Frequent itemsets

Tiphaine Viard

2020 – 2021

What is the frequent itemset problem?

The market-basket model

A large set of **items**, a large set of **baskets**

A basket is a subset of the item set

items = apple, banana, cranberry, durian, elderberry

b_1 = apple, banana, durian b_2 = apple, cranberry, elderberry

b_3 = apple, durian b_4 = banana, elderberry

b_5 = apple, cranberry, elderberry

b_6 = apple, banana, durian, elderberry

b_7 = banana, durian, elderberry b_8 = banana, durian

What does **frequent** mean?

What is the frequent itemset problem?

The problem

Minimum number of baskets with item: **support** $= 3$ ($\in \mathbb{N}$)

items = apple, banana, cranberry, durian, elderberry

b_1 = apple, banana, durian b_2 = apple, cranberry, elderberry

b_3 = apple, durian b_4 = banana, elderberry

b_5 = apple, cranberry, elderberry

b_6 = apple, banana, durian, elderberry

b_7 = banana, durian, elderberry b_8 = banana, durian

What are the frequent itemsets?

Quiz

Items = bread, milk, diaper, beer, eggs, coke

b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

What are the frequent itemsets with support ≥ 3?

A) {bread}, {milk}, {diaper}, {beer}

B) {milk, bread, diaper}

C) {bread}, {milk}, {diaper}, {beer}, {diaper, beer}, {milk, bread}

D) {bread}, {milk}, {diaper}, {beer}, {coke}

Applications

- (Physical) marketing

  Infamous beer-diaper association

- (Online) marketing
- Plagiarism detection

  I = documents, B = sentences

- Teaching planning
- Document mining
- ...

Finding frequent itemsets

Typically, **small baskets** and **many items**

items = apple, banana, cranberry, durian, elderberry
b_1 = apple, banana, durian b_2 = apple, cranberry, elderberry
b_3 = apple, durian b_4 = banana, elderberry
b_5 = apple, cranberry, elderberry
b_6 = apple, banana, durian, elderberry
b_7 = banana, durian, elderberry b_8 = banana, durian

Association rule: apple, durian *implies* banana

Confidence: $\frac{2}{3}$

Support and **confidence** are the two most important notions

Quiz

Items = bread, milk, diaper, beer, eggs, coke

b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

What is the confidence of these association rules?

- beer $\rightarrow$ diaper
- milk, diaper $\rightarrow$ coke
- milk $\rightarrow$ eggs

A) 1; $\frac{2}{3}$; 0
B) 0; $\frac{1}{2}$; 1
C) $\frac{3}{6}$; $\frac{3}{6}$; $\frac{1}{6}$
D) 1; 1; 0

Mining association rules

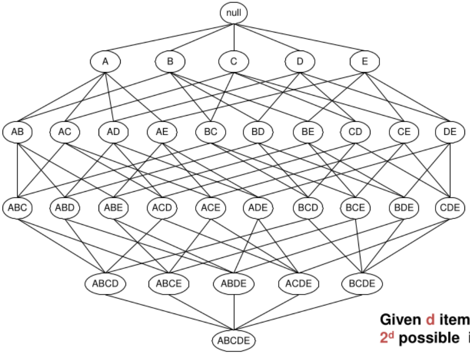
1. Find all sets with support at least $c \cdot s$
2. Find all sets with support at least s
3. If $\{i_0, i_1, ..., i_k, j\}$ has support at least $s_1 = cs$, see how many leave-one-out have support at least $s_2 \geq s$

   Imagine j is missing

4. The rule is **acceptable** iff $s_2 \geq s$, $s_1 \geq cs$, *i.e.* $\frac{s_1}{s_2} \geq c$

This can be used to extract the rules once we have the frequent itemsets

Frequent items of size 2 (aka pairs)



Given d items, there are 2^d possible itemsets

Bottleneck: finding all frequent itemsets

Finding pairs is **hard**

- Pairs are common (compared to triples)
- Support is high so few itemsets
- Let us start with pairs, then expand :)

Any ideas?

Naïve approach

Read file, count pairs as you go

For each basket of k items, $\frac{k(k-1)}{2}$ pairs

Use k nested loops to generate sets of size up to k

OK if k small

Pros:

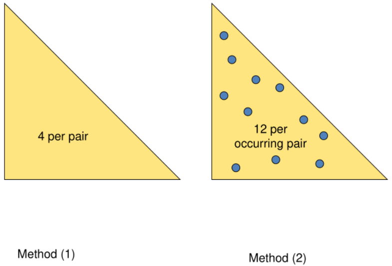
- Darn easy to implement

Cons:

- This can fail quickly (number of items^2)
- Typical datasets are huge

Computation model

How to store this data?
(1) Store everything as a list of triples i, j, c
(2) Store everything in a triangular matrix $i, j \to c$



(1) requires 4 bytes / pair
(2) requires 12 bytes / pair **that occurs**

(2) better if $\leq \frac{1}{3}$ of pairs occur

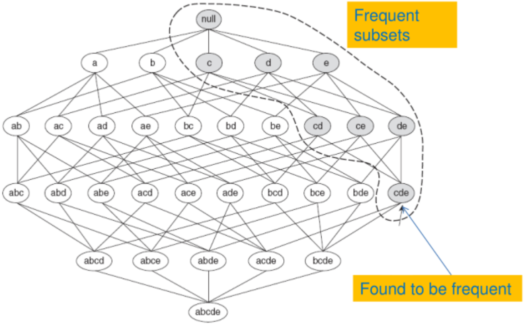
We measure an algorithm efficiency in number of *passes*
Be careful about what you store for each such set!

A-priori algorithm

Key idea: **monotonicity**

X frequent $\implies$ $X' \subseteq X$ frequent
X not frequent $\implies$ $X' \supseteq X$ not frequent

$$\forall X, X' : (X' \subseteq X) \implies s(X) < s(X')$$

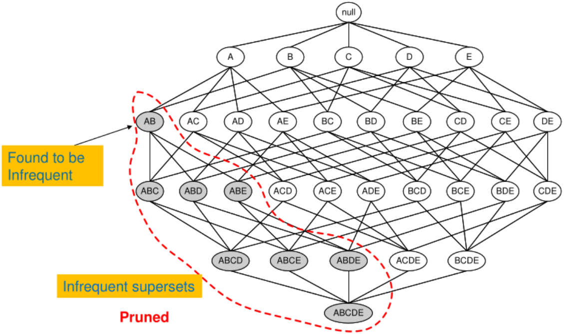


A-priori algorithm

Key idea: **monotonicity**

X frequent $\implies$ $X' \subseteq X$ frequent
X not frequent $\implies$ $X' \supseteq X$ not frequent

$$\forall X, X' : (X' \subseteq X) \implies s(X) < s(X')$$



A-priori algorithm

Pass 1: count occurrences of each item
Pass 2: Construct each pair, and count occurrences

Thanks to monotonicity, (2) is a lot faster than (1)!

Overhead: need to renumber items between steps
Combination of filter - build steps

$$C_1 \rightarrow F \rightarrow L_1 \rightarrow B \rightarrow C_2 \dots$$

A-priori example

Items = bread, milk, diaper, beer, eggs, coke

b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

Pass 1

bread	4
coke	2
milk	4
beer	3
diaper	4
eggs	1

A-priori example

Items =	bread, milk, diaper, beer, eggs, coke
b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

Pass 1

bread	4
coke	*2*
milk	4
beer	3
diaper	4
eggs	*1*

A-priori example

Items = bread, milk, diaper, beer, eggs, coke

b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

Pass 1		**Pass 2**	
bread	4	bread, milk	3
coke	*2*	bread, beer	2
milk	4	bread, diaper	3
beer	3	milk, beer	2
diaper	4	milk, diaper	3
eggs	*1*	beer, diaper	3

A-priori example

Items =	bread, milk, diaper, beer, eggs, coke
b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

Pass 1		**Pass 2**	
bread	4	bread, milk	3
coke	*2*	*bread, beer*	*2*
milk	4	bread, diaper	3
beer	3	*milk, beer*	*2*
diaper	4	milk, diaper	3
eggs	*1*	beer, diaper	3

A-priori example

Items = bread, milk, diaper, beer, eggs, coke

b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

Pass 1		**Pass 2**		**Pass 3**
bread	4	bread, milk	3	bread, milk, diaper
coke	*2*	*bread, beer*	*2*	2
milk	4	bread, diaper	3	
beer	3	*milk, beer*	*2*	
diaper	4	milk, diaper	3	
eggs	*1*	beer, diaper	3	

A-priori example

Items =	bread, milk, diaper, beer, eggs, coke
b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

Pass 1		**Pass 2**		**Pass 3**
bread	4	bread, milk	3	*bread, milk, diaper*
coke	*2*	*bread, beer*	*2*	*2*
milk	4	bread, diaper	3	
beer	3	*milk, beer*	*2*	
diaper	4	milk, diaper	3	
eggs	*1*	beer, diaper	3	

A-priori example

Items =	bread, milk, diaper, beer, eggs, coke
b_1	bread, milk
b_2	bread, diaper, beer, eggs
b_3	milk, diaper, beer, coke
b_4	bread, milk, diaper, beer
b_5	bread, milk, diaper, coke

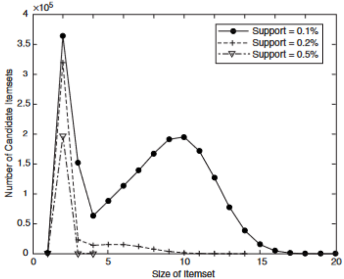
Pass 1		**Pass 2**		**Pass 3**
bread	4	bread, milk	3	*bread, milk, diaper*
coke	*2*	*bread, beer*	*2*	*2*
milk	4	bread, diaper	3	
beer	3	*milk, beer*	*2*	
diaper	4	milk, diaper	3	
eggs	*1*	beer, diaper	3	

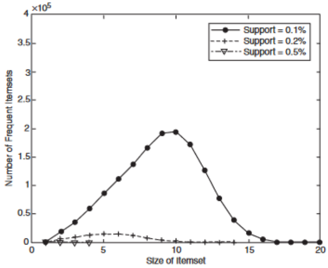
$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$ vs
$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$

A-priori algorithm

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

$k = 1$.
$F_k = \{\ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
repeat
  $k = k + 1$.
  $C_k =$ apriori-gen(F_{k-1}). {Generate candidate itemsets}
  for each transaction $t \in T$ **do**
    $C_t =$ subset(C_k, t). {Identify all candidates that belong to t}
    for each candidate itemset $c \in C_t$ **do**
      $\sigma(c) = \sigma(c) + 1$. {Increment support count}
    end for
  end for
  $F_k = \{\ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets}
until $F_k = \emptyset$
Result $= \bigcup F_k$.





Improving A-priori

Park-Chen-Yu algorithm

Key idea: 1st pass does not use a lot of memory, so keep a hashtable of pairs as well as frequent items

Have buckets contain **counts** of pairs that hash to said bucket

Hashing = not perfect; nothing in the bucket can be eliminated
But, items hashing $\rightarrow$ "bad" buckets can be eliminated

Store $\{0, 1\}$ bitmap to recover frequent buckets

Pass 2: only count pairs that hash to **frequent buckets**

Count all (i, j) that are candidates, *i.e.* i,j freq items + (i,j) maps to a frequent bucket

FP-growth algorithm

What is costly: **candidate generation**; can we skip it?

Use **divide and conquer** approach:

- Sort items by frequency
- Iteratively (on baskets) build FP-tree
- Projecting the trees per item allows to build every pair

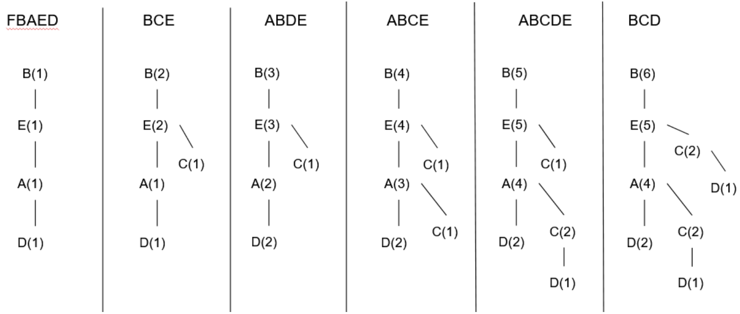
Build trees

FBAED, BCE, ABDE, ABCE, ABCDE, BCD
Counts: B: 6, E: 5, A: 4, C: 4, D:4

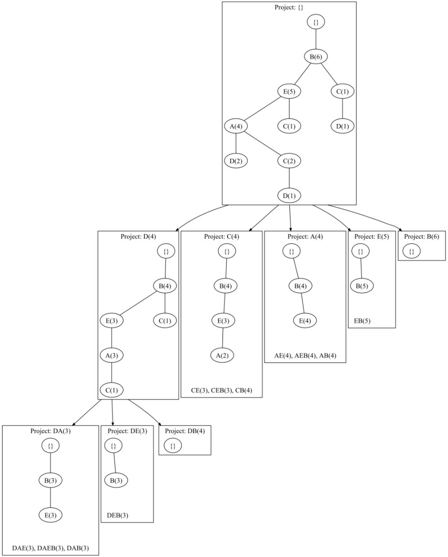
Build trees

FBAED, BCE, ABDE, ABCE, ABCDE, BCD

Counts: B: 6, E: 5, A: 4, C: 4, D:4



Get patterns



Conclusion

Finding association rules is easy, if we have the frequent itemsets

Both A-priori and FP-growth are “basic” building blocks to find frequent itemsets

There are countless improvements:

- Multistage
- Dynamic FP-growth
- Approximate methods
- FP-bonsai
- ...

The simplicity allows these algorithms to **scale**

Generalisation: Formal concept analysis