13 Frequent Itemsets

A classic problem in data mining is *association rule mining*. The basic problem is posed as follows: We have a large set of m *tuples* $\{T_1, T_2, \ldots, T_m\}$, each tuple $T_j = \{t_{j,1}, t_{j,2}, \ldots, t_{j,k}\}$ has a small number (not all the same k) of items from a domain $[n]$. *Think of the items as products in a grocery store and the tuples as things bought in the same purchase.*

Then the goal is to find times which frequently co-occur together. *Think of what items to people frequently buy together: a famous example is "diapers" and "beer" found in real data.*

Again, like finding frequent items (heavy hitters), we assume m and n is very large (maybe not quite so large as before), but we don't want to check all pairs $\binom{n}{2} \approx n^2/2$ against all m tuples since that would take roughly $n^2 m/2$ time and be way too long. Moreover, m does not fit in memory, so we only want to scan over it, in one pass (or a small number of passes).

13.1 A-Priori Algorithm

We now describe the *A-Priori Algorithm* for finding frequent item sets, by Agrawal + Srikant [1]. The key idea is that any itemset that occurs frequently together must have each item (or any subset) occur at least as frequently.

First Pass. We first make one pass on all tuples, and keep a count for all n items. A hash table can be used. We set a threshold ε and only keep items that occur at least εm times (that is in at least ε percent of the tuples). For any frequent itemset that occurs in at least $100\varepsilon\%$ of the tuples, must have each item also occur in at least $100\varepsilon\%$ of the tuples.

A reasonable choice of ε might be 0.01, so we only care about itemsets that occur in 1% of the tuples. Consider that there are only n_1 items above this threshold. For instance if the maximum tuple size is $k_{\max}$ then we know $n_1 \leq k_{\max}/\varepsilon$. For $k_{\max} = 80$, and $\varepsilon = 0.01$, then $n_1 \leq 8000$, easily small enough to fit in memory. Note this is independent of the n or m.

Second Pass. We now make a second pass over all tuples. On this pass we search for frequent pairs of items, specifically, those items which occur in at least an ε-fraction of all tuples. Both items must have been found in the first pass. So we need to consider only $\binom{n_1}{2} \approx n_1^2/2$ pairs of counters for these pairs of elements.

After the pass, again we can then discard all pairs which occur less than an ε-fraction of all tuples. This remaining set is likely far less than $n_1^2/2$.

These remaining pairs are already quite interesting! They record all pairs that co-occur in more than an ε-fraction of purchases, and of course include those pairs which occur together even more frequently.

Further Passes. On the ith pass we can find sets of i items that occur together frequently (above an ε-threshold). For instance, on the third pass we only need to consider triples were all sub-pairs occur at least an ε-fraction of times themselves. These triples can be found as follows:

Sort all pairs (p, q) by their smaller indexed item (w.l.o.g. let this be p). Then for each smaller indexed item p, consider all completions of this pair q. Now look at the pairs (q, r) with smaller item as q. For each of these pairs, check if the pair (p, r) also remains. Only triples (p, q, r) which pass all of these tests are given counters in the third pass.

13.1.1 Example

Consider the following dataset where I want to find all itemsets that occur in at least $1/3$ of all tuples (at least 4 times):

$$T_1 = \{1, 2, 3, 4, 5\}$$
$$T_2 = \{2, 6, 7, 9\}$$
$$T_3 = \{1, 3, 5, 6\}$$
$$T_4 = \{2, 6, 9\}$$
$$T_5 = \{7, 8\}$$
$$T_6 = \{1, 2, 6\}$$
$$T_7 = \{0, 3, 5, 6\}$$
$$T_8 = \{0, 2, 4\}$$
$$T_9 = \{2, 4\}$$
$$T_{10} = \{6, 7, 9\}$$
$$T_{11} = \{3, 6, 9\}$$
$$T_{12} = \{6, 7, 8\}$$

After the first pass 1 have the following counters:

0	1	**2**	**3**	4	5	**6**	**7**	8	**9**
2	3	**5**	**4**	3	3	**8**	**4**	2	**4**

So only $n_1 = 5$ items survive $\{2, 3, 6, 7, 9\}$.

In pass 2 we consider $\binom{n_1}{2} = 10$ pairs: $\{(2,3), (2,6), (2,7), (2,9), (3,6), (3,7), (3,9), (6,7), (6,9), (7,9)\}$. And we find the following counts:

$(2,3)$	$(2,6)$	$(2,7)$	$(2,9)$	$(3,6)$	$(3,7)$	$(3,9)$	$(6,7)$	$(\mathbf{6,9})$	$(7,9)$
1	3	1	2	3	0	1	3	**4**	2

We find that the only itemset pair that occurs in at least $1/3$ of all baskets is $(6, 9)$.

Thus there can be no itemset triple (or larger grouping) which occurs in all $1/3$ of all tuples since then all of its pairs would need to be in $1/3$ of all baskets, but there is only one such pair that satisfies that property.

We can now examine the association rules. And see that the count of item 6 is quite large 8, and is much bigger than that of item 9 which is only 4. Since there are 4 pairs $(6, 9)$, then every time 9 occurs, 6 also occurs.

Improvements. We can do better than keeping a counter for each item. If we know that n_1 will be at most a certain value (based on $k_{\max}$ and ε), then we only need that many counters, and we can use Misra-Gries to find all of these items. Although we may also find some other items, so its advised to use maybe $3n_1$ counters in Misra-Gries.

Other techniques have been developed to improve on this using Bloom Filters.

13.2 Bloom Filters

The goal of a Bloom Filter [2] is to maintain a subset $S \subset [n]$ in small space. It must maintain all items, but may allow some items to appear in the set even if they are not. That is, it allows *false positives*, but not *false negatives*.

Algorithm 13.2.1 Bloom(S)

for $x \in S$ **do**
  for $j = 1$ **to** k **do**
    Set $B[h_j(x)] = 1$

It maintains an array B of m bits. Each is initialized to 0. It uses k (random) hash functions $\{h_1, h_2, \ldots, h_k\} \in \mathcal{H}$. It then runs streaming Algorithm 13.2.1 on S.

Then on a query to see if $y \in S$, it returns YES only if for *all* $j \in [k]$ that $B[h_j(y)] = 1$. Otherwise it returns NO.

So any item which was put in B it sets all associated hash values as 1, so on a query it will always return YES (no false negatives).

However, it may return YES for an item even if it does not appear in the set. It does not even need to have the exact same hash values from another item in the set, each hash collision could occur because a different item.

Analysis: We consider the case with $|S| = n$ items, where we maintain m bits in B, and use k hash functions.

$$
\begin{aligned}
1 - 1/m &\text{ Probability a bit not set to 1 by a single hash function} \\
(1 - 1/m)^k &\text{ Probability a bit not set to 1 by } k \text{ hash functions} \\
(1 - 1/m)^{kn} &\text{ On inserting } n \text{ elements, probability a bit is 0 } (\star) \\
1 - (1 - 1/m)^{kn} &\text{ On inserting } n \text{ elements, probability a bit is 1} \\
(1 - (1 - 1/m)^{kn})^k &\text{ Probability a false positive (using } k \text{ hash functions)} \\
\approx (1 - e^{-kn/m})^k &
\end{aligned}
$$

The "right" value of k is about $k \approx (m/n) \ln 2$.

($\star$) *This step is not quite right, although it is not too far off. It assumes independence of which bits are being set. Unfortunately, in most presentations of Bloom filters (and some peer-reviewed research papers) this is presented as the right analysis, but it is not. But it gives a pretty good illustration of how it works. The correct analysis is much more complicated [3].*

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings 20th International Conference on Very Large Data Bases*, 1994.

[2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[3] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. On the false-positive rate of Bloom filters. *Journal of Information Priocessing Letters*, 108:210–213, 2008.