Frequent Itemsets

The Market-Basket Model
Association Rules
A-Priori Algorithm

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

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.

Support

- 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*.
  - Sometimes given as a percentage.
- Given a *support threshold* *s*, 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} 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}, {c}, {b}, {j},

{m,b}, {b,c}, {c,j}.

Applications

- Items = products; baskets = sets of products someone bought in one trip to the store.
- Example application: given that many people buy beer and diapers together:
  - Run a sale on diapers; raise price of beer.
- Only useful if many buy diapers & beer.
  - Essential for brick-and-mortar stores, not on-line stores.

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.
  - But it is better if baskets have small numbers of items, while items can be in large numbers of baskets.

Applications – (3)

- Baskets = documents; items = words.
- Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.

Scale of the Problem

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

Association Rules

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

Example: Confidence

+ $B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$

– $B_3 = \{m, b\}$ $B_4 = \{c, j\}$

– $B_5 = \{m, p, b\}$ + $B_6 = \{m, c, b, j\}$

$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

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

Finding Association Rules

- Question: "find all association rules with support $\geq s$ and confidence $\geq c$."
  - Note: "support" of an association rule is the support of the set of items on the left.
- Hard part: finding the frequent itemsets.
  - Note: 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."

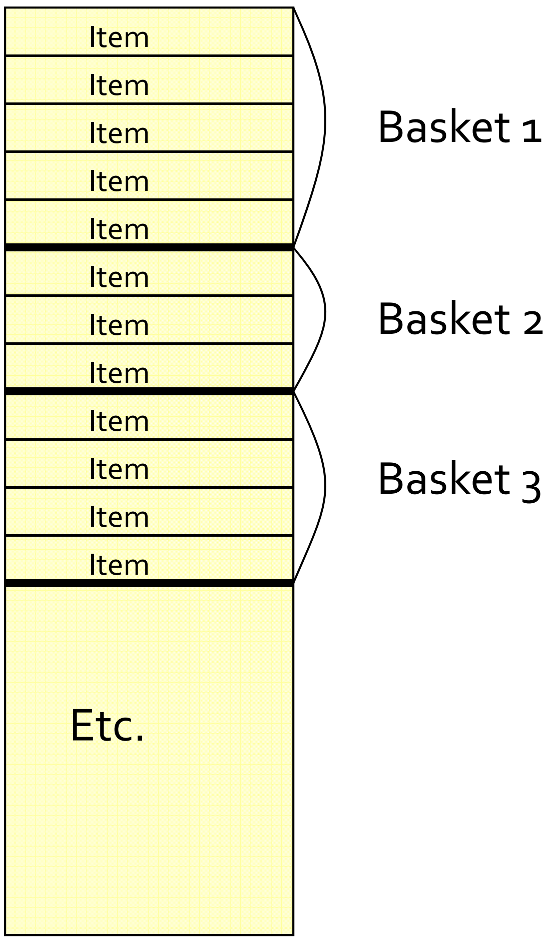
Finding Association Rules – (2)

1. Find all sets with support at least cs.
2. Find all sets with support at least s.
3. If $\{i_1, i_2,\ldots, i_k, j\}$ has support at least cs, see which subsets missing one element have support at least s.
   - Take j to be the missing element.
4. $\{i_1, i_2,\ldots, i_k\} \rightarrow j$ is an acceptable association rule if $\{i_1, i_2,\ldots, i_k\}$ has support $s_1 \geq s$, $\{i_1, i_2,\ldots, i_k, j\}$ has support $s_2 \geq cs$, and s_2/s_1, the confidence of the rule, is at least c.

Computation Model

- Typically, data is kept in flat files.
- 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.

Computation Model – (2)

- The true cost of mining disk-resident data is usually the number of disk I/O's.
- In practice, algorithms for finding frequent itemsets read the data in *passes* – all baskets read in turn.
- Thus, we measure the cost by the number of passes an algorithm takes.

Main-Memory Bottleneck

- For many frequent-itemset algorithms, main memory is the critical resource.
- As we read baskets, we need to count something, e.g., occurrences of pairs.
- The number of different things we can count is limited by main memory.
- Swapping counts in/out is a disaster.

Finding Frequent Pairs

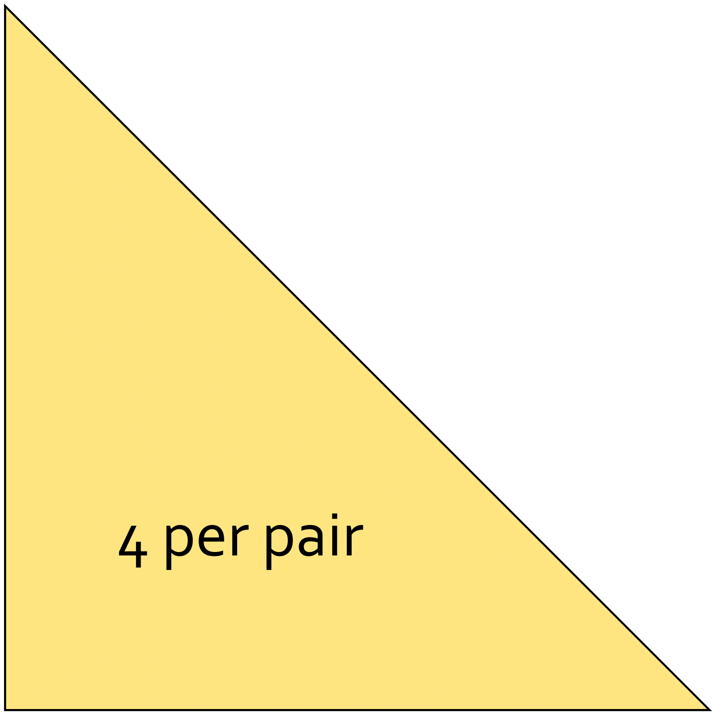
- The hardest problem often turns out to be finding the frequent pairs.
  - Why? Often frequent pairs are common, frequent triples are rare.
    - Why? Support threshold is usually set high enough that you don't get too many frequent itemsets.
- We'll concentrate on pairs, then extend to larger sets.

Naïve Algorithm

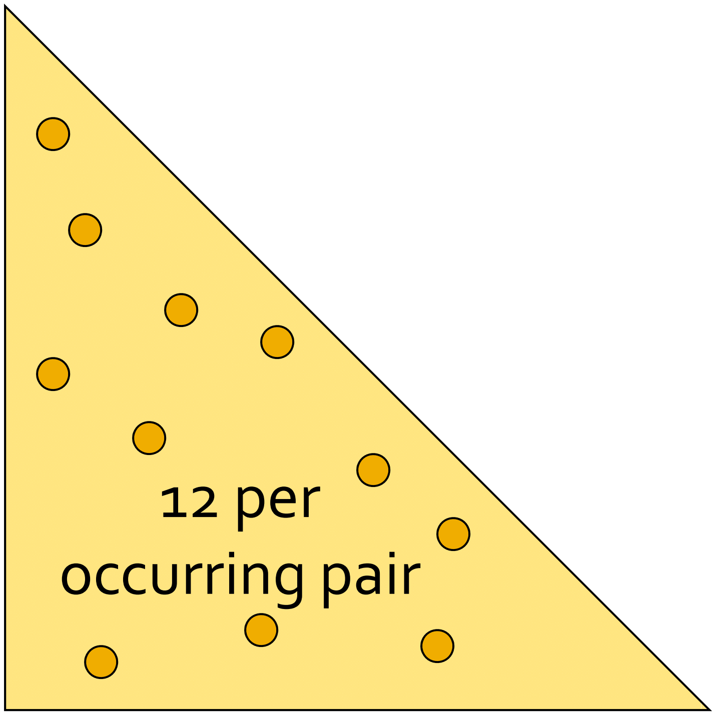
- Read file once, counting in main memory the occurrences of each pair.
  - From each basket of n items, generate its $n(n-1)/2$ pairs by two nested loops.
- Fails if $(\#\text{items})^2$ exceeds main memory.
  - Remember: #items can be 100K (Wal-Mart) or 100B (Web pages).

Details of Main-Memory Counting

- Two approaches:
  1. Count all pairs, using a triangular matrix.
  2. Keep a table of triples [*i*, *j*, *c*] = "the count of the pair of items {*i*, *j*} is *c*."
- (1) requires only 4 bytes/pair.
  - Note: always assume integers are 4 bytes.
- (2) requires 12 bytes, but only for those pairs with count > 0.



Triangular matrix



Tabular method

Triangular-Matrix Approach

- Number items 1, 2,…
  - 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 $\{1,2\}$, $\{1,3\}$,…, $\{1,n\}$, $\{2,3\}$, $\{2,4\}$,…,$\{2,n\}$, $\{3,4\}$,…, $\{3,n\}$,…$\{n-1,n\}$.

Triangular-Matrix Approach – (2)

- Find pair $\{i, j\}$, where $i<j$, at the position:

  $$(i - 1)(n - i/2) + j - i$$

- Total number of pairs $n(n-1)/2$; total bytes about $2n^2$.

Details of Tabular Approach

- Total bytes used is about 12*p*, where *p* is the number of pairs that actually occur.
  - Beats triangular matrix if at most 1/3 of possible pairs actually occur.
- May require extra space for retrieval structure, e.g., a hash table.