COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Spring 2020.

Lecture 8

tinyurl.com/musco2020

LOGISTICS

- Problem Set 1 was due this past Friday. Will be graded by next week.
- Problem Set 2 to be released end of this week and due $\sim 3/6$.

ACADEMIC HONESTY ON PROBLEM SETS

- We take academic honestly on the problem sets seriously.
- If caught copying from another group (or allowing someone to copy your work), copying from problem sets or answer keys from past semesters, etc. you will receive a 0% **on the problem set and** 5% **off your final course grade.**
- Even if one group member copies, the rest of the group is at risk of the same deduction. Don't just split up the problems and not work on them together.
- You can change your problem set group from assignment to assignment.

Last Class:

- SimHash for cosine similarity
- Applications to e.g., approximate neural network computation.
- Introduction to the Frequent Elements (heavy-hitters) problem in data streams.
- The Boyer-Moore voting algorithm for majority.

This Class:

- Extend Boyer-Moore to the general Frequent Elements: problem: Misra-Gries summaries.
- Count-min sketch (random hashing for frequent element estimation).

UPCOMING

Next Few Classes:

- Random compression methods for high dimensional vectors. The Johnson-Lindenstrauss lemma.
- Compressed sensing (sparse recovery) and connections to the frequent elements problem.

After That: Spectral Methods

- PCA, low-rank approximation, and the singular value decomposition.
- Spectral clustering and spectral graph theory.

Will use a lot of linear algebra. May be helpful to refresh.

- Vector dot product, addition, length. Matrix vector multiplication.
- Linear independence, column span, orthogonal bases, rank.
- Orthogonal projection, eigendecomposition, linear systems.

k-**Frequent Items (Heavy-Hitters) Problem**: Consider a stream of n items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times. E.g., for $n = 9$, $k = 3$:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
5	12	3	3	4	5	5	10	3

- At most $\frac{n}{n/k} = k$ items are ever returned.
- Think of $k = 100$. Want items appearing $\geq 1\%$ of the time.
- Easy with $O(n)$ space – store the count for each item and return the one that appears $\geq n/k$ times.

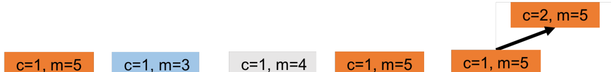
Applications: Finding viral products/media/searches, frequent itemset mining, detecting DoS and other attacks, 'iceberg queries' in databases.

MISRA-GRIES SUMMARIES

k-Frequent Items (Heavy-Hitters) Problem: Consider a stream of n items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

Boyer-Moore Voting Algorithm:Misra-Gries Summary:

- Initialize count $c := 0$, majority element $m := \perp$counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \perp$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.
  - If $m_j = x_i$ for some j, set $c_j := c_j + 1$.
  - Else let $t = \arg\min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all j.



Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \perp$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some j, set $c_j := c_j + 1$.
  - Else let $t = \arg\min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all j.

c_1=0, m_1=⊥ | c_1=1, m_1=5

c_2=0, m_1=⊥ | c_2=0, m_1=⊥

c_3=0, m_1=⊥ | c_3=0, m_1=⊥

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_1	x_2	x_3
5	12	3	3	4	5	5	10	3	5	12	3

Claim: At the end of the stream, all items with frequency $\geq \frac{n}{k}$ are stored.

Claim: At the end of the stream, the Misra-Gries algorithm stores k items, including all those with frequency $\geq \frac{n}{k}$.

Intuition:

- If there are exactly k items, each appearing exactly n/k times, all are stored (since we have k storage slots).
- If there are $k/2$ items each appearing $\geq n/k$ times, there are $\leq n/2$ irrelevant items, being inserted into $k/2$ 'free slots'.
- May cause $\frac{n/2}{k/2} = \frac{n}{k}$ decrement operations. Few enough that the heavy items (appearing n/k times each) are still stored.

Anything undesirable about the Misra-Gries output guarantee?

May have false positives – infrequent items that are stored.

APPROXIMATE FREQUENT ELEMENTS

Issue: Misra-Gries algorithm stores k items, including all with frequency $\geq n/k$. But may include infrequent items.

- In fact, no algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency n/k (should be output) and $n/k - 1$ (should not be output).

x_1	x_2	x_3	x_4	x_5	x_6	...	$x_{n-n/k+1}$	...	x_n
3	12	9	27	4	101		3		3

n/k-1 occurrences

(ϵ, k)-Frequent Items Problem: Consider a stream of n items $x_1, \ldots, x_n$. Return a set F of items, including all items that appear at least $\frac{n}{k}$ times and only items that appear at least $(1-\epsilon) \cdot \frac{n}{k}$ times.

Misra-Gries Summary: (ϵ-error version)

- Let $r := \lceil k/\epsilon \rceil$
- Initialize counts $c_1, \ldots, c_r := 0$, elements $m_1, \ldots, m_r := \perp$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some j, set $c_j := c_j + 1$.
  - Else let $t = \arg\min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all j.
- Return any m_j with $c_j \geq (1 - \epsilon) \cdot \frac{n}{k}$.

Claim: For all m_j with true frequency $f(m_j)$:

$$f(m_j) - \frac{\epsilon n}{k} \leq c_j \leq f(m_j).$$

Intuition: $\#$ items stored r is large, so relatively few decrements.

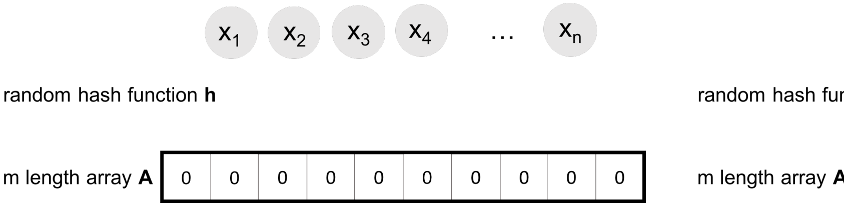
Implication: If $f(m_j) \geq \frac{n}{k}$, then $c_j \geq (1 - \epsilon) \cdot \frac{n}{k}$ so the item is returned.
If $f(m_j) < (1 - \epsilon) \cdot \frac{n}{k}$, then $c_j < (1 - \epsilon) \cdot \frac{n}{k}$ so the item is not returned.

Upshot: The (ϵ, k)-Frequent Items problem can be solved via the Misra-Gries approach.

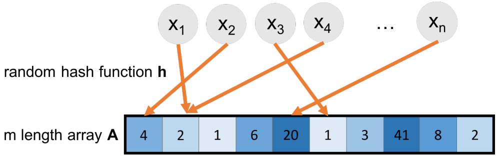
- Space usage is $\lceil k/\epsilon \rceil$ counts – $O\left(\frac{k \log n}{\epsilon}\right)$ bits and $\lceil k/\epsilon \rceil$ items.
- Deterministic approximation algorithm.

A common alternative to the Misra-Gries approach is the count-min sketch: a randomized method closely related to bloom filters.

- A major advantage: easily distributed to processing on multiple servers. Build arrays $A_1, \ldots, A_s$ separately and then just set $A := A_1 + \ldots + A_s$.



Will use $A[\mathbf{h}(x)]$ to estimate $f(x)$, the frequency of x in the stream. I.e. $|\{x_i : x_i = x\}|$



Use $A[\mathbf{h}(x)]$ to estimate $f(x)$

Claim 1: We always have $A[\mathbf{h}(x)] \geq f(x)$. Why?

- $A[\mathbf{h}(x)]$ counts the number of occurrences of any y with $\mathbf{h}(y) = \mathbf{h}(x)$, including x itself.
- $A[\mathbf{h}(x)] = f(x) + \sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y)$.

> $f(x)$: frequency of x in the stream (i.e., number of items equal to x). $\mathbf{h}$: random hash function. m: size of count-min sketch array.

$$A[\mathbf{h}(x)] = f(x) + \underbrace{\sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y)}_{\text{error in frequency estimate}}.$$

Expected Error:

$$\mathbb{E}\left[\sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y)\right] = \sum_{y \neq x} \Pr(\mathbf{h}(y) = \mathbf{h}(x)) \cdot f(y)$$

$$= \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}$$

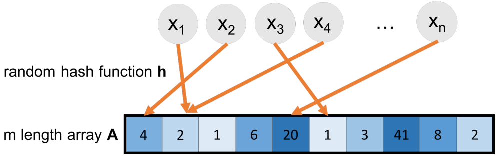
What is a bound on probability that the error is $\geq \frac{3n}{m}$?

Markov's inequality: $\Pr\left[\sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y) \geq \frac{3n}{m}\right] \leq \frac{1}{3}$.

What property of **h** is required to show this bound? 2-universal.

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). $\mathbf{h}$: random hash function. m: size of count-min sketch array.

COUNT-MIN SKETCH ACCURACY



Claim: For any x, with probability at least $2/3$,

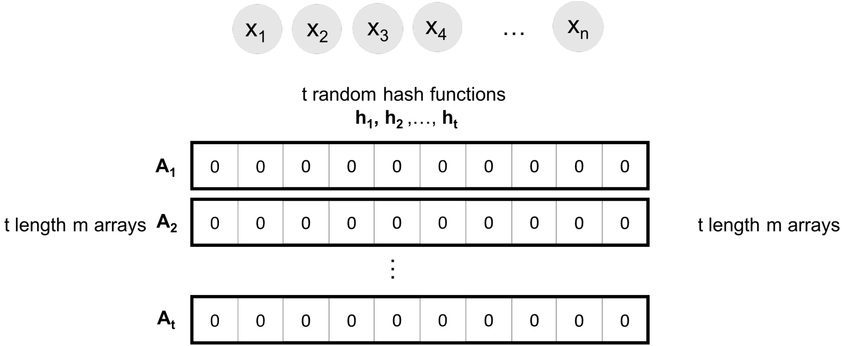
$$f(x) \leq A[\mathbf{h}(x)] \leq f(x) + \frac{3n}{m} \cdot \frac{\epsilon n}{k}.$$

To solve the (ϵ, k)-Frequent elements problem, set $m = \frac{3k}{\epsilon}$.

How can we improve the success probability? Repetition.

> $f(x)$: frequency of x in the stream (i.e., number of items equal to x). **h**: random hash function. m: size of count-min sketch array.

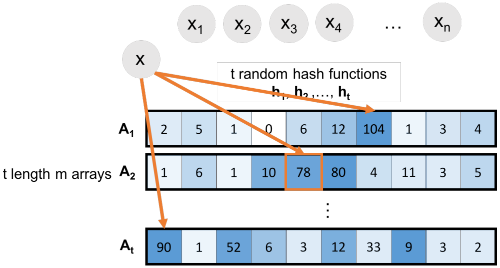
COUNT-MIN SKETCH ACCURACY



Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[\mathbf{h}_i(x)]$. (count-min sketch)

Why min instead of mean or median? The minimum estimate is always the most accurate since they are all overestimates of the true frequency!

COUNT-MIN SKETCH ANALYSIS



Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[\mathbf{h}_i(x)]$

- For every x and $i \in [t]$, we know that for $m = O(k/\epsilon)$, with probability $\geq 2/3$:

$$f(x) \leq A_i[\mathbf{h}_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$? $1 - 1/3^t$.
- To get a good estimate with probability $\geq 1 - \delta$, set $t = O(\log(1/\delta))$.

Upshot: Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O\left(\log(1/\delta) \cdot k/\epsilon\right)$ space.

- Accurate enough to solve the (ϵ, k)-Frequent elements problem.
- Actually identifying the frequent elements quickly requires a little bit of further work.

  One approach: Store potential frequent elements as they come in. At step i remove any elements whose estimated frequency is below i/k. Store at most $O(k)$ items at once and have all items with frequency $\geq n/k$ stored at the end of the stream.

Questions on Frequent Elements?