Association Rules

Market Baskets

Frequent Itemsets

A-Priori Algorithm

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.

Market-Baskets – (2)

- Really a general many-many mapping (association) between two kinds of things.
  - But we ask about connections among "items," not "baskets."
- The technology focuses on common events, not rare events ("long tail").

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 – (1)

- 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.

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.

Applications – (3)

- 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.

Aside: Words on the Web

- Many Web-mining applications involve words.
  1. Cluster pages by their topic, e.g., sports.
  2. Find useful blogs, versus nonsense.
  3. Determine the sentiment (positive or negative) of comments.
  4. Partition pages retrieved from an ambiguous query, e.g., "jaguar."

Words – (2)

- Here's everything I know about computational linguistics.
  1. Very common words are *stop words*.
     - They rarely help determine meaning, and they block from view interesting events, so ignore them.
  2. The TF/IDF measure distinguishes "important" words from those that are usually not meaningful.

Words – (3)

TF/IDF = "term frequency, inverse document frequency": relates the number of times a word appears to the number of documents in which it appears.

- Low values are words like "also" that appear at random.
- High values are words like "computer" that may be the topic of documents in which it appears at all.

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 $i_1,\ldots,i_k$.

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%.

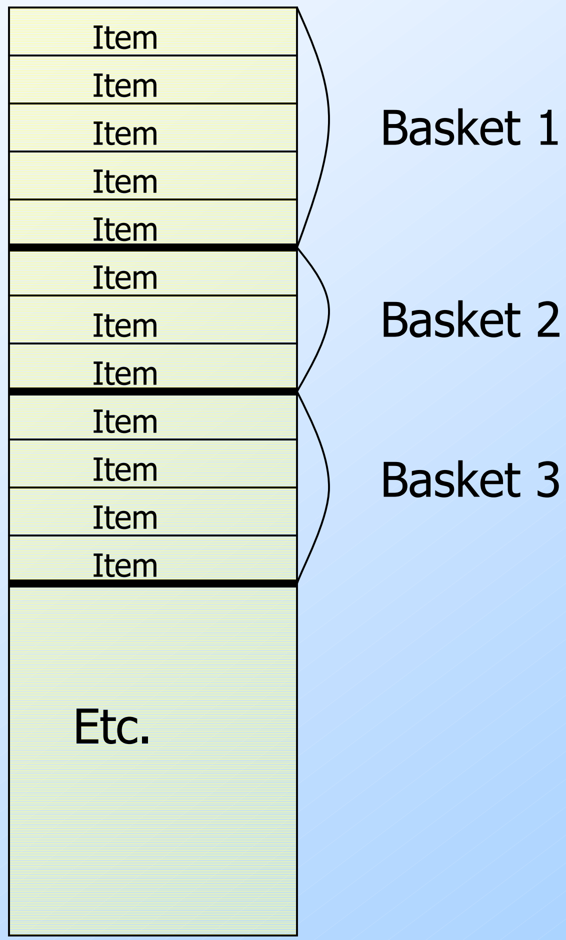
Finding Association Rules

- 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."

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.

Computation Model – (2)

- 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.
- 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 (why?).

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? 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

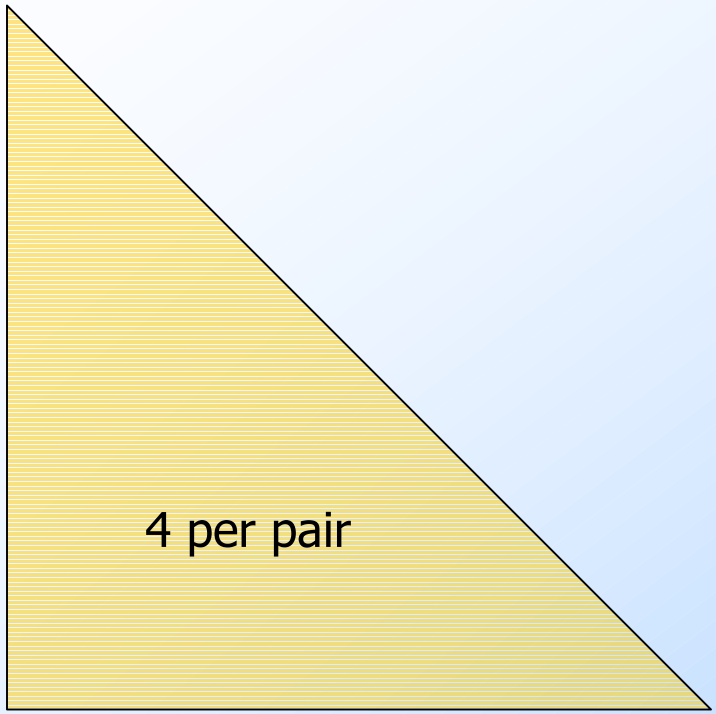
- Read file once, counting in main memory the occurrences of each pair.
  - From each basket of n items, generate its $n(n-1)/2$ pairs by two nested loops.
- Fails if $(\#items)^2$ exceeds main memory.
  - Remember: #items can be 100K (Wal-Mart) or 10B (Web pages).

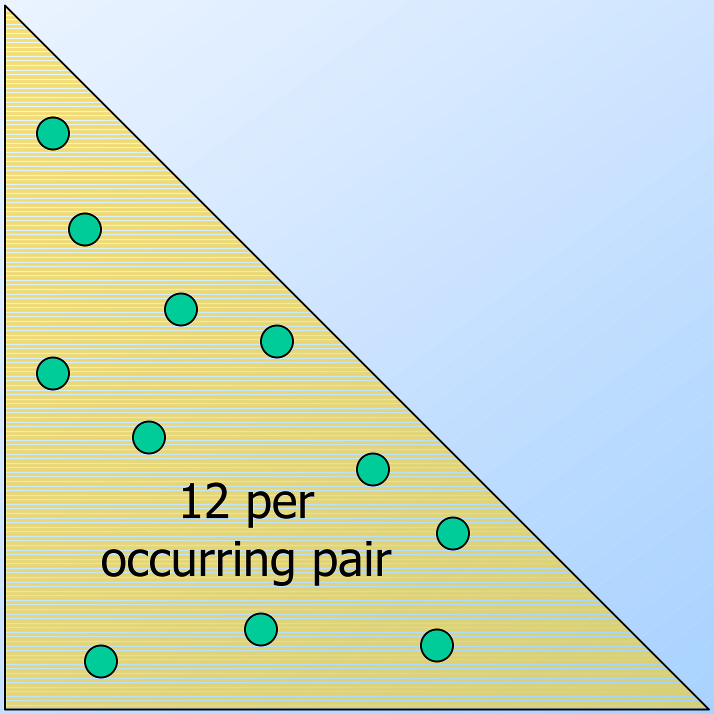
Example: Counting Pairs

- Suppose 10^5 items.
- Suppose counts are 4-byte integers.
- Number of pairs of items: $10^5(10^5-1)/2 = 5*10^9$ (approximately).
- Therefore, $2*10^{10}$ (20 gigabytes) of main memory needed.

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.





Method (1)

Method (2)

Triangular-Matrix Approach – (1)

- 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 $\{i, j\}$ 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 Approach #2

- Total bytes used is about $12p$, 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.

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.
  - Contrapositive for pairs: if item i does not appear in s baskets, then no pair including i can appear in s baskets.

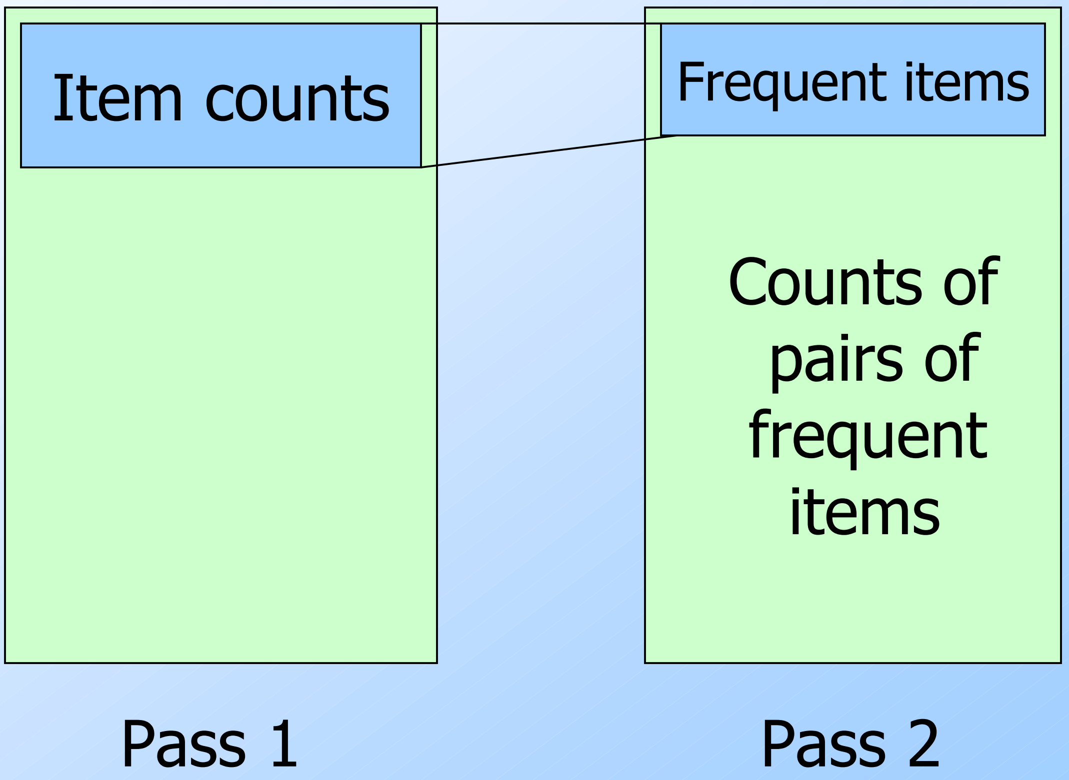
A-Priori Algorithm – (2)

- 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*.

A-Priori Algorithm – (3)

- 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).

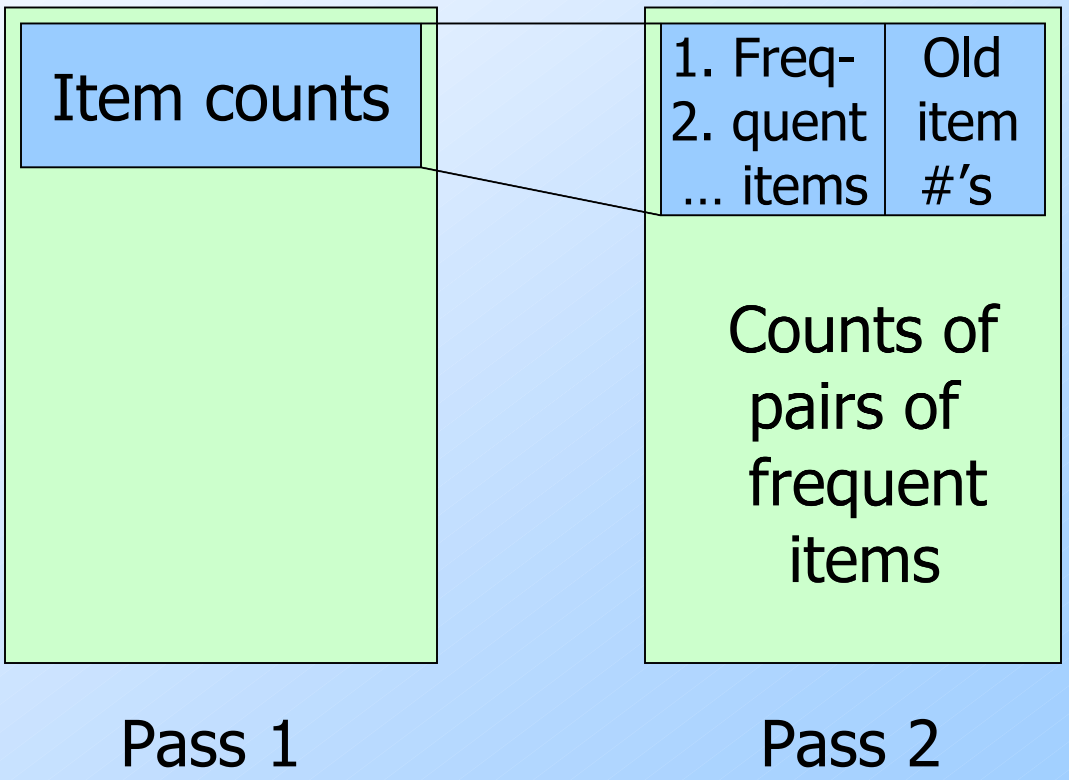
Picture of A-Priori



Detail for A-Priori

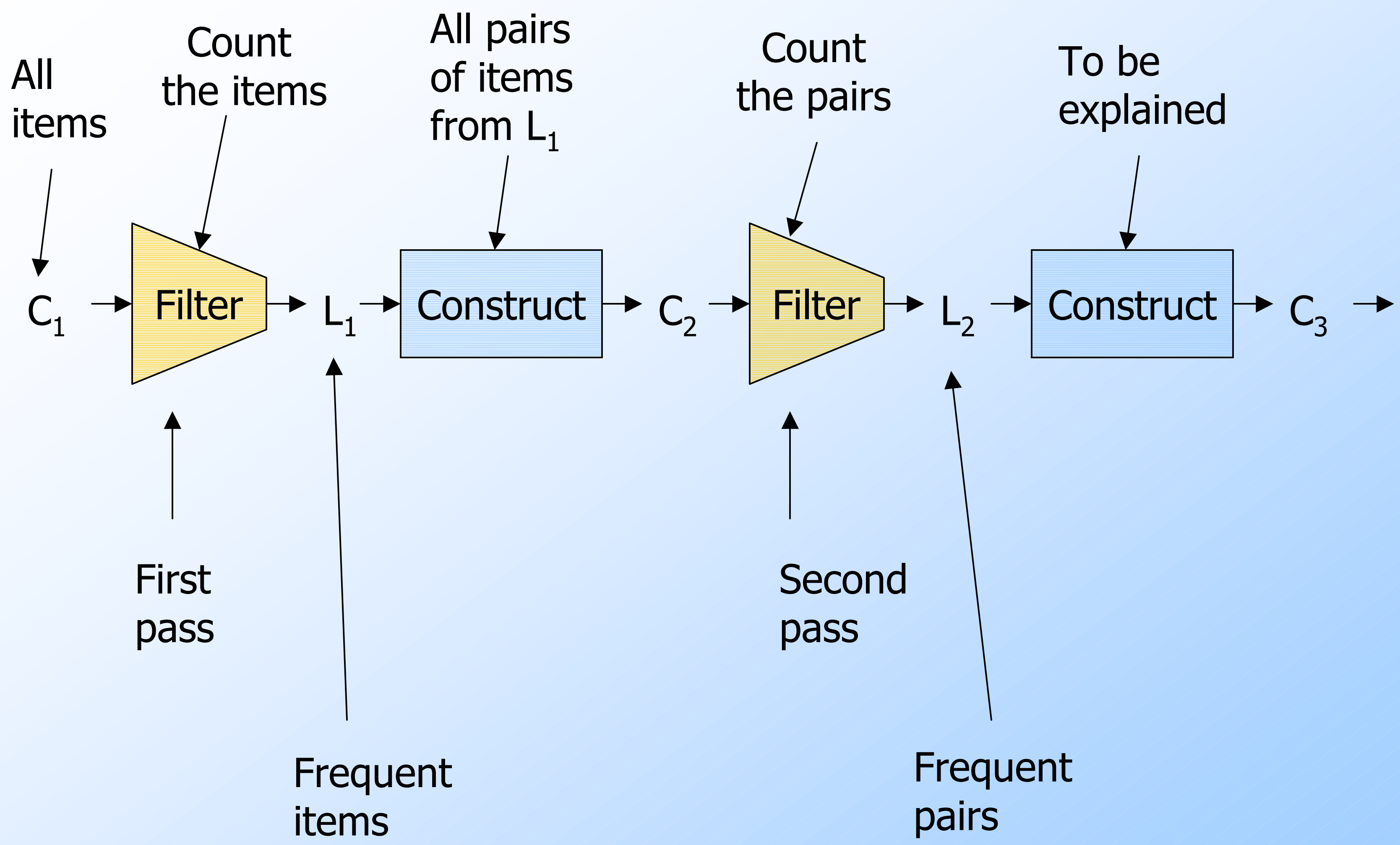
- You can use the triangular matrix method with n = number of frequent items.
  - May save space compared with storing triples.
- Trick: number frequent items 1,2,… and keep a table relating new numbers to original item numbers.

A-Priori Using Triangular Matrix for Counts



Frequent Triples, Etc.

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



All items
Count the items
All pairs of items from L_1
Count the pairs
To be explained
C_1
Filter
L_1
Construct
C_2
Filter
L_2
Construct
C_3
First pass
Second pass
Frequent items
Frequent pairs

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.

Frequent Itemsets – (2)

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