INF 553: Foundations and Applications of Data Mining

Finding Frequent Itemsets
(Chapter 6)

Frequent Itemsets and Association Rules

◆ **Family of techniques for characterizing data: discovery of frequent itemsets**

➢ **E.g. Identify sets of items that are frequently purchased together**

➢ hamburger and ketchup

Frequent Itemsets and Association Rules

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Network Analysis	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Outline

◆Introduce market-basket model of data

◆Define frequent itemsets

◆Discover association rules

➢Confidence and interest of rules

◆A-Priori Algorithm and variations.

THE MARKET-BASKET MODEL

Association Rule Discovery

Supermarket shelf management – Market-basket model:

◆ **Goal:** Identify items that are bought together by **sufficiently many customers**

◆ **Approach:** Process the sales data to find dependencies among items

  ➢ Brick and mortar stores: data collected with barcode scanners

  ➢ Online retailers: transaction records for sales

◆ **A classic rule:**

  ➢ If someone buys <u>diaper and milk</u>, then he/she is likely to buy <u>beer</u>

  ➢ Don't be surprised if you find six-packs next to diapers!

The Market-Basket Model

◆ A **large set** of **items**

  ➢ e.g., things sold in a supermarket

◆ A **large set** of **baskets**

◆ Each basket is a **small** **subset of items**

  ➢ e.g., the things one customer buys on one day

◆ **Want to discover Association Rules**

  ➢ People who bought {x,y,z} tend to buy {v,w}

    • Brick and mortar stores: Influences setting of prices, what to put on sale when, product placement on store shelves

    • Recommender systems: Amazon, Netflix, etc.

Input:

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

Output:

Rules Discovered:
{Milk} --> {Coke}
{Diaper, Milk} --> {Beer}

7

Market-Baskets

◆ Really a **general many-many mapping** (association) between two kinds of things: <u>items</u> and <u>baskets</u>

  ➢ But we ask about <u>connections among "items,"</u> not "baskets."

◆ The technology focuses on common events, not rare events

  ➢ Don't need to focus on identifying *all* association rules

  ➢ Want to focus on common events, <u>focus pricing strategies or product recommendations</u> on those items or association rules.

Market Basket Applications: Identify items bought together

◆ **Items** = products

◆ **Baskets** = sets of products someone bought in one trip to the store

◆ **Real market baskets:** Stores (Walmart, Target, Ralphs, etc.) keep <u>terabytes of data</u> about what items customers buy together

  ➢ Tells how <u>typical</u> customers navigate stores

  ➢ Lets them <u>position tempting items</u>

  ➢ Suggests tie-in "tricks", e.g., run sale on diapers and raise the price of beer

  ➢ **Need the rule to occur <u>frequently</u>, or no profits!**

◆ **Amazon's people who bought *X* also bought *Y***

  ➢ Recommendation Systems.

Market Basket Applications: Plagiarism detection

◆ **Baskets** = sentences

◆ **items** = documents containing those sentences

➢ Item/document is "in" a basket if sentence is in the document

➢ May seem backward, but relationship between baskets and items is many-to-many

◆ Look for items that appear together in several baskets

➢ Multiple documents share sentence(s)

◆ Items (documents) that appear together too often could represent plagiarism.

Market Basket Applications: Identify related concepts in web documents

◆ **Baskets** = Web pages

◆ **items** = words

◆ Baskets/documents contain items/words in the document

◆ Look for sets of words that appear together in many documents

◆ Ignore most common words

◆ Unusual words appearing together in a large number of documents, e.g., "World" and "Cup," may indicate an interesting relationship or joint concept.

Market Basket Applications: Drug interactions

◆ **Baskets** = patients

◆ **items** = drugs and 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!!

Data Mining Helps in Hypothesis Formation?

◆ Background:
  ➢ A single-malt scotch whisky is a product of a single distillery
  ➢ A blended whisky is a product that contains a mix of barrel-aged malt and grain whiskies (can be from a single or multiple distilleries)

◆ Observation (Hypothesis):
  ➢ it is a popular belief that single-malt whiskies taste better than blend whiskies

 VS

◆ Experiment:
  ➢ Eight volunteers, blindfolded and given a glass of each of six whiskies
  ➢ The whiskies included three malts and three blends
  ➢ Each subject tasted each whisky six times
  ➢ They were asked whether the whisky was malt or blended, whether they could identify the distillery, and whether they liked it (ranked on a nine-point scale)
  ➢ Test to see if results are statistically significant

Chadwick, S. J., & Dudley, H. A. (1983). Can malt whisky be discriminated from blended whisky? The proof. A modification of Sir Ronald Fisher's hypothetical tea tasting experiment. British medical journal,287(6409), 1912.

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.

DEFINE FREQUENT ITEMSETS

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 of size 1: {m}, {c}, {b}, {j}

◆ Frequent itemsets of size 2

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

17

ASSOCIATION RULES

Association Rules

◆ **If-then rules about the contents of baskets**

◆ Basket I contains $\{i_1, i_2,\ldots, i_k\}$

◆ **Rule** $\{i_1, i_2,\ldots ,i_k\} \to j$ 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** $i_1,\ldots,i_k$

  ➢ **Ratio of support for** $I \cup \{j\}$ **with support for** I

  ➢ Support for I: number of baskets containing I.

$$\text{conf}(I \to j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

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} → c**

➢ **Confidence: Ratio of support for I ∪ {j} with support for I**

➢ Ratio of support for {m,b} ∪ {c} to support for {m,b}

➢ Confidence = 2/4 = 50%

➢ **Want to identify association rules with high confidence.**

Interesting Association Rules

◆ **Not all high-confidence rules are interesting**

➢ The rule $X \rightarrow$ **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**

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

➢ **Interesting rules are those with high positive or negative interest values (usually above 0.5)**

➢ High **positive**/**negative** interest means presence of I **encourages** or **discourages** presence of j

➢ Example: {coke} -> pepsi should have high negative interest.

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} →c**
  ➢ Confidence: Ratio of support for I ∪ {j} with support for I
  ➢ **Confidence** = 2/4 = 0.5
  ➢ Interest: $\text{Interest}(I \rightarrow j) = \text{conf}(I \rightarrow j) - \Pr[j]$
  ➢ **Difference between its confidence and the fraction of baskets that contain j**
  ➢ **Interest** = |0.5 − 5/8| = 1/8
    • Pr [c]= Item c appears in **5/8** of the baskets
    • Rule is not very interesting!

Finding Useful 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"

◆ **Assume: not too many frequent itemsets or candidates for high support, high confidence association rules**

   ➢ Not so many that they can't be acted upon

   ➢ Adjust support threshold to avoid too many frequent itemsets.

Example: Find Association Rules with support ≥ *s* and confidence ≥ *c*

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

$B_3 = \{m, c, b, n\}$ $B_4 = \{c, j\}$

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

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

◆ **Support threshold *s = 3*, confidence *c = 0.75***

◆ **1) Frequent itemsets:**

➢ **{b} {c} {j} {m} {b,m} {b,c} {c,m} {c,j} {m,c,b}**

◆ **2) Generate rules:**

➢ **b→m**: *c*=?   **b→c**: *c*=?   **b,c→m**: *c*=?

➢ **m→b**: *c*=?   …   **b,m→c**: *c*=?

➢   **b→c,m**: *c*=?

$$\text{conf}(I \to j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

24

Example: Find Association Rules with support ≥ *s* and confidence ≥ *c*

B_1 = {m, c, b} B_2 = {m, p, j}

B_3 = {m, c, b, n} B_4 = {c, j}

B_5 = {m, p, b} B_6 = {m, c, b, j}

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

◆ **Support threshold *s = 3*, confidence *c = 0.75***

◆ **1) Frequent itemsets:**

➢ **{b} {c} {j} {m} {b,m} {b,c} {c,m} {c,j} {m,c,b}**

◆ **2) Generate rules:**

➢ ~~**b →m**: *c=4/6*~~   **b→c**: *c=5/6*   ~~**b,c →m**: *c=3/5*~~

➢ **m→b**: *c=4/5*   …   **b,m→c**: *c=3/4*

➢   ~~**b→c,m**: *c=3/6*~~

$$\text{conf}(I \to j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

Difficult part is identifying frequent itemsets: algorithms to find them are the focus of this chapter

25

FIND FREQUENT ITEMSETS

Computation Model

◆ Typically, market basket data are kept in **flat files** rather than in a database system

➢ Stored **on disk because they are very large files**

➢ Stored **basket-by-basket**

➢ **Goal: Expand baskets into pairs, triples, etc. as you read baskets**

- Use k nested loops to generate all sets of size k.

File Organization

Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Item
Etc.

Basket 1
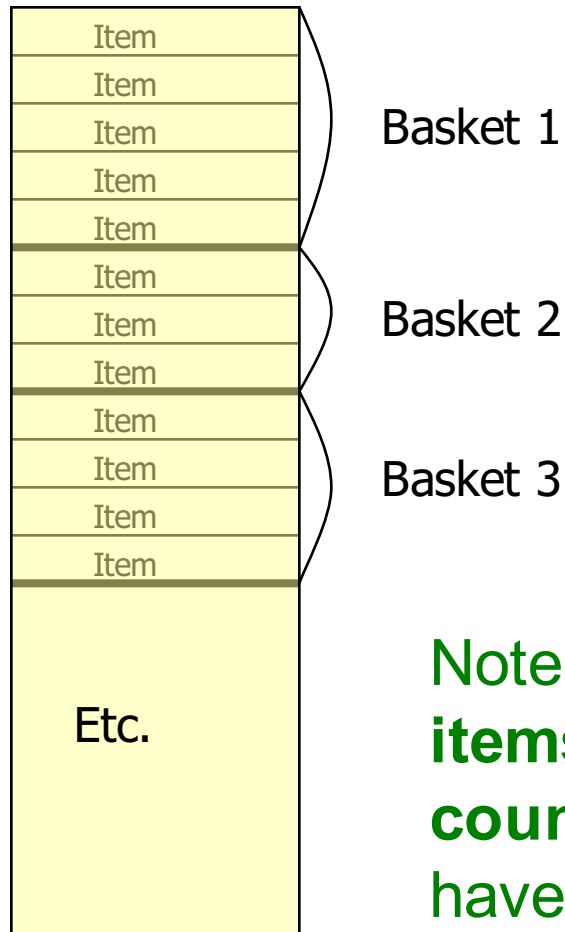
Basket 2

Basket 3

Example: items are positive integers, and boundaries between baskets are −1

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

28

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

> ➤ **Algorithms are designed so that counts can fit into main memory.**

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 itemsets.**

Naïve Algorithm

◆ **Read file once, counting in main memory the occurrences of each pair**

  ➢ Number of pairs in a basket of n items: n choose 2

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

  ➢ From each basket of n items, **generate its $n*(n-1)/2$ pairs using two nested loops,** add to the count for each pair

  ➢ **First basket: (a,b), (a,c), (a,y), (b,c), (b,y), (c,y)**

  ➢ **Second basket: (a,b), (a,x), (a,y), (a,z), (b,x), (b,y), (b,z), ...**

➢ **Total possible number of pairs in all baskets:**
  (#items)(#items -1)/2

◆ **Fails if (#items)^2 exceeds main memory**

  ➢ Remember: #items can be 100K (Wal-Mart) or 10B (Web pages).

32

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.

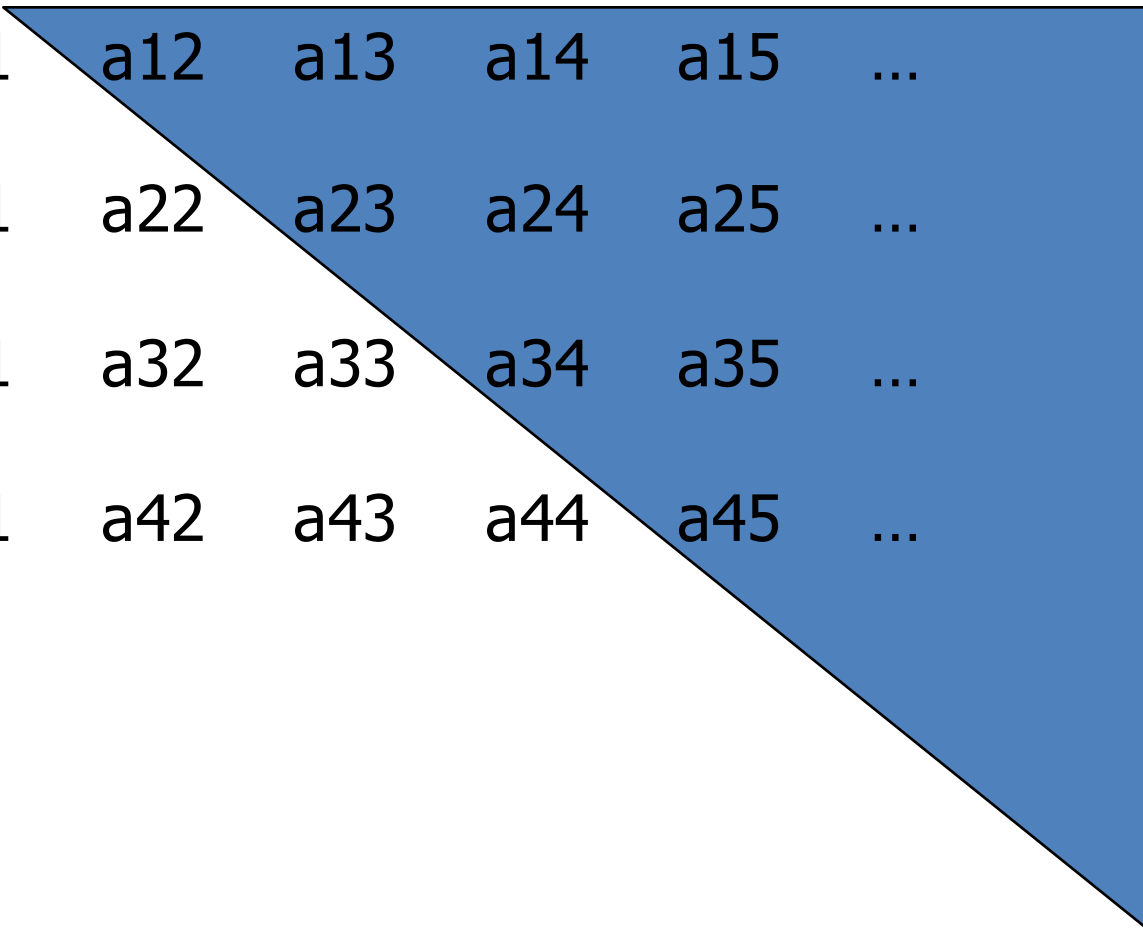# Details of Main-Memory Counting

◆ **Two approaches:**

➢ Method (1) **Count all pairs, using a triangular matrix**

➢ **Requires only 4 bytes/pair, but requires a count for each pair**

Note: assume integers are 4 bytes.

Triangular Matrix

A[i,j]   i<j

a11   a12   a13   a14   a15   …

a21   a22   a23   a24   a25   …

a31   a32   a33   a34   a35   …

a41   a42   a43   a44   a45   …

…

Details of Main-Memory Counting

◆ **Two approaches:**

➢ Method (1) **Count all pairs, using a triangular matrix**

➢ Method (2) **Keep a table of triples $[i, j, c]$ =** "**the count of the pair of items $\{i, j\}$ is c**"

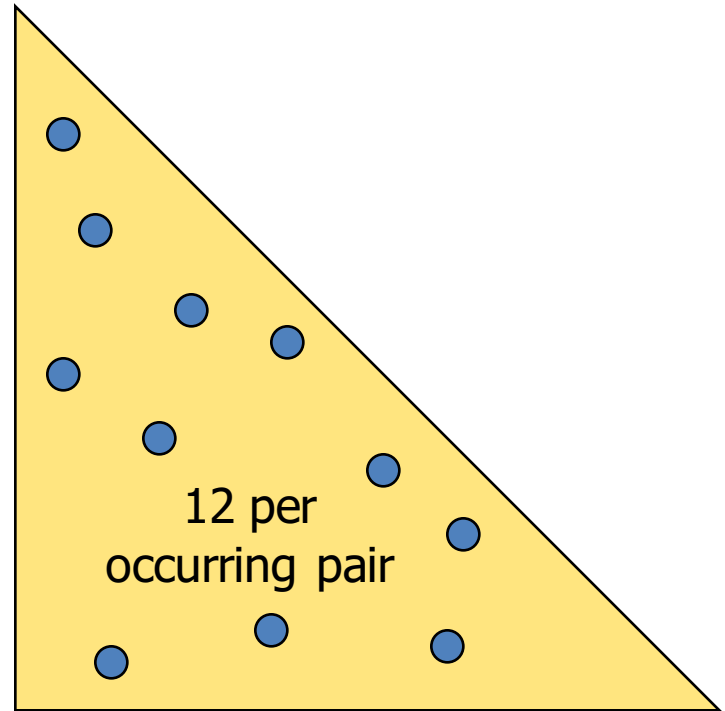**(1) requires only 4 bytes/pair, but requires a count for each pair**

Note: assume integers are 4 bytes

(2) requires 12 bytes, but only for those pairs with count > 0

Plus some additional overhead for a hashtable.

4 per pair

12 per
occurring pair
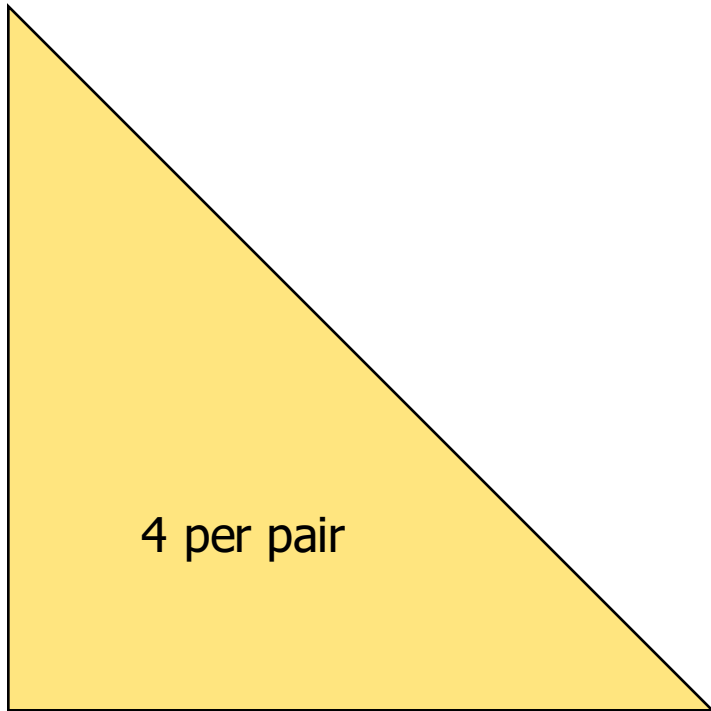
Method (1)
triangular matrix

Method (2)
table of triples [*i*, *j*, *c*]

Triangular-Matrix Approach – (1)

◆ **n** = total number of items

◆ Order each pair of items $\{i,j\}$ so that $i<j$

◆ 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 **$(i-1)(n-i/2)+j-i$**

 ➢ *Every time you see a pair {i,j} from a basket, increment the count at the corresponding position in triangular matrix*

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

◆ **Triangular Matrix** requires 4 bytes per pair.

Comparing the two approaches

◆ **Approach 1: Triangular Matrix**

  ➢ **n** = total number items

  ➢ **Count pair of items $\{i, j\}$ only if $i<j$**

  ➢ Keep pair counts in lexicographic order:

    • $\{1,2\}, \{1,3\},\dots, \{1,n\}, \{2,3\}, \{2,4\},\dots,\{2,n\}, \{3,4\},\dots$

  ➢ **Pair $\{i, j\}$ is at position $(i-1)(n-i/2)+j-i$**

  ➢ 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 occurring pair
(but only for pairs with count > 0)

  ➢ **Beats Approach 1 if fewer than 1/3 of possible pairs actually occur in the market basket data.**

Comparing the two approaches

◆ **Approach 1: Triangular Matrix**

  ➢ **n** = total number items

  ➢ Co~~unt pair~~ (i, j) ~~if~~

  ➢ ~~F~~

 ➢ ~~F~~

  ➢ ~~T~~

  ➢ ~~T~~

◆ **App**

  (bu

  ➢ B

  possible pairs actually occur

> **Problem is if we have
> too many items so the
> pairs
> do not fit into memory.**
>
> **Can we do better?**

A-Priori Algorithm

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 *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
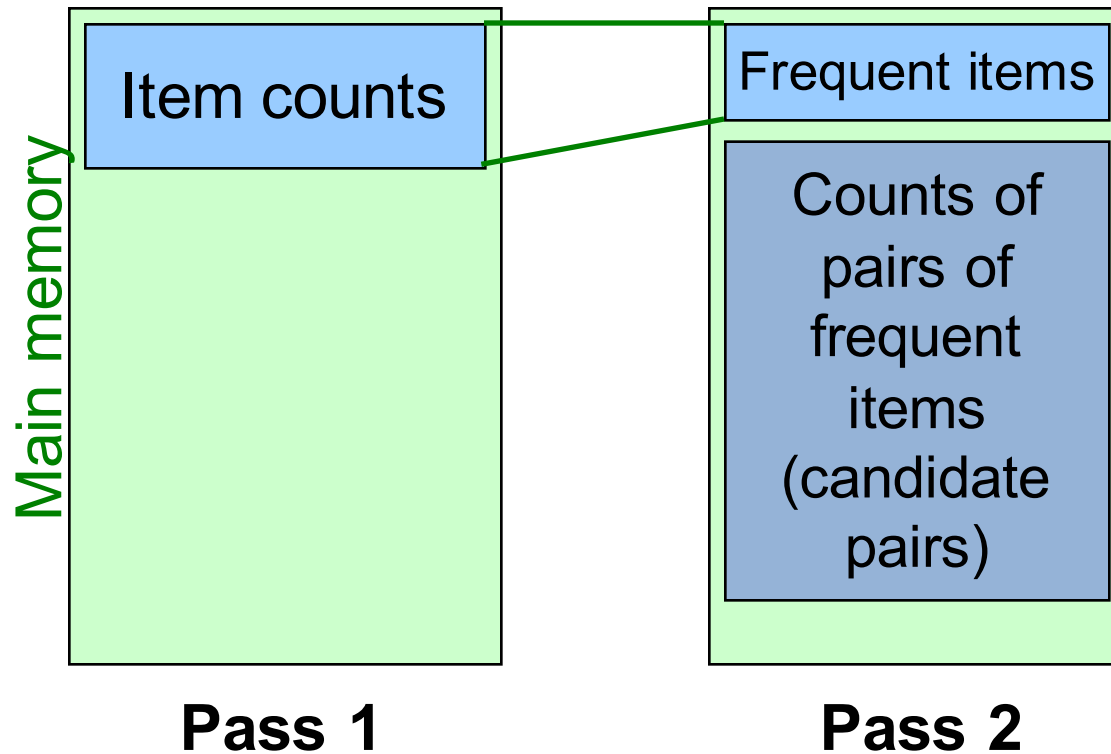
◆ **So, how does A-Priori find freq. pairs?**
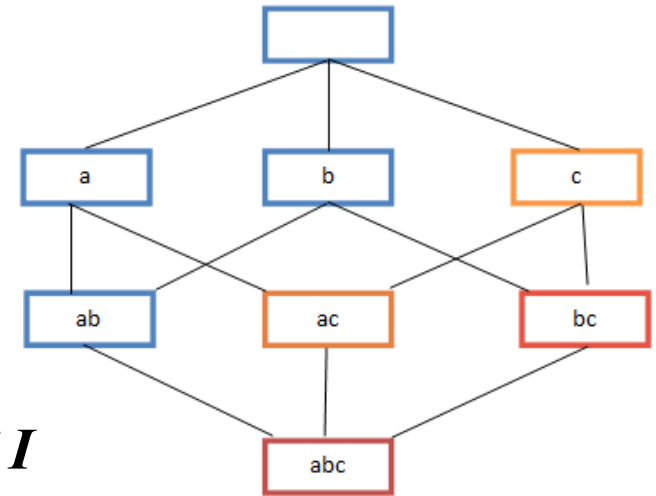
A-Priori Algorithm

◆ **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***

  ➢ **At the end of pass 1, after the complete input file has been processed, check the count for each item**

  ➢ **If count > s, then that item is frequent: saved for the next pass**

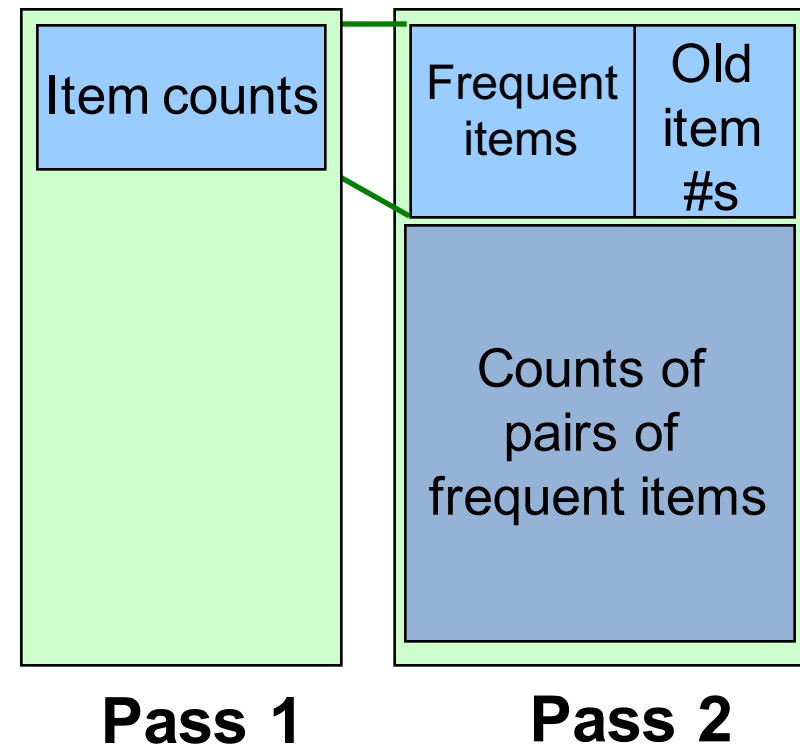◆ **Pass 1 identifies frequent itemsets of size 1.**

A-Priori Algorithm

◆ **Pass 2: Read baskets again and count in main memory only those pairs of items where both were found in Pass 1 to be frequent**

➤ **Requires:**

    ➤ **Memory proportional to square of *frequent* items only** (to hold counts of pairs)

    ➤ **List of the frequent items from the first pass** (so you know what must be counted)

◆ **Pairs of items that appear at least *s* times are the *frequent pairs***

    ➤ **At the end of pass 2, check the count for each pair**

    ➤ **If count > s, then that pair is frequent**

◆ **Pass 2 identifies frequent pairs: itemsets of size 2.** 44

Main-Memory: Picture of A-Priori

Pass 1: Item counts

Pass 2: Frequent items / Counts of pairs of frequent items (candidate pairs)

Main memory

Pass 1   **Pass 2**
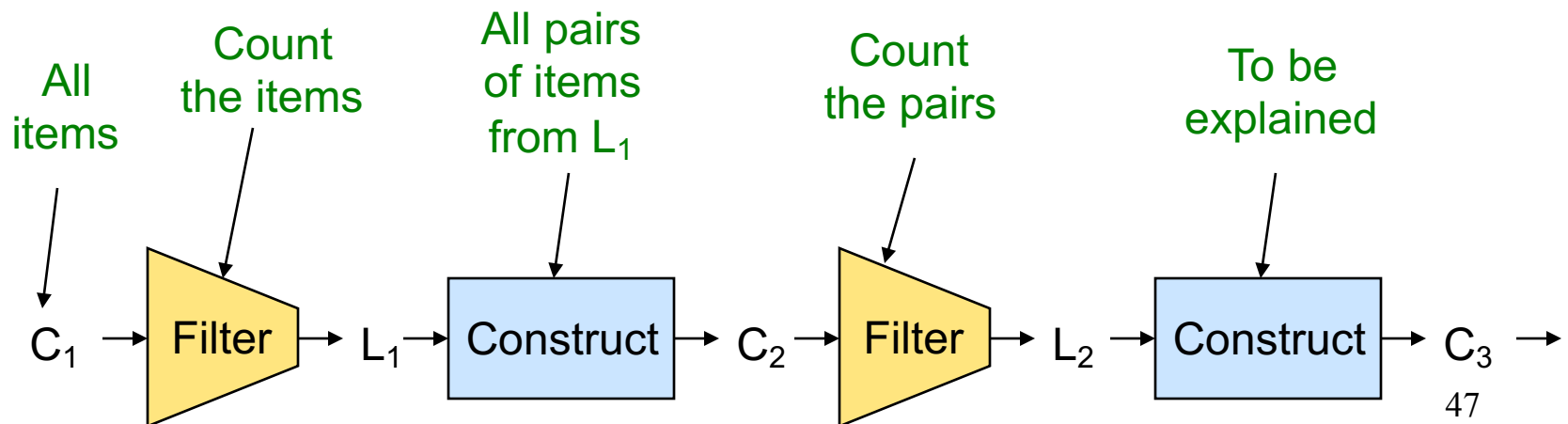
Detail for A-Priori

◆ **You can use the triangular matrix method with n = number of frequent items**

  ➢ May save space compared with storing triples

◆ **Trick: re-number frequent items 1,2,…** and keep a table relating new numbers to original item numbers.

Item counts

Frequent items	Old item #s
Counts of pairs of frequent items	

Pass 1   **Pass 2**

What About Larger Frequent Itemsets? Frequent Triples, Etc.

◆ **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**

All items → Count the items → Filter → L_1 → All pairs of items from L_1 → Construct → C_2 → Count the pairs → Filter → L_2 → To be explained → Construct → C_3 →

C_1 → Filter → L_1 → Construct → C_2 → Filter → L_2 → Construct → C_3 →

Recall: Example

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

$B_3 = \{m, c, b, n\}$ $B_4 = \{c, j\}$

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

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

◆ **Frequent itemsets (s=3):**

➢ $\{b\}$, $\{c\}$, $\{j\}$, $\{m\}$

➢ $\{b,m\}$ $\{b,c\}$ $\{c,m\}$ $\{c,j\}$

➢ $\{m,c,b\}$

Example (cont.)

$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$
$B_3 = \{m, c, b, n\}$ $B_4 = \{c, j\}$
$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$
$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

Frequent itemsets (s=3):
- $\{b\}, \{c\}, \{j\}, \{m\}$
- $\{b,m\}$ $\{b,c\}$ $\{c,m\}$ $\{c,j\}$
- $\{m,c,b\}$

Hypothetical steps of the A-Priori algorithm

➢ $C_1 = \{ \{b\} \{c\} \{j\} \{m\} \{n\} \{p\} \}$:
  - all candidate items

➢ Count the support of itemsets in C_1

➢ Prune non-frequent: $L_1 = \{ b, c, j, m \}$

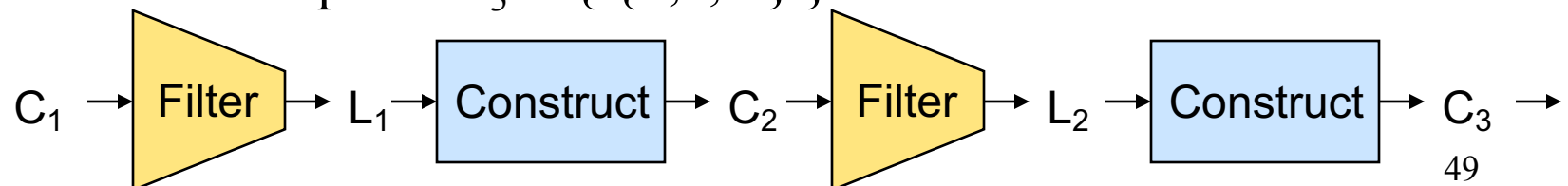➢ Generate $C_2 = \{ \{b,c\} \{b,j\} \{b,m\} \{c,j\} \{c,m\} \{j,m\} \}$

➢ Count the support of itemsets in C_2

➢ Prune non-frequent: $L_2 = \{ \{b,m\} \{b,c\} \{c,m\} \{c,j\} \}$

➢ Generate $C_3 = \{ \{b,c,m\} \}$ Why not $\{b,m,j\}$? $\{m,j\}$ is not frequent

➢ Count the support of itemsets in C_3

➢ Prune non-frequent: $L_3 = \{ \{b,c,m\} \}$

$C_1 \rightarrow$ Filter $\rightarrow L_1 \rightarrow$ Construct $\rightarrow C_2 \rightarrow$ Filter $\rightarrow L_2 \rightarrow$ Construct $\rightarrow C_3 \rightarrow$

Example (cont.)

Hypothetical steps of the A-Priori algorithm

➤ C_1 = { {b} {c} {j} {m} {n} {p} }: all candidate items

➤ Count the support of itemsets in C_1

➤ Prune non-frequent: 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: L_2 = { {b,m} {b,c} {c,m} {c,j} }

➤ **Generate C_3 = { {b,c,m} }**

➤ **Count the support of itemsets in C_3**

➤ **Prune non-frequent: L_3 = { {b,c,m} }**

◆ Note: here we generate new candidates by generating C_k from L_{k-1} and L_1.
 But that one can be more careful with candidate generation. For example, in C_3 we know **{b,m,j} cannot be frequent** since **{m,j} is not frequent**

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.

Summary

◆ *Market-basket model:* A large set of **items,** A large set of **baskets**, Each basket is a small subset of items

> ➤ **Goal : to discover Association Rules**

◆ An *association rule* has two parts, an antecedent (if) and a consequent (then)-> if/then patterns

> ➤ An antecedent is an item found in the basket. A consequent is an item that is found in combination with the antecedent.

◆ *Frequent itemsets:* Find sets of items that appear "frequently" in the baskets

> ➤ *Support* is an indication of how frequently the items appear in the baskets,
>
> ➤ *Confidence* indicates the number of times the if/then statements have been found to be true,
>
> ➤ **Interest**: Positive or negative,

◆ *Finding Frequent Itemsets:* triangular matrix, table of triples, A-Priori Algorithm, and variations (Next lecture).