Lecture 13

CountMin Algorithm

Course: Algorithms for Big Data

Instructor: Hossein Jowhari

Department of Computer Science and Statistics
Faculty of Mathematics
K. N. Toosi University of Technology

Spring 2021

Heavy Hitters: Previous Lecture

Definition: Given a frequency vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, the coordinate i is a ϵ-HH iff

$$x_i \geq \epsilon \sum_{i=1}^{n} x_i = \epsilon \|\boldsymbol{x}\|_1 = \epsilon F_1$$

Definition: Let H_ϵ denote all ϵ-HHs.

The Majority algorithm is a counter-based algorithm for computing H_ϵ. It outputs the subset $S \subseteq [n]$ where $H_\epsilon \subseteq S$ and $|S| \leq \frac{1}{\epsilon}$. The algorithm works in $O(\frac{1}{\epsilon})$ words of space.

CountMin

- CountMin is a randomized data structure for estimating the frequency of the elements in the stream.
- Given a stream of m items where each item $\in \{1, \ldots, n\}$, let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be the associated frequency vector. Note that $m = \sum_{i=1}^{n} x_i$. Given index i, the CountMin data structure outputs x'_i where

$$x_i \leq x'_i \leq x_i + \epsilon m$$

with probability $1 - \delta$.

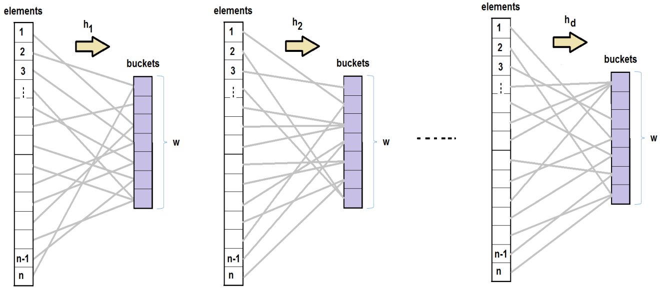
- Note that CountMin alone is not efficient in finding the heavy hitters but using additional ideas we can use it to find the heavy hitters.
- CountMin takes $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$ words of space.

How does CountMin work?

CountMin randomly hashes the elements $[n]$ into buckets. For this it uses a series of pairwise independent hash functions

$$h_i(x) = a_i x + b_i \mod w \qquad i = 1, \ldots, d$$

Each hash function h_i hashes the elements into w buckets. The algorithm stores the hash functions and the buckets.



Notation: Let $C_i(j)$ denote the value j-th bucket in the function h_i.

Initialization: In the beginning, all buckets are zero. $\forall i, j,\ C_i(j) = 0$.

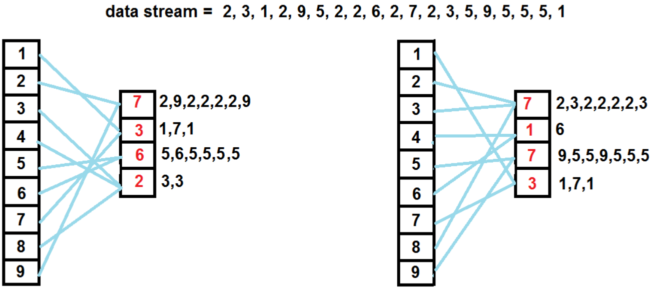
Stream Processing: For each x in the stream, the algorithm increments the value of $C_i(h(x))$ for each i.

$$\forall i,\ \ C_i(h(x)) \leftarrow C_i(h(x)) + 1$$

Query Processing Given an element $y \in [n]$, the estimate for x_y is

$$\min_{i=1}^{n} C_i(h(y))$$

Example



In the above example, the true frequencies are

$$x_1 = 2, x_2 = 5, x_3 = 2, x_4 = 0, x_5 = 5, x_6 = 1, x_7 = 1, x_8 = 0, x_9 = 2$$

Some of the estimates are as follows

$$x'_1 = 3, x'_2 = 7, x'_4 = 1, x'_5 = 6$$

Analysis of CountMin

Fix an element $y \in [n]$ and hash function h_i. Suppose $h_i(y) = b$.

Let random variable $X_j = x_j$ if $h_i(j) = b$ otherwise $X_j = 0$.

$$E[C_i(b)|h_i(y) = b] = E[\sum_{j=1}^{n} X_j|h_i(y) = b] = x_y + \frac{1}{w}\sum_{j \neq y} x_j \leq x_y + \frac{m}{w}$$

Fact: $C_i(b) \geq x_y$

Using Markov Inequality:

Conditioned on $h_i(y) = b$, we have

$$Pr(C_i(b) \geq x_y + \frac{2m}{w}) = Pr(C_i(b) - x_y \geq \frac{2m}{w}) \leq \frac{E[C_i(b)] - x_y}{\frac{2m}{w}} \leq \frac{1}{2}$$

Since we select the hash functions h_i's independently, we have

$$Pr(\min_{i=1}^{d} C_i(h(y)) \geq x_y + \frac{2m}{w}) \leq \prod_{i=1}^{d} (\frac{1}{2}) = (\frac{1}{2})^d \leq \delta$$

$$d = \Omega(\log(\frac{1}{\delta})), \qquad w = \frac{2}{\epsilon}$$

$$Pr(\min_{i=1}^{d} C_i(h(y)) \geq x_y + \epsilon m) \leq \delta$$

Space Complexity: CountMin stores the buckets and the hash functions. Therefore the space needed is $wd + 2d + 1 = O(wd) = O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$. (Each hash function is represented by 2 numbers.)

How to find the heavy hitters using CountMin?

Inefficient way: We find the estimates for all numbers in $[n]$. We query the data structure for all $i \in [n]$. Let the approximation parameter in CountMin be λ. We also set $\delta \leftarrow \frac{\delta}{n}$ and (using union bound) we get

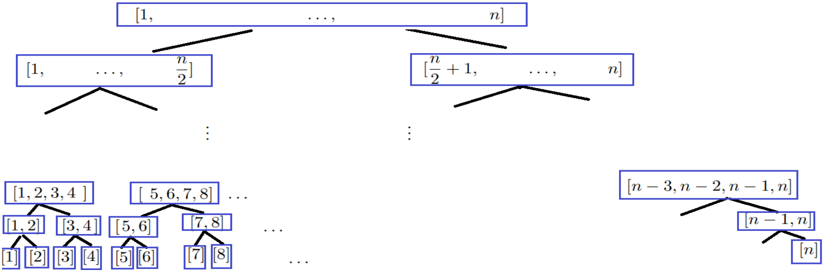
$$\forall y \in [n], \;\; x_y \leq x'_y \leq x_y + \lambda m$$

with probability $1 - \delta$. This takes $O(\frac{1}{\lambda}\log(\frac{n}{\delta}))$ space.

Our goal is to find H_ϵ. We let $\lambda = \frac{\epsilon}{2}$. We query every $i \in [n]$. If $x'_i \geq \epsilon m$, we include i in S (the candidate set for heavy hitters.)

Fact: (With probability $1 - \delta$) if $i \in H_\epsilon$ then $i \in S$. Also $|S| \leq \frac{2}{\epsilon}$.

More efficient way: Consider the set of dyadic intervals as show below.

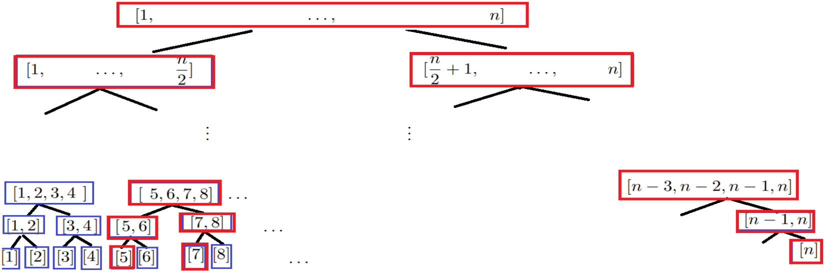


We treat each interval as an element and build the CountMin data structure for each level. There are $\log n$ levels.

Observation 1: Number of occurrences in each level is m. Therefore in each level there are at most $\frac{1}{\epsilon}$ heavy hitters.

Observation 2: If the interval $[u, v]$ is not a heavy hitter, then none of its sub-intervals can be a heavy hitter.

Observation 3: Suppose the red numbers at the bottom level (the leaves of the tree) are the heavy hitters. If a leaf u is a heavy hitter its ancestors are all heavy hitters.



Algorithm: From the top we inspect the tree of the intervals. In each level we find the heavy hitters. If an interval is not a heavy hitter we do not inspect its sub-intervals.

Question: How many intervals do we check?

If the interval $[u, v]$ is a heavy hitter, we check its two children. Therefore at most $k = O(\frac{1}{\epsilon}\log n)$ intervals are checked.

Space complexity: $O(\frac{1}{\epsilon}\log\frac{k}{\delta}\log n)$

References

[1] Misra, Jayadev, and David Gries. "Finding repeated elements." Science of computer programming 2.2 (1982): 143-152.

[2] Cormode, Graham, and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications." Journal of Algorithms 55.1 (2005): 58-75.