>
<thead>
<tr>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
<th>Item</th>
</tr>
</thead>
</table>

Basket 1

Basket 2

Basket 3

Etc.
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 makes over the data.
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 of items
  - 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 of items \( \{i_1, i_2\} \)

- Often frequent pairs are common, frequent triples are rare

- We’ll concentrate on pairs, then extend to larger sets

**The game:**

- We always need to generate all the itemsets
- But we would only like to count/keep track only of those that at 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 \( \frac{n(n-1)}{2} \) pairs by two nested loops

- Fails if \((\#\text{items})^2\) exceeds main memory
  - \textbf{Remember:} \#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, we need approximately 12 bytes for pairs with count \(> 0\)
- Plus some additional overhead for the hashtable

What if most pairs occur, even if infrequently?
Triangular Matrix Approach

- **Approach 2**: Count all pairs
  - Number items 1, 2, 3,..., n
  - 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 \((i−1)(n− i/2) + j −i\)
    - Only requires 4 bytes per pair
  - **But**: Total number of pairs \(n(n −1)/2\)

- **Approach 1** uses \(12p\) bytes,
  - \(p\) is the number of pairs that actually occur
  - Beats triangular matrix if less than 1/3 of possible pairs actually occur
Comparing approaches

- Triangular Matrix: 4 bytes per pair
- Triples: 12 per occurring pair
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
Pass 1: Read baskets and count in main memory the occurrences of each individual 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 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 $n = \text{number of frequent items}$
- 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
For each $k$, we construct two sets of \textit{k-tuples} (sets of size $k$):

- $C_k = \textit{candidate } k\text{-tuples} = \text{those that might be frequent sets (support } \geq s\text{) based on information from the pass for } k-1$
- $L_k = \text{the set of truly frequent } k\text{-tuples}$
Frequent Itemsets – (2)

- $C_1 = \text{all items}$
- $L_1 = \text{those counted on first pass to be frequent}$
- $C_2 = \text{pairs, both elements are frequent}$
  (appear in $L_1$)
- $L_2 = \text{those in } C_2 \text{ that are frequent (supp } \geq s)$

In general:

- $C_k = k \text{ –tuples, each } k-1 \text{ of which is in } L_{k-1}$
- $L_k = \text{members of } C_k \text{ with support } \geq s$
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
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 fit in memory
- Keep a count for each bucket into which pairs of items are hashed
  - Just the count, not the pairs that hash to the bucket!
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
    }
}
Observations about Buckets

- If a bucket contains a frequent pair, then the bucket is surely frequent
  - We cannot use the hash table to eliminate any member of this bucket
- Even without any frequent pair, a bucket can still be frequent
- But, for a bucket with total count less than $s$, none of its pairs can be frequent
  - Now, all pairs that hash to this bucket can be eliminated as candidates, even if the pair consists of two frequent items
PCY Algorithm – Between Passes

- Replace the buckets by a bit-vector:
  - 1 means the bucket count exceeded the support $s$ (a frequent bucket); 0 means it did 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
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 whose bit in the bit vector is 1 (i.e., frequent bucket)

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

Pass 1
- Hash table
- Item counts

Pass 2
- Bitmap
- Frequent items
- Counts of candidate pairs
Main-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 (why?)
  - Hash table must eliminate approx. 2/3 of the candidate pairs for PCY to beat a-priori.
Limit the number of candidates to be counted

- Remember: memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

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

Requires 3 passes over the data
Main-Memory: Multistage

First hash table
- Item counts
- Bitmap 1
  - Counts of candidate pairs

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

Pass 1
Pass 2
Pass 3

Freq. items

Jure Leskovec, Stanford C246: Mining Massive Datasets
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.
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
Refinement: 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
Main-Memory: Multihash

- Item counts
- First hash table
- Second hash table

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

Pass 1
Pass 2
Frequent Itemsets In $\leq 2$ Passes

- A-Priori, PCY, etc., take $k$ 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 (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 the sample size
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
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 – 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.**