Introduction to

Machine Learning and Data Mining

(Học máy và Khai phá dữ liệu)

Khoat Than

School of Information and Communication Technology

Hanoi University of Science and Technology

2020

Contents

- Introduction to Machine Learning & Data Mining
- Unsupervised learning
- Supervised learning
- Probabilistic modeling
- **Data mining**
  - **Association rule**
  - **Apriori**
- Practical advice

Association rule discovery

- **Supermarket shelf management:** Market-basket model
  - **Goal:** identify items that are bought together by sufficiently many customers.
    (tìm ra những sản phẩm mà hay được mua cùng nhau)
  - **Approach:** process the sales data collected with barcode scanners to find dependencies among items.
- A classical rule:
  - If someone buys diaper and milk, then he/she is likely to buy beer.
  - Do not surprised if you find beer packs next to diapers !

(Adapted from a lecture by Jure Leskovec)

The Market-Basket Model

- A large set of **items**
  - E.g., things sold in a supermarket

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

- A large set of **baskets**
- Each basket
  is a small subset of items
  - e.g., the things one customer buys on one day
- **Association:** a general many-to-many mapping between two kinds of things (một ánh xạ nhiều-nhiều giữa hai loại đối tượng)
  - But we ask about *connections among "items"*, not "baskets"

Association rules: approach

- Given a set of baskets
- Want to discover **association rules**
  (tìm tập các luật kết hợp)
  - People who bought {x,y,z} tend to buy {v,w}
- **2 step approach:**
  - *Find frequent itemsets*
    (tìm tập thường xuyên)
  - *Generate association rules*
    (sinh các luật kết hợp)

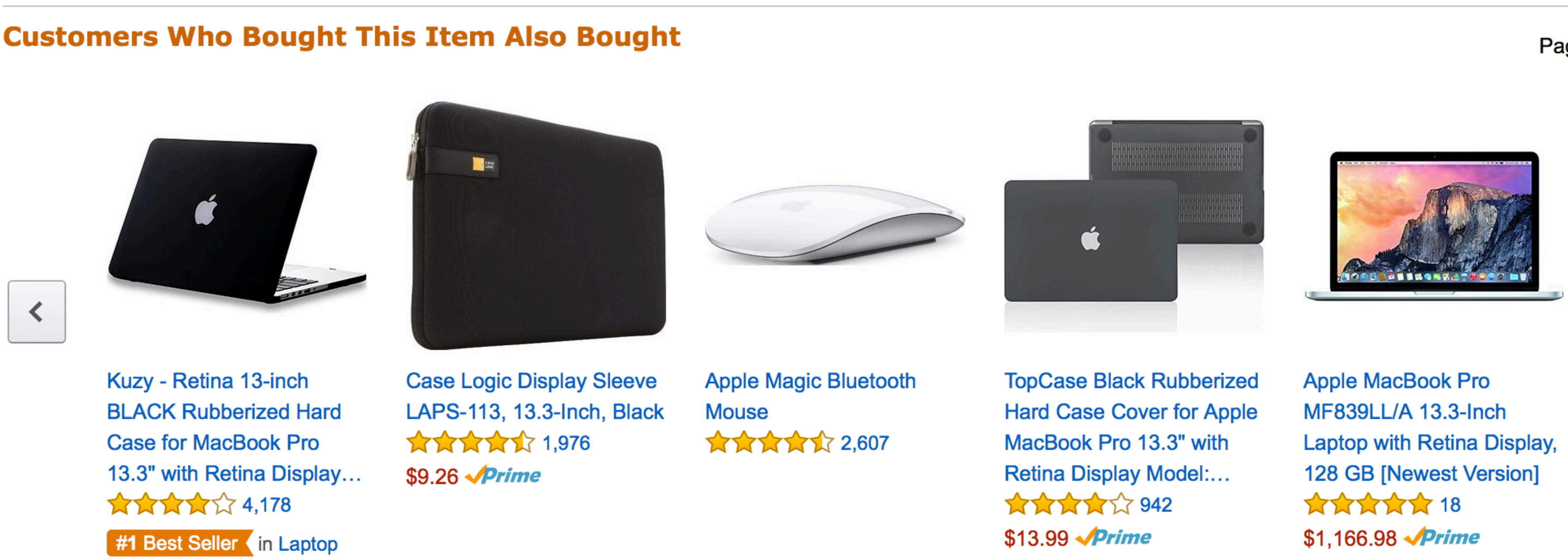
TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Rules discovered:
{Milk} → {Coke}
{Diaper, Milk} → {Beer}

Applications

- **Items** = products; **baskets** = sets of products someone bought in one trip to the store
- **Real market baskets:** stores might 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
- *Amazon’s people who bought X also bought Y.*

Applications: amazon



Frequent itemsets

- **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***.
  *(tập thường xuyên là tập những sản phẩm mà chúng xuất hiện cùng nhau trong ít nhất **s** giỏ hàng)*

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Support of {Beer, Bread} = 2

Frequent itemsets: example

- **Items** = {Milk, Coke, Pepsi, Beer, Juice}
- Support threshold = 3 baskets
  - B1 = {m, c, b}
  - B2 = {m, p, j}
  - B3 = {m, b}
  - B4 = {c, j}
  - B5 = {m, p, b}
  - B6 = {m, c, b, j}
  - B7 = {c, b, j}
  - B8 = {b, c}
- **Frequent itemsets:** {m}, {c}, {b}, {j}, {m, b}, {b, c}, {c, j}

Association rules

- **Association rules:** If-then rules about the content of baskets
- ***$\{i_1, i_2, \ldots, i_k\} \rightarrow j$*** means: "if a basket contains all of ***$\{i_1, i_2, \ldots, i_k\}$***, then it is likely to contain ***j***"

  *(nếu một giỏ hàng mà chứa **$\{i_1, i_2, \ldots, i_k\}$** thì nó cũng có thể chứa j)*
- In practice there are many rules, we want to **find significant or interesting rules.**
- ***Confidence*** of this association rule is the probability of **j** given **$I = \{i_1, i_2, \ldots, i_k\}$**
  - *Confidence của luật **I → j** là xác suất của j với điều kiện **I***

$$\mathrm{conf}(I \rightarrow j) = \frac{\mathrm{support}(I \cup j)}{\mathrm{support}(I)}$$

Interesting association rules

- Not all high-confidence rules are interesting
  - The rule ***X → milk*** may have high confidence for many itemsets X, because milk is purchased very often (independent of X) and the confidence will be high
- **Interest** of an association rule **I →j**:
  *difference between its confidence and the fraction of baskets that contain **j**.*

$$\text{Interest}(\mathbf{I} \rightarrow \mathbf{j}) = \text{conf}(\mathbf{I} \rightarrow \mathbf{j}) - \Pr(\mathbf{j})$$

  - Interesting rules are those with high positive or negative interest values (usually above 0.5)
  - Why?

Confidence and Interest: example

- B1 = {m, c, b}
- B2 = {m, p, j}
- B3 = {m, b}
- B4 = {c, j}
- B5 = {m, p, b}
- B6 = {m, c, b, j}
- B7 = {c, b, j}
- B8 = {b, c}

- Association rule: **{m,b} →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 ≥ s and confidence ≥ 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***
- **Step 2:** rule generation
  - For every subset A of ***I***, generate rule *A → I \ A*
    - Since I is frequent, A is also frequent
    - *Variant 1:* Single pass to compute the rule confidence
      confidence(A,B→C,D) = support(A,B,C,D) / support(A,B)
    - *Variant 2:*
      - Observation: If A,B,C→D is below confidence, so is A,B→C,D
      - Can generate "bigger" rules from smaller ones!
  - Output the rules above the confidence threshold

Example

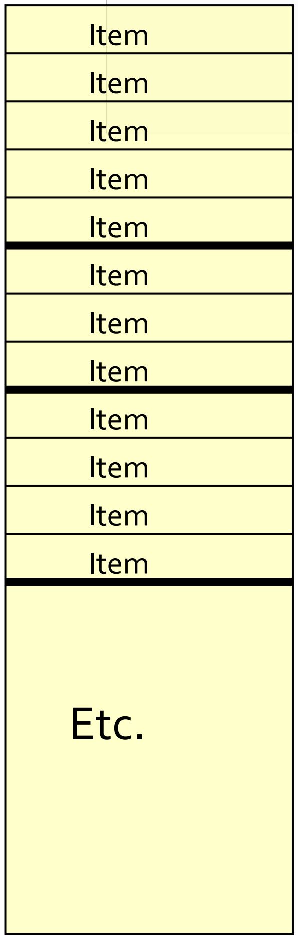
- B1 = {m, c, b}
- B2 = {m, p, j}
- B3 = {m, b}
- B4 = {c, j}
- B5 = {m, p, b, c}
- B6 = {m, c, b, j}
- B7 = {c, b, j}
- B8 = {b, c}

- Support threshold s = 3, confidence **c** = 0.75
  - *Frequent itemsets:* {b,m}, {b,c}, {c,m}, {c,j}, {m,c,b}
  - *Generate rules:*
    - ~~b→m: **c**=4/6~~
    - m→b: **c**=4/5
    - b→c: **c**=5/6 ...
    - ~~b,c→m: **c**=3/5~~
    - b,m→c: **c**=3/4
    - ~~b→c,m: **c**=3/6~~

Finding Frequent Itemsets

Itemsets: computation model

- Typically, data is kept in flat files rather than in a database system:
  - Stored on disk
  - Stored basket-by-basket
  - Baskets are small but we have many baskets and many items
    - Expand baskets into pairs, triples, etc. as you read baskets
    - Use k nested loops to generate all sets of size k



Note: We want to find frequent itemsets. To find them, we have to count them. To count them, we have to generate them.

Items are positive integers, and boundaries between baskets are -1.

Computational model

- The true cost of mining disk-resident data is usually the **number of disk I/Os**
- 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?)

Finding Frequent Pairs

- **The hardest problem often turns out to be finding the frequent pairs of items $\{i_1, i_2\}$**
  - Why?
  - Frequent pairs are common, frequent triples are rare.
  - Probability of being frequent drops exponentially with size, number of sets grows more slowly with size.
- Let's first concentrate on pairs, then extend to larger sets
- The approach:
  - We always need to generate all the itemsets.
  - But we would only like to count (keep track of) those itemsets that in the end turn out to be frequent.

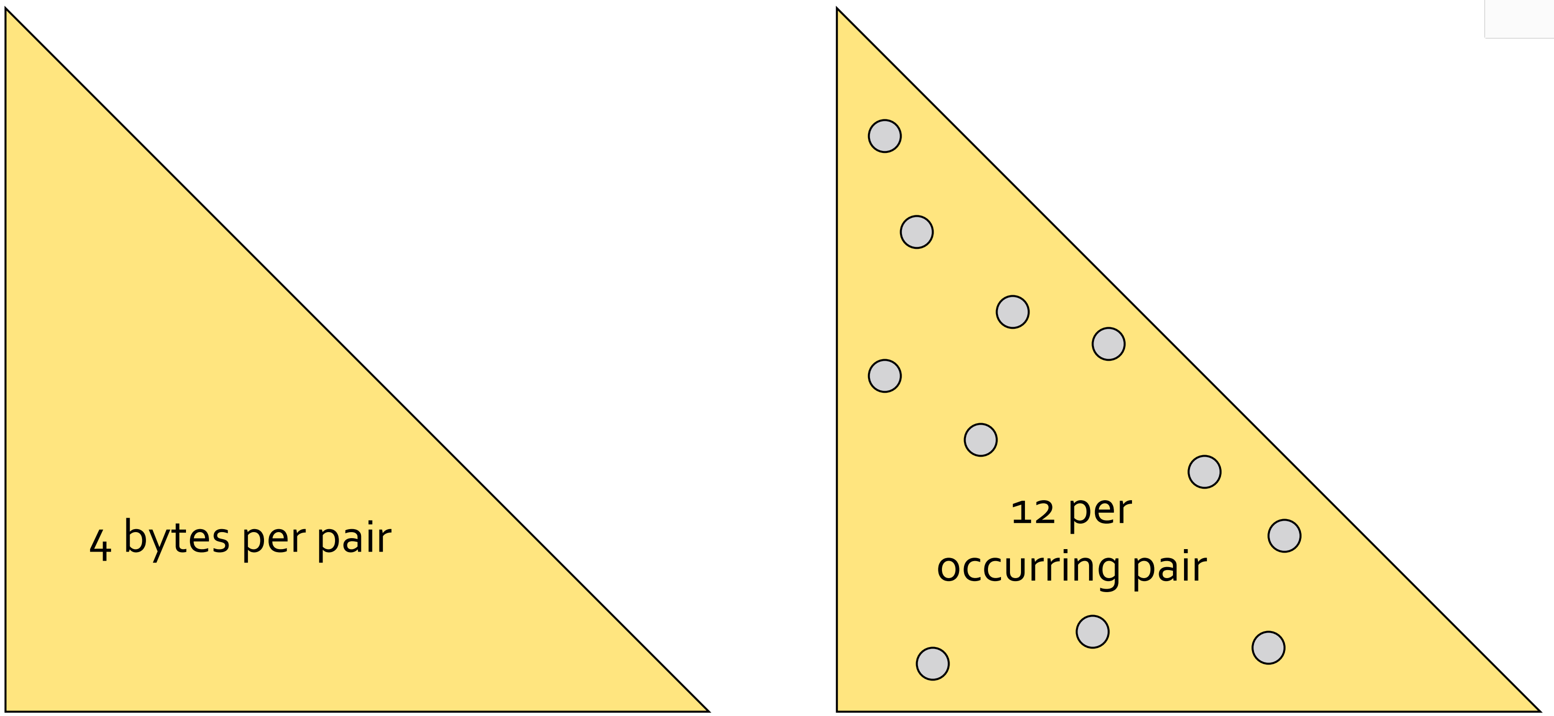
Naïve algorithm

- Naïve approach to finding frequent pairs
- *Read file once, counting in main memory the occurrences of each pair:*
  - If a basket has ***n*** items, then we need to generate ***n(n-1)/2*** pairs.
- **Fail if (#items)^2 exceeds main memory**
- In practice, #items can be
  - 100K (Wal-Mart) or 10B (Web pages)
  - Suppose 10^5 items, counts are 4-byte integers
  - Number of pairs of items: $10^5(10^5-1)/2 = 5*10^9$
  - Therefore, $2*10^{10}$ (20 gigabytes) of memory needed

Counting pairs in memory

- **Approach 1:** count all pairs using a matrix
- **Approach 2:** keep a table of triples
  {i, j, c} = "the number of pairs of items {i,j} is 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.
- **Note:**
  - Approach 1 only requires 4 bytes per pair
  - Approach 2 uses 12 bytes per pairs
    (but only for pairs with count > 0)

Comparing the two approaches



Triangular Matrix

Triples

Comparing the two approaches

- Approach 1: triangular matrix
  - n = total number of items
  - Count pair of items {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 $(i-1)(n-i/2)+j-1$
  - Total number of pairs $n(n-1)/2$; total bytes= $2n^2$
  - Triangular Matrix requires 4 bytes per pair
- Approach 2 uses 12 bytes per counting pair (but only for pairs with count > 0)
  - Beats Approach 1 if **less than 1/3** of possible pairs actually occur.

Comparing the two approaches

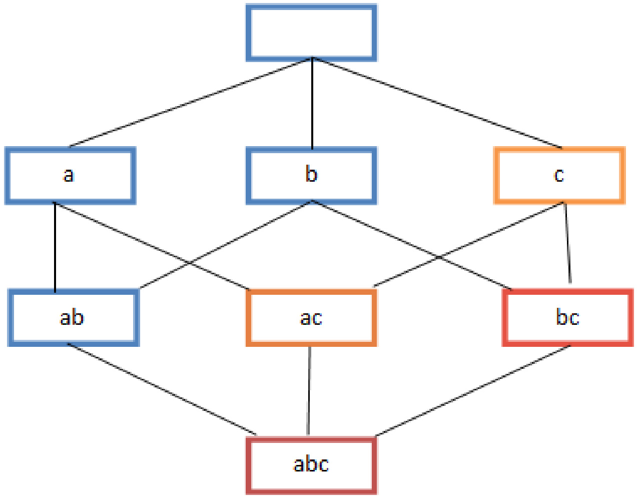
- Approach 1: triangular matrix
  - n = total number of items
  - C
  - K
    {
  - F
  - T
  - T

Problem is
the pairs do not fit into memory
if we have too many items.

Can we do better?

- Ap
  (but only for pairs with count > 0)
  - Beats Approach 1 if **less than 1/3** of possible pairs actually occur.

A-Priori algorithm (1)

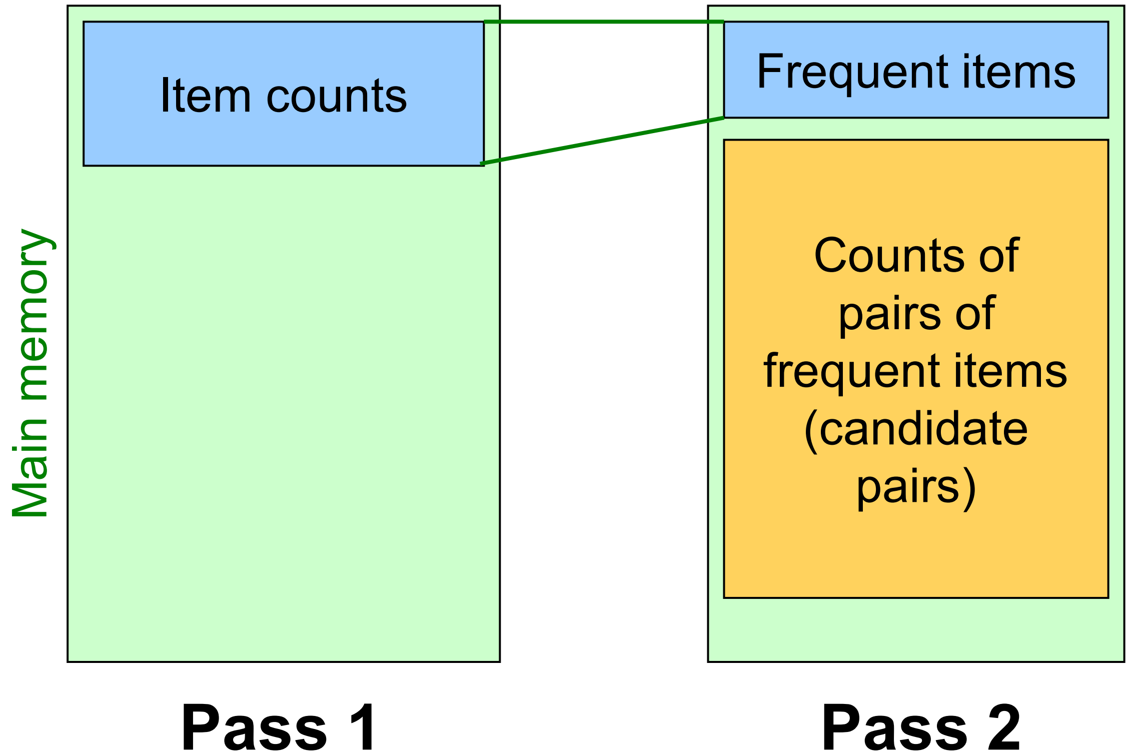


- A two-pass approach called **A-Priori** (by Agrawal and Srikant, 1994) limits the need for main memory
- Key idea: **monotonicity**
  - If a set ***I*** of items appears at least ***s*** times, so does every subset ***J*** of ***I***.
- **Contrapositive for pairs:** If item i does not appear in s baskets, then no pair including i can appear in s baskets
- So, how does A-Priori find frequent pairs?

A-Priori algorithm (2)

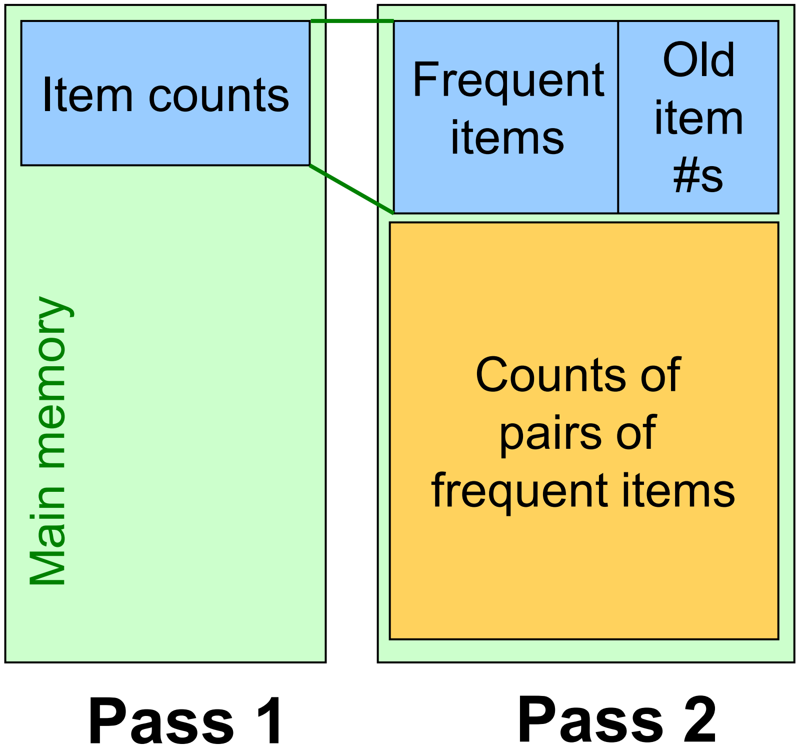
- **Pass 1:** read baskets and count in main memory the occurrences of each *individual item*.
  - Requires only memory proportional to #items
- Items that appear at least s times are the frequent items
- **Pass 2:** Read baskets again and count in main memory *only those pairs* where both elements are 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



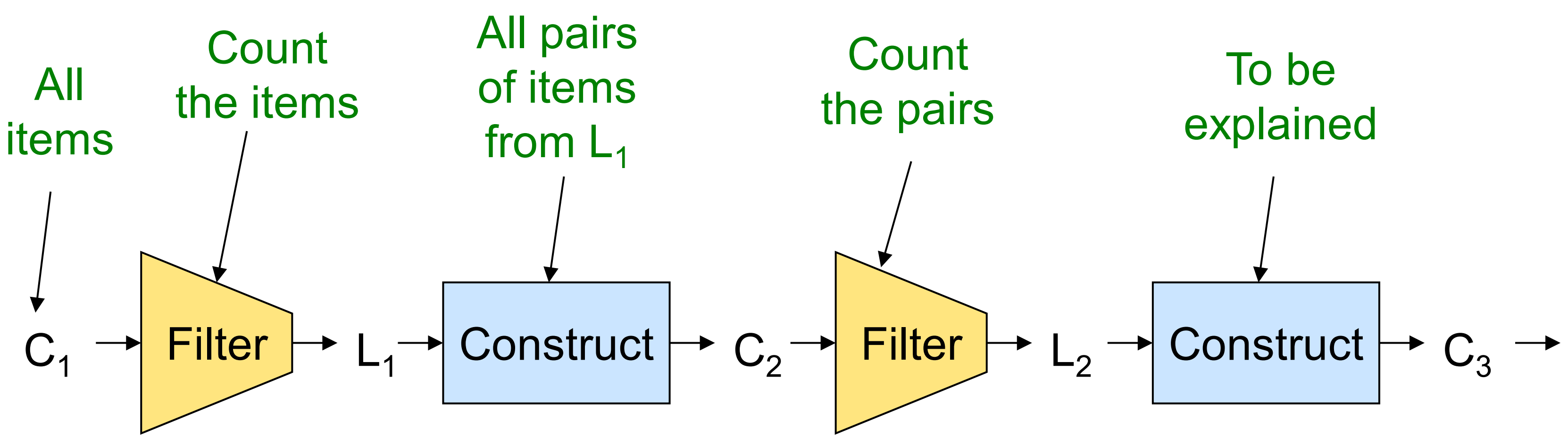
Details of A-Priori

- 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, …

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



A-Priori: example

- **Hypothetical steps of the A-Priori algorithm**
  - Generate C_1 ={{b} {c} {j} {m} {n} {p}}
  - Count the support of itemsets in C_1
  - Prune non-frequent to get 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 to get 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 to get L_3 = { {b,c,m} }

A-Priori for All frequent itemsets

- One pass for each **k** (itemset size)
- Needs 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 → FruitJam
    - BakedGoods, MilkProduct → PreservedGoods
  - Lower the support s as itemset gets bigger