KTH ROYAL INSTITUTE
OF TECHNOLOGY

Frequent Itemsets

Vladimir Vlassov

Acknowledgement: Most of the slides are adopted from the slide provided at http://www.mmds.org for the book "Mining of Massive Datasets" by Jure Leskovec, Anand Rajaraman, Jeff Ullman, Stanford University

Outline

- Introduction
- The ”Market-basket”model of the data
- Define:
  - Frequent itemsets: support
  - Association rules: confidence, interestingness
- Algorithms for finding frequent itemsets
  - Finding frequent pairs
  - A-Priori algorithm
  - (Self-study) PCY algorithm + 2 refinements

The Discovery of Frequent Itemsets

Goal: Identify ***items*** that are frequently occurs together in sets of items (***baskets***) – the ***frequent itemsets***

- E.g., identify products (itemsets) bought together by sufficiently many customers
- Usually, the number of items in a basket is much smaller than the total number of items
- Usually, the number of baskets is very large
  - cannot fit in the main memory

Association Rule Discovery

The problem of frequent itemsets is often viewed as ***the discovery of "association rules"***

- "if then" rules in the form of implication $X \rightarrow Y$ (if X then Y)
- where X and Y are itemsets (Y can be just an item)

For example: If ***butter*** and ***bread*** are bought together, customers also buy ***milk***

$$\{butter, bread\} \rightarrow \{milk\}$$

- To be statistically significant, the rule needs a support of many (several hundred) transactions.

We need to count the number of occurrences of a given itemset in all trasactions (baskets)

Mining Association Rules

Rakesh Agrawal, Tomasz Imieliński, Arun Swam,
”*Mining association rules between sets of items in large databases*”,
SIGMOD '93

Rakesh Agrawal

Follow

Technical Fellow, Microsoft Research
Data Mining, Web Search, Education, Privacy
Verified email at microsoft.com

Title 1–20	Cited by	Year
Fast algorithms for mining association rules R Agrawal, R Srikant Proc. 20th int. conf. very large data bases, VLDB 1215, 487-499	20664	1994
Mining association rules between sets of items in large databases R Agrawal, T Imieliński, A Swami Acm sigmod record 22 (2), 207-216	17943	1993

Google Scholar

Citation indices	All	Since 2011
Citations	99824	36606
h-index	96	59
i10-index	226	157

Association Rule Discovery

Supermarket shelf management:

Goal: 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 a customer buys diaper and milk, then (s)he is likely to buy beer
- Don’t be surprised if you find six-packs next to diapers!

The Market-Basket Model

A large set of ***items*** $I = \{i_1, i_2, \ldots, i_m\}$
- e.g., things sold in a supermarket

A large set of ***baskets*** (transactions)
Each basket is a small subset of items $t \subset I$
- e.g., the things a customer bought in one trip to a shop

Transaction/basket database $T = \{t_1, t_2, \ldots, t_n\}$.

Want to discover association rules:
- People who bought $\{x, y, z\}$ tend to buy $\{v, w\}$
- Amazon! (as well as Netflix, Spotify, …)

The Market-Basket Model

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 }

Applications – Market Baskets

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

Real market baskets: Chain stores 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;
  - run sale on hot-dogs and raise the price of mustard
- Need the rule to occur frequently, or no $$'s

Amazon's customers who bought *X* also bought *Y*

Applications – Plagiarism

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 – Drugs with Side-Effects

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 – Related Concepts

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.

Association Rule Discovery: Approach

Given a set of baskets, discover association rules

- People who bought {a, b, c} tend to buy {d, e}

2-step approach

1. Find frequent itemsets
2. Generate association rules

Some Definitions

An ***itemset X*** is a subset of items ***I: X*** $\subset$ ***I***

E.g., ***X* = {milk, bread, cereal}** is an itemset.

A ***k-itemset*** is an itemset with ***k*** items.

E.g., **{milk, bread, cereal}** is a 3-itemset

An ***association rule*** is an implication of the form:

X $\rightarrow$ ***Y***, where ***X***, ***Y*** $\subset$ ***I***, and ***X*** $\cap$ ***Y* =** $\varnothing$

Frequent Itemsets. Support

Simplest question: Find sets of items that appear together "frequently" in baskets

Support for itemset ***I*** is the number of baskets containing all items in ***I***

- Often expressed as a fraction of the total number of baskets

Given a **support threshold** ***s***, sets of items that appear in at least ***s*** baskets are called ***frequent itemsets***

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

Example: Frequent Itemsets

Items = {milk, coke, pepsi, beer, juice}
Support threshold = 3 baskets

BID	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 (at least in 3 baskets):

singletons	{ m }, { c }, { b }, { j } 5 5 6 4
doubletons	{m, b}, {c, b}, {c, j} 4 4 3
triples	{m, c, b} 2

Important Observations

- The support of the itemset $J \subset I$ is at least as big as the support of the itemset I,
  i.e., the support of the itemset J is at least as big as its superset I
- For the itemset I to be frequent all subsets of I must be frequent

Association Rules

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

Confidence of the rule $I \rightarrow j$ is the ratio of the support for $I \cup \{j\}$ to the support for I.

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

That is the confidence of the rule $I \rightarrow j$ is the fraction of the baskets with all of I that also contain j.

Example: Confidence

Let's take a frequent doubleton **{m, b}**

- Support of **{m, b})** is 4

The confidence of **{m, b}** → **c** is

conf({m, b} → c) = 2/4 = 0,5 (50%)

- The itemset **{m, b}** appears in 4 baskets, B1, B3, B5 and B6.
- Of theses, **c** appears in B1 and B6, or 2/4 of the baskets.

BID	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}

Interesting Association Rules

In practice there are many rules, want to find significant/interesting ones!

- The rule $X \rightarrow$ ***milk*** may have high confidence for many itemsets ***X***, because milk is just purchased very often (independent of ***X***) and the confidence will be high

Interest of an association rule $I \rightarrow j$ is the 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)***

Example: Confidence and Interest

Association rule: {m, b} →c

- **Confidence =** 2/4 = 0.5
- **Interest =** |0.5 – 5/8| = |4/8 – 5/8| = 1/8 ≈ 0
  - *Rule is not very interesting!*
  - Item ***c*** appears in 5/8 of the baskets
- The probability of **c** given **{m, b}** (that is the confidence) is 50%, but **c** appears in baskets rather often, about 60% (with almost the same probability)
  - thus **{m, b}** has no (little) influence on **c**

BID	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}

Example: Confidence and Interest

{diapers} → beer

interest({diapers} → beer) =
(support({diapers,beer})/support({diapers})) –(support({beer})/num_baskets)

The fraction of diaper-buyers that buy beer is significantly greater than the fraction of all customers that buy beer. *The rule is of high interests*

{coke} → pepsi

interest({coke} → pepsi) = (support({coke,pepsi})/support({coke})) –
(support({pepsi})/num_baskets)

Negative interest – people who buy coke are unlikely to also buy pepsi

Finding Association Rules

Goal: Find all association rules with support $\geq s$ (at least s) and confidence $\geq c$ (at least 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 $I \rightarrow j$ has reasonably high support and confidence, then both I and $I \cup \{j\}$ will be "frequent"

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

Finding Association Rules (cont)

We are looking for rules $I \rightarrow j$ with reasonably high support and confidence

- both I and $I \cup \{j\}$ will have fairly high support, i.e. should be "frequent"

For brick-and-mortar marketing:

- support of 1% is reasonably high
- confidence of 50% is adequate, otherwise rule has little practical effect

Challenges

- Completeness: find all rules.
- Mining with data on hard disk (not in memory)

Mining Association Rules

Step 1: Find all frequent itemsets ***I*** with at least as a given support

Step 2: **Rule generation**

- For every subset ***A*** of ***I***, generate a rule $A \rightarrow I \setminus A$
  - Since ***I*** is frequent, then so is ***A***
  - **Variant 1:** Single pass to compute the rule confidence
    - conf(***A,B*** → ***C,D***) = supp(**A,B,C,D**)/supp(**A,B**)
  - **Variant 2:**
    - **Observation:** If **A,B,C** → **D** is below confidence, so is **A,B** → **C,D**
      because of supp(**A,B**) ≥ supp(**A,B,C**)
    - Can generate bigger rules from smaller ones
- **Output the rules above the confidence threshold**

Example

generate a rule $\{I,j\} \rightarrow \{I,j\} \setminus I$
conf(I→j) = supp(I,j)/supp(I)

BID	Baskets
B1	{m, c, b}
B2	{m, p, j}
B3	{m, c, b, n}
B4	{c, j}
B5	{m, p, b}
B6	{m, c, b, j}
B7	{c, b, j}
B8	{b, c}

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

1) Frequent itemsets:

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

2) Generate rules:

{b,m}	{b,c}	...	{m,c,b}
~~**b** → **m**: c = 4/6~~	**b** → **c**: c = 5/6	...	b,m → **c**: c = 3/4
m → **b**: c = 4/5	**c** → **b**: c = 5/6		~~b,c → **m**: c = 3/5~~
			~~b → **c,m**: c = 3/6~~
			...

If K,L,M→N is below confidence, so is K,L→M,N

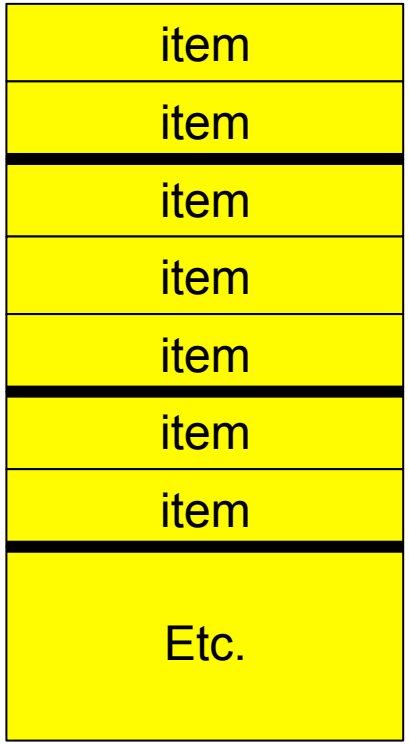
***Itemsets: Computation Model

Back to finding frequent itemsets

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.

Computation Model

- Cost of mining disk-resident data is the ***number of disk I/Os***
- In practice, association-rule algorithms read data in ***passes***
  - all baskets read in turn
- We measure the cost by the ***number of passes*** over the data

- **Main-Memory Bottleneck**
  - 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

Finding frequent pairs is the hardest problem

- Frequent pairs are common, frequent triples are rare
- The probability of being frequent drops exponentially with size; number of sets grows more slowly with size

Let's first concentrate on pairs…

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

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

- 100K (Walmart) or 10B (Web pages)
  - Suppose 10^5 items at Walmart; counts are 4B integers
  - Number of pairs of items: $10^5(10^5-1)/2 = 5*10^9$
  - Therefore, $2*10^{10}$B = 20GB of memory needed

Two Approaches to Counting Pairs

Assume item numbers and counters are integers of 4B

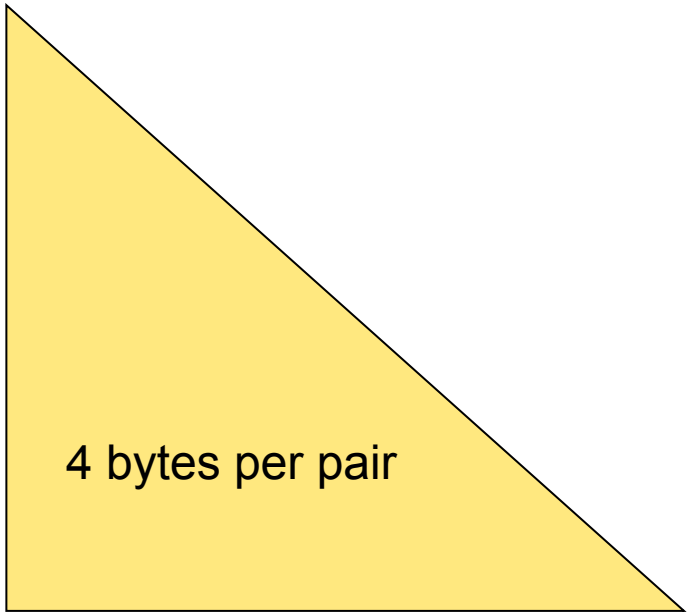
Approach 1: Count all pairs using a matrix

- 4B per pair

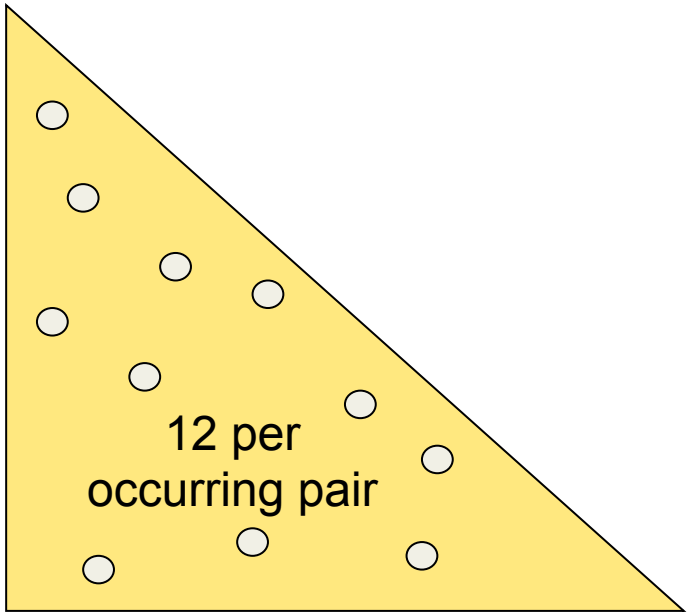
Approach 2: Keep a table of triples **[*i*, *j*, *c*]** = "the count of the pair of items **{*i*, *j*}** is ***c***."

- 12B for pairs with count > 0
- Plus some additional overhead for the hashtable

Comparing the 2 Approaches



Triangular Matrix



Triples

Comparing the 2 Approaches (cont)

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},…, {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**; memory demand = $\mathbf{2n^2}$ **B**
- **Triangular Matrix** requires 4B per pair

	1,2	1,3	1,4	1,5
		2,3	2,4	2,5
			3,4	3,5
				4,5

Comparing the 2 Approaches (cont)

Approach 2: Triples

- [i, j, count] – 3 integers – **12B** per occurring pair
  (but only for pairs with count > 0)
- Beats triangular matrix if at most 1/3 of possible pairs actually occur, because it uses 3 times more memory per pair then matrix
- May require extra space for retrieval structure, e.g., a hash table h(i,j) → count

Can we do better?

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

A-Priori Algorithm – (1)

A two-pass approach called ***A-Priori*** limits the memory demand.

Key idea: ***monotonicity of support***

- If a set of items appears at least ***s*** times, so does every subset, i.e., the support of a subset is at least as big as the support of its superset

The **downward closure property** of frequent patterns

- Any subset of a frequent itemset must be frequent

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)

Based on candidate generation-and-test approach

A-priori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested, because it's also infrequent

[Agrawal & Srikant,@VLDB'94, Mannila, et al. @ KDD'94]

A-Priori Algorithm – (2)

Pass 1: Read baskets and count in main memory the occurrences of each **individual item**

- Requires O(n) memory, where n is #items

Items that appear ≥s times are the *frequent items*

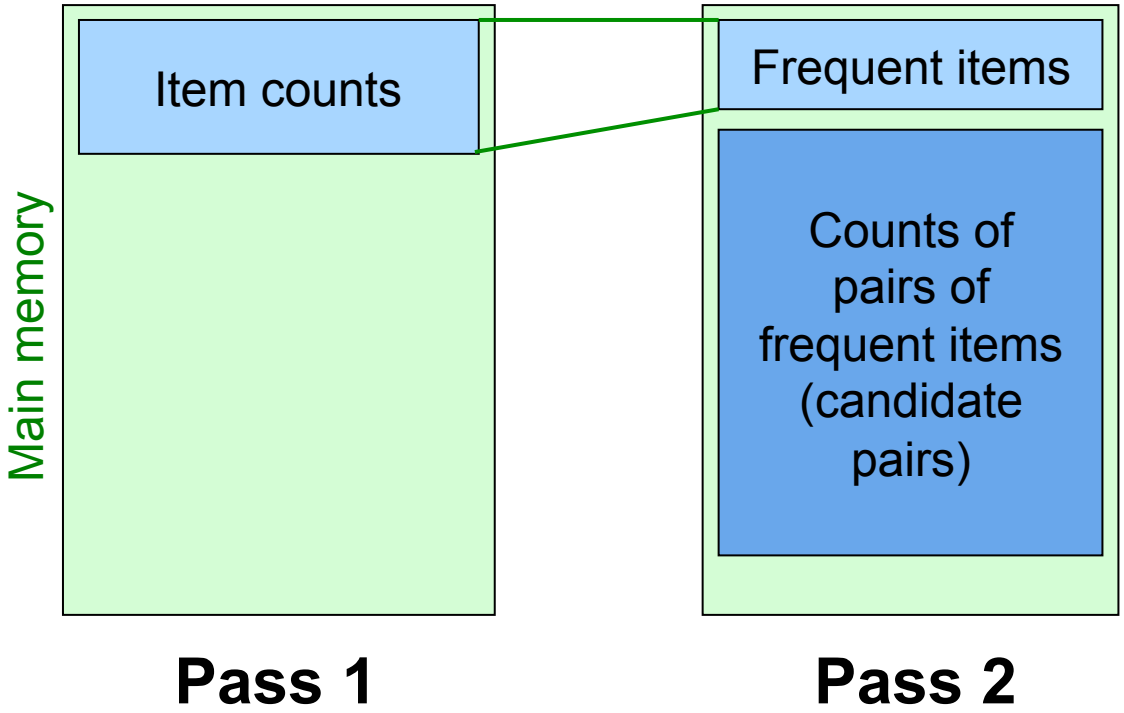
Typical s=1% as many singletons will be infrequent

(***s*** is the support threshold)

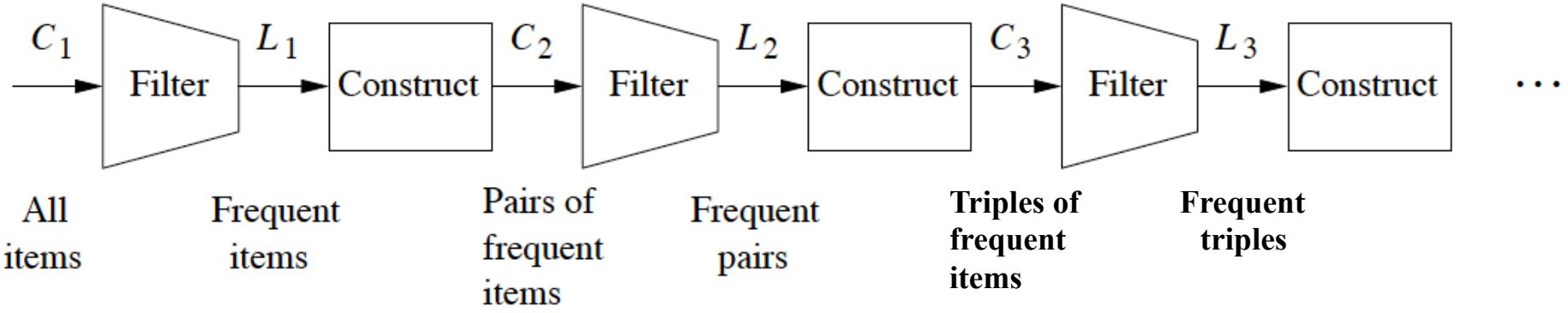
Pass 2: Read baskets again and count only those pairs where both elements are frequent (discovered in Pass 1)

- Requires memory proportional to square of **frequent** items only (for counts) – 2^m instead 2^n
- Plus a list of the frequent items (so you know what must be counted)

Main-Memory: Picture of A-Priori



The Pipeline of the A-Priory Algorithm



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 $\geq$ **s**) based on information from the pass for ***k*–1**
- L_k = the set of truly frequent ***k***-tuples, i.e. filter only those ***k***-tuples from C_k that have support at least ***s***

Example: Steps of the A-Priori Algorithm

- C_1 = { {b} {c} {j} {m} {n} {p} } /* singletons */
- Count the support of itemsets in C_1
- Prune (filter out) 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,m,c} {b,c,j} {b,m,j} {c,m,j} }
- Count the support of itemsets in C_3
- Prune non-frequent: L_3 = { {b,c,m} }

Note: Candidates in $\boldsymbol{C_k}$ can be generated by combining itemsets from $\boldsymbol{L_{k-1}}$ and singletons from $\boldsymbol{L_1}$.

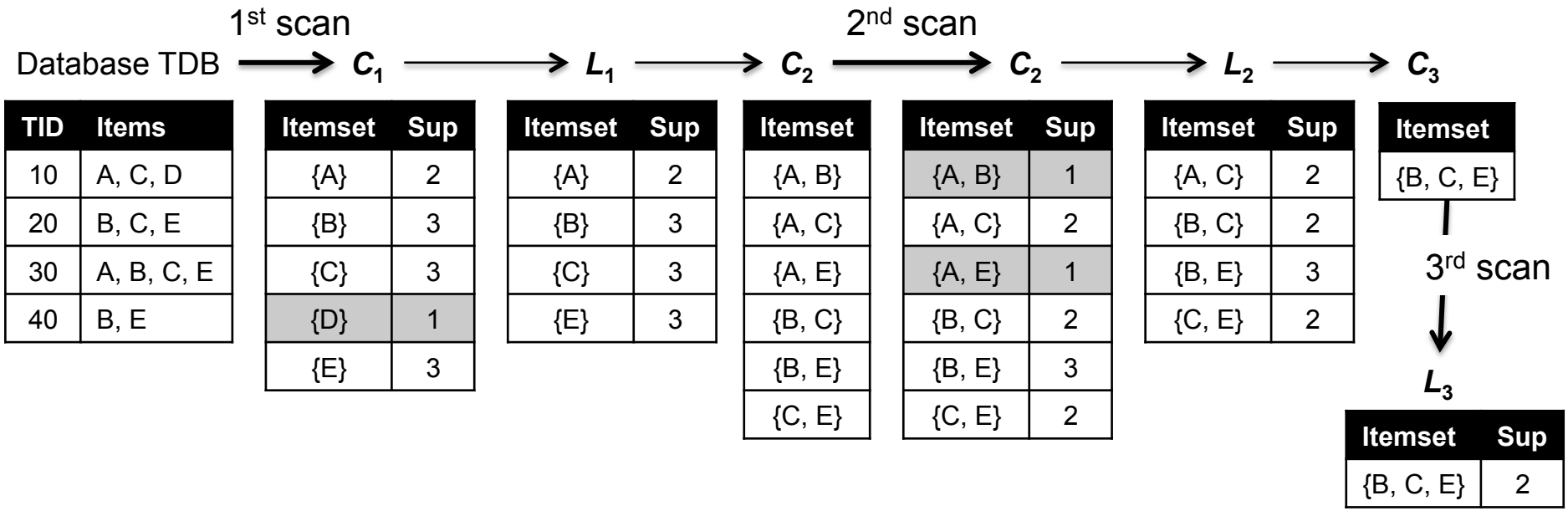
Generating Candidates

Candidates in C_k can be generated by combining itemsets from L_{k-1} and singletons from L_1.

One should be careful and selective with candidate generation: for a candidate in C_k to be a frequent itemset, *all its subsets* must be frequent, not only the itemsets from L_{k-1} and L_1 that the candidate is constructed from, i.e., each of its subsets should be in the corresponding L_m, $m = 1,\ldots, k-1$

For example, in C_3, {b,m,j} cannot be frequent since {m,j} is not frequent, therefore {b,m,j} should not be even included in C_3

The Apriori Algorithm – An Example (s = 2)



TID	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

Itemset	Sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

Itemset	Sup
{A}	2
{B}	3
{C}	3
{E}	3

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

Itemset	Sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

Itemset	Sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

Itemset
{B, C, E}

Itemset	Sup
{B, C, E}	2

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

A-Priori – 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

(Self-study) PCY (Park-Chen-Yu) Algorithm

PCY reduce the number of candidates pairs and hence improves memory usage

- In pass 1, there is a lot of memory left, leverage that to help with pass 2
- Maintain a hash table with as many buckets as fit in memory
- Keep count for each bucket into which pairs of items are hashed
- Just the count, not the pairs!

Multistage improves PCY by using several successive hash tables to reduce further the number of candidate pairs

(Self-study) Frequent Itemsets in < 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)***