

All (Or Most) Frequent Itemsets In ≤ 2 Passes

Simple Algorithm

Savasere-Omiecinski- Navathe (SON)

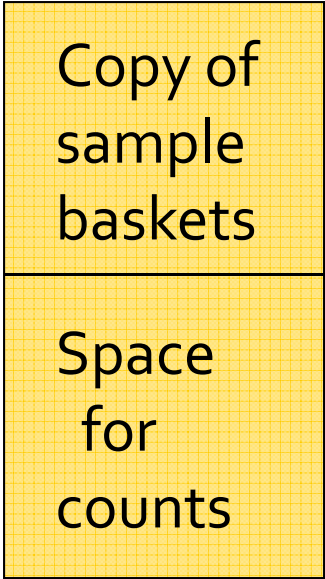
Algorithm

Toivonen's Algorithm

Simple Algorithm

- Take a random sample of the market baskets.
- Run a-priori or one of its improvements (for sets of all sizes, not just pairs) in main memory, so you don't pay for disk I/O each time you increase the size of itemsets.
- Use as your support threshold a suitable, scaled-back number.
 - **Example:** if your sample is $1/100$ of the baskets, use $s/100$ as your support threshold instead of s .

Main-Memory Picture



Copy of
sample
baskets

Space
for
counts

Simple Algorithm – Option

- Optionally, verify that your guesses are truly frequent in the entire data set by a second pass.
- But you don't catch sets frequent in the whole but not in the sample.
 - Smaller threshold, e.g., $s/125$ instead of $s/100$, helps catch more truly frequent itemsets.
 - But requires more space.

SON Algorithm

- Repeatedly read small subsets of the baskets into main memory and perform the first pass of the simple algorithm on each subset.
- An itemset becomes a candidate if it is found to be frequent in *any* one or more subsets of the baskets.

SON Algorithm – Pass 2

- On a second pass, count all the candidate itemsets and determine which are frequent in the entire set.
- Key “monotonicity” idea: an itemset cannot be frequent in the entire set of baskets unless it is frequent in at least one subset.

SON Algorithm – Distributed Version

- This idea lends itself to distributed data mining.
- If baskets are distributed among many nodes, compute *local* frequent itemsets at each node, then distribute the candidates from each node.
- Each node counts all the candidate itemsets.
- Finally, accumulate the counts of all candidates.

Toivonen's Algorithm

- Start as in the simple algorithm, but lower the threshold slightly for the sample.
 - **Example:** if the sample is 1% of the baskets, use $s/125$ as the support threshold rather than $s/100$.
 - Goal is to avoid missing any itemset that is frequent in the full set of baskets.

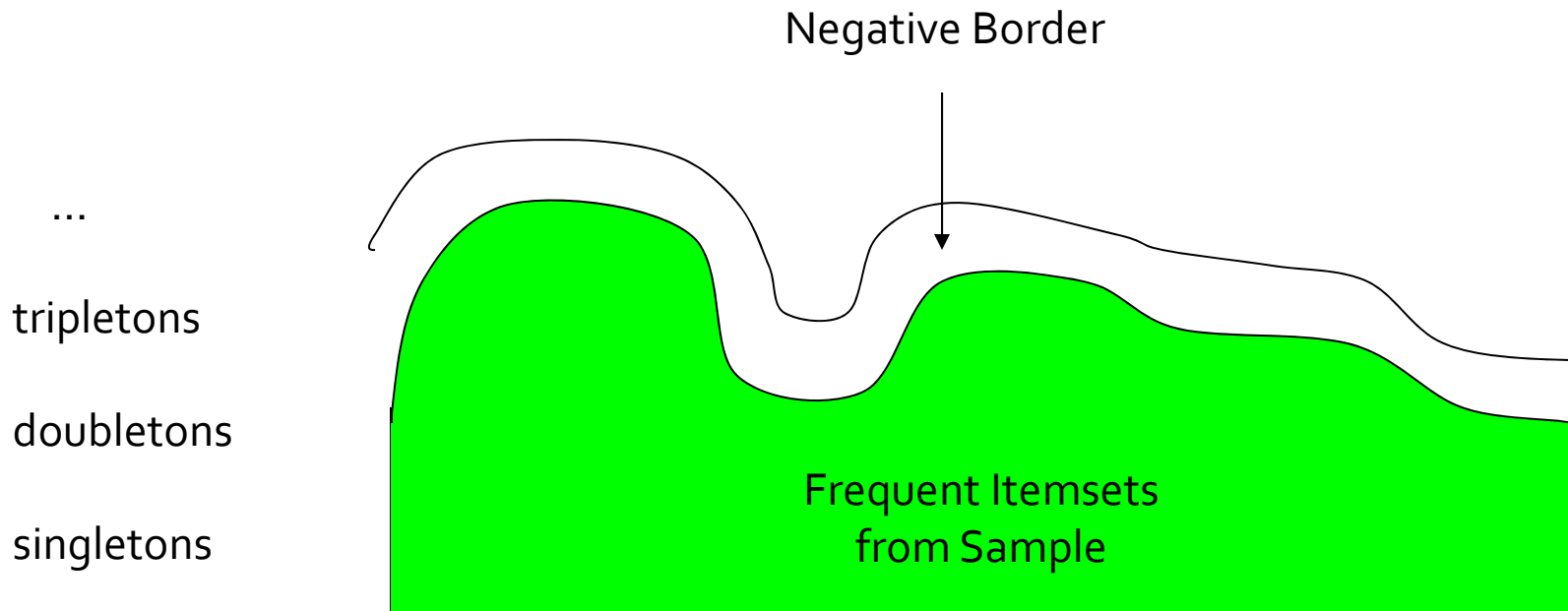
Toivonen's Algorithm – (2)

- Add to the itemsets that are frequent in the sample the *negative border* of these itemsets.
- An itemset is in the negative border if it is not deemed frequent in the sample, but *all* its immediate subsets are.

Example: Negative Border

- $\{A,B,C,D\}$ is in the negative border if and only if:
 1. It is not frequent in the sample, but
 2. All of $\{A,B,C\}$, $\{B,C,D\}$, $\{A,C,D\}$, and $\{A,B,D\}$ are.
- $\{A\}$ is in the negative border if and only if it is not frequent in the sample.
 - Because the empty set is always frequent.
 - Unless there are fewer baskets than the support threshold (silly case).

Picture of Negative Border



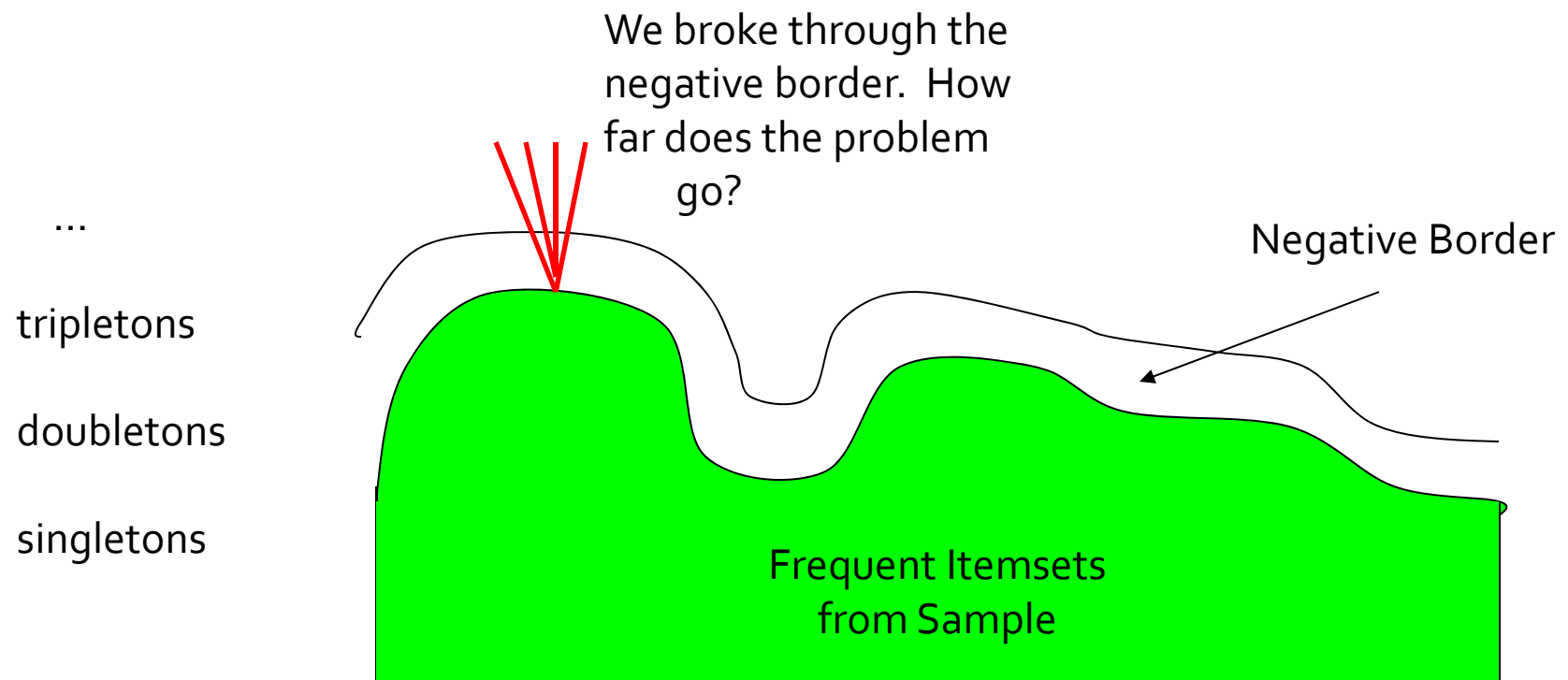
Toivonen's Algorithm – (3)

- In a second pass, count all candidate frequent itemsets from the first pass, and also count their negative border.
- If no itemset from the negative border turns out to be frequent, then the candidates found to be frequent in the whole data are *exactly* the frequent itemsets.

Toivonen's Algorithm – (4)

- What if we find that something in the negative border is actually frequent?
- We must start over again with another sample!
- Try to choose the support threshold so the probability of failure is low, while the number of itemsets checked on the second pass fits in main-memory.

If Something in the Negative Border Is Frequent . . .



Theorem:

- If there is an itemset that is frequent in the whole, but not frequent in the sample, then there is a member of the negative border for the sample that is frequent in the whole.

Proof:

- Suppose not; i.e.;
 1. There is an itemset S frequent in the whole but not frequent in the sample, and
 2. Nothing in the negative border is frequent in the whole.
- Let T be a **smallest** subset of S that is not frequent in the sample.
- T is frequent in the whole (S is frequent + monotonicity).
- T is in the negative border (else not “smallest”).