Frequent Item Sets

Chau Tran & Chun-Che Wang

Outline

1. Definitions
    - Frequent Itemsets
    - Association rules
2. Apriori Algorithm

Frequent Itemsets

What? Why? How?

Motivation 1: Amazon suggestions

Amazon suggestions (German version)

Motivation 2: Plagiarism detector

- Given a set of documents (eg. homework handin)
  - Find the documents that are similar

Motivation 3: Biomarker

- Given the set of medical data
  - For each patient, we have his/her genes, blood proteins, diseases
  - Find patterns
    - which genes/proteins cause which diseases

What do they have in common?

- A large set of items
  - things sold on Amazon
  - set of documents
  - genes or blood proteins or diseases
- A large set of baskets
  - shopping carts/orders on Amazon
  - set of sentences
  - medical data for multiple of patients

Goal

- Find a general many-many mapping between two set of items
  - {Kitkat} ⇒ {Reese, Twix}
  - {Document 1} ⇒ {Document 2, Document 3}
  - {Gene A, Protein B} ⇒ {Disease C}

Approach

- A = {A1, A2,..., Am}
- B = {B1, B2,..., Bn}

A, B are subset of
I = set of items

$$P(B|A) = \frac{P(A,B)}{P(A)}$$

$$= \frac{Count(A,B)}{Count(A)}$$

Definitions

- Support for itemset A: Number of baskets containing all items in A
  - Same as Count(A)
- Given a support threshold s, the set of items that appear in at least s baskets are called frequent itemsets

Example: Frequent Itemsets

- Items = {milk, coke, pepsi, beer, juice}

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 for support threshold = 3:
  - {m}, {c}, {b}, {j}, {m,b}, {b,c}, {c,j}

Association Rules

- $A \Rightarrow B$ means: “if a basket contains items in A, it is likely to contain items in B”
- There are exponentially many rules, we want to find significant/interesting ones
- Confidence of an association rule:
  - $\mathrm{Conf}(A \Rightarrow B) = P(B \mid A)$

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), and the confidence will be high
- Interest of an association rule:
  - $\text{Interest}(A \Rightarrow B) = \text{Conf}(A \Rightarrow B) - P(B)$

$$= P(B \mid A) - P(B)$$

- $\text{Interest}(A \Rightarrow B) = P(B \mid A) - P(B)$
  - > 0 if $P(B \mid A) > P(B)$
  - $= 0$ if $P(B \mid A) = P(B)$
  - < 0 if $P(B \mid A) < P(B)$

Example: Confidence and Interest

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: $\{m,b\} \Rightarrow c$
  - Confidence = 2/4 = 0.5
  - Interest = $0.5 - \frac{5}{8} = -\frac{1}{8}$
    - High confidence but not very interesting

Overview of Algorithm

- Step 1: Find all frequent itemsets I
- Step 2: Rule generation
  - For every subset A of I, generate a rule $A \Rightarrow I \setminus A$
    - Since I is frequent, A is also frequent
  - Output the rules above the confidence threshold

Example: Finding association rules

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}

- Min support s=3, confidence c=0.75
- 1) Frequent itemsets:
  - {b,m} {b,c} {c,n} {c,j} {m,c,b}
- 2) Generate rules:
  - $\cancel{b \Rightarrow m = 4/6}$ $b \Rightarrow c = 5/6$ $b,m \Rightarrow c = 3/4$
  - $m \Rightarrow b = 4/5$ … $\cancel{b,c \Rightarrow m = 3/5}$

…

How to find frequent itemsets?

- Have to find subsets A such that Support(A) > s
  - There are 2^n subsets
  - Can’t be stored in memory

How to find frequent itemsets?

- Solution: only find subsets of size 2

Really?

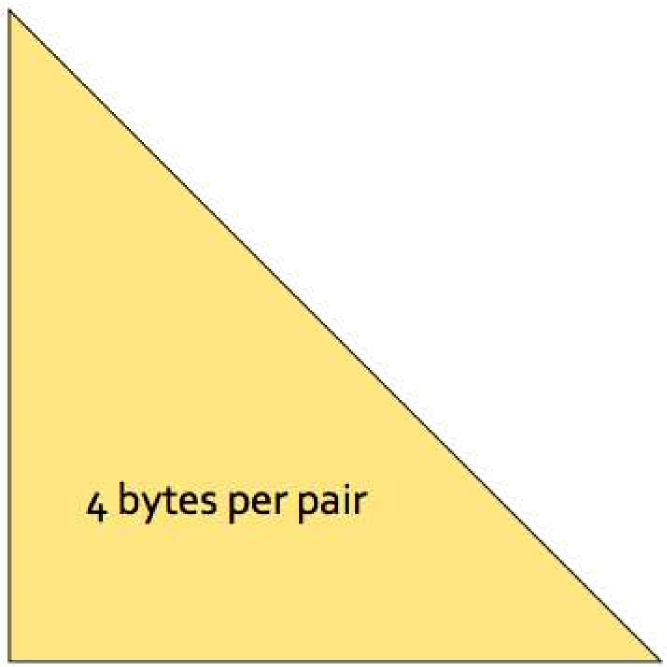
- Frequent pairs are common, frequent triples are rare, don't even talk about n=4
- Let's first concentrate on pairs, then extend to larger sets (wink at Chun)
- The approach
    - Find Support(A) for all A such that |A| = 2

Naive Algorithm

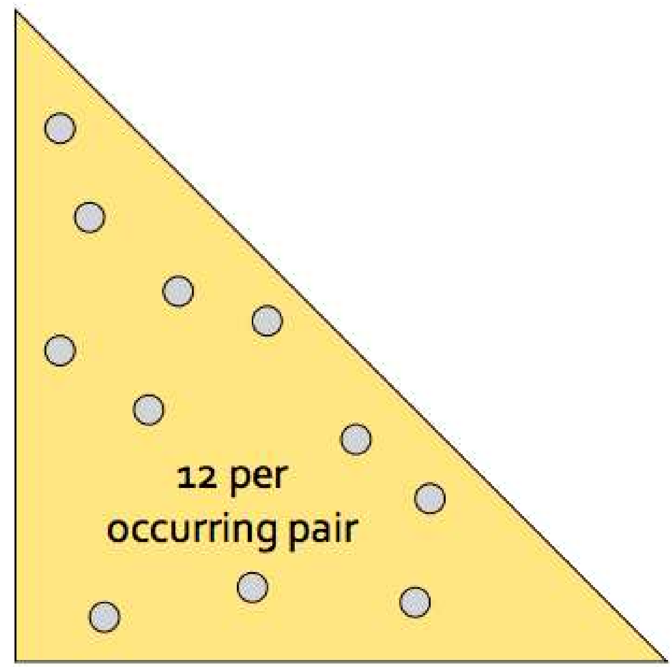
- For each basket b:
  - for each pair (i1,i2) in b:
    - increment count of (b1,b2)
- Still fail if (#items)^2 exceeds main memory
  - Walmart has 10^5 items
  - Counts are 4-byte integers
  - Number of pairs = 10^5*(10^5-1) /2 = 5 * 10^9
  - 2 * 10^10 bytes (20 GB) of memory needed

Not all pairs are equal

- Store a hash table
  - (i1, i2) => index
- Store triples [i1, i2, c(i1,i2)]
  - uses 12 bytes per pair
  - but only for pairs with count > 0
- Better if less than ⅓ of possible pairs actually occur



Triangular Matrix



Triples

Summary

- What?
  - Given a large set of baskets of items, find items that are correlated
- Why?
- How?
  - Find frequent itemsets
    - subsets that occur more than s times
  - Find association rules
    - $\text{Conf}(A \Rightarrow B) = \text{Support}(A,B) / \text{Support}(A)$

A-Priori Algorithm

Naive Algorithm Revisited

- Pros:
  - Read the entire file (transaction DB) once
- Cons
  - Fail if (#items)^2 exceeds main memory

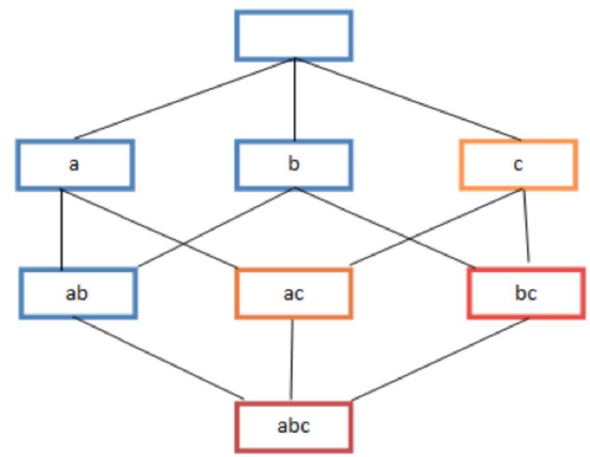
A-Priori Algorithm

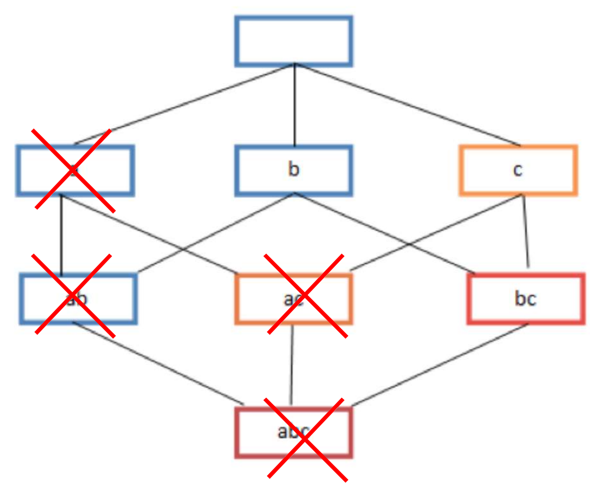
- Designed to reduce the number of pairs that need to be counted
- How?

  hint: There is no such thing as a free lunch
- Perform 2 passes over data

A-Priori Algorithm

- **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

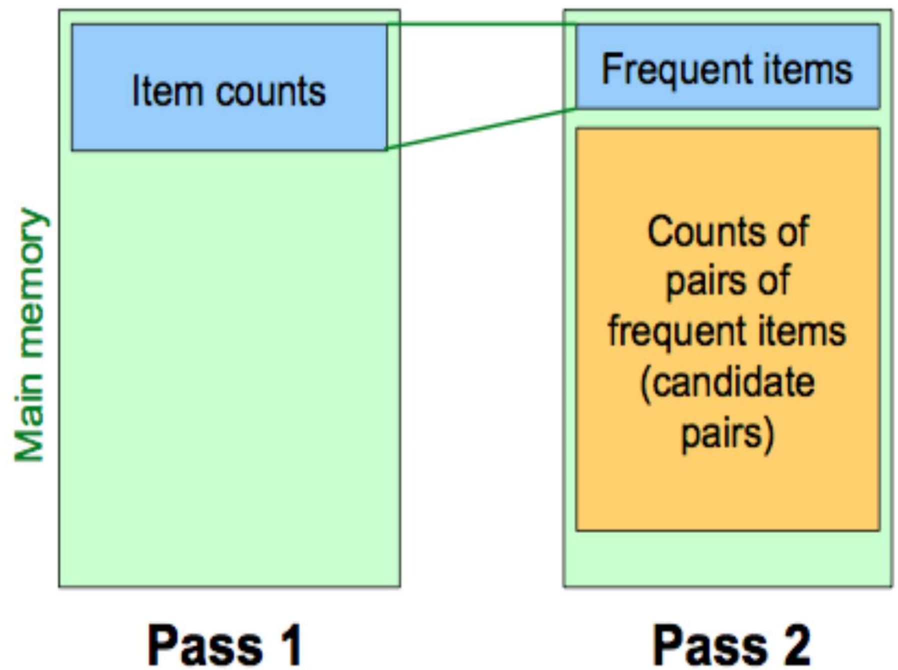




A-Priori Algorithm

- Pass 1:
  - Count the occurrences of each **individual item**
  - items that appear at least s time are the frequent items
- Pass 2:
  - Read baskets again and count in only those pairs where both elements are frequent (from pass 1)

A-Priori Algorithm

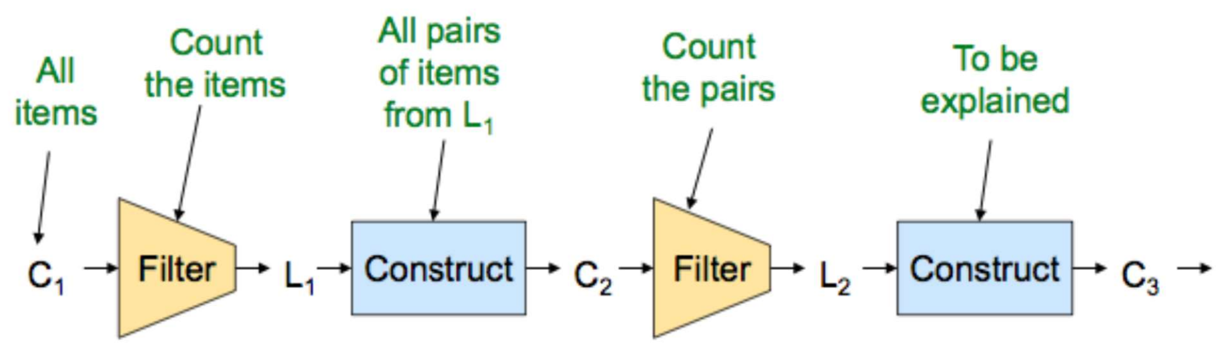


Frequent Tripes, Etc.

For each k, we construct two sets of *k-tuples*

C_k Candidate k-tuples = those might be frequent sets (support > s)

L_k The set of truly frequent k-tuples



Example

- **Hypothetical steps of the A-Priori algorithm**
  - C_1 = { {b} {c} {j} {m} {n} {p} }
  - 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} {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} }

A-priori for All Frequent Itemsets

- For finding frequent k-tuple: Scan entire data k times
- Needs room in main memory to count each candidate k-tuple
- Typical, k = 2 requires the most memory

What else can we improve?

- Observation

In pass 1 of a-priori, most memory is idle !

Can we use the idle memory to reduce memory required in pass 2?