Algorithms for Big Data (Fall 25)

Lecture 4

Frequency Moment Estimation in Streaming

Ali Vakilian (vakilian@vt.edu)

Virginia Tech

Algorithms for Big Data
Fall 2025

Intro to Frequency Moments

Input: A data stream $S = (e_1, e_2, e_3, \dots, e_N)$, that are seen one by one, where each $e_i \in [n]$ (for known n or an upper bound on n).

Setting: Streaming; the algorithm has B tokens of memory ($B \ll N$)

The Goal: Compute some norm of the observed vector; a fundamental class of problems [Alon, Matias, Szegedy'99].

Example: $n = 9$ and stream is $9, 1, 1, 3, 5, 8, 9, 7, 2, 1, 3, 9, 8, 4$

Frequency Moments

Input: A data stream $S = (e_1, e_2, e_3, \dots, e_N)$, that are seen one by one, where each $e_i \in [n]$ (for known n or an upper bound on n).

- Let f_i denotee the frequency of item i in the stream
- Consider vector $\boldsymbol{f} = (f_1, \dots, f_n)$

The Goal: Given $k \geq 0$, compute the k-th moment of $\boldsymbol{f}$ denoted as

$$F_k = \sum_{i \in [n]} f_i^k$$

(similarly, we can also consider ℓ_k norm of f which is $(F_k)^{1/k}$)

Example: $n = 9$ and stream is $9, 1, 1, 3, 5, 8, 9, 7, 2, 1, 3, 9, 8, 4$

- $F_1 = 14$
- $F_2 = 30$

$\boldsymbol{f} = (3,1,2,1,1,0,1,2,3)$

Frequency Moments

Input: A data stream $S = (e_1, e_2, e_3, \dots, e_N)$, that are seen one by one, where each $e_i \in [n]$ (for known n or an upper bound on n).

- Let f_i denotee the frequency of item i in the stream
- Consider vector $\boldsymbol{f} = (f_1, \dots, f_n)$

The Goal: Given $k \geq 0$, compute the k-th moment of $\boldsymbol{f}$ denoted as

$$F_k = \sum_{i \in [n]} f_i^k$$

(similarly, we can also consider ℓ_k norm of f which is $(F_k)^{1/k}$)

Important Regimes

F_0: number of distinct elements

F_1: length of stream

F_2: fundamental function (MSE, distance in ,...)

F_∞: maximum frequency (heavy hitters)

F_k for $0 < k < 1$ and $1 < k < 2$

F_k for $2 < k < \infty$

Frequency Moments: Questions

(I) Estimation. Given k, estimate F_k exactly/approximately using small memory in one pass over the stream.

(II) Sampling. Given k, sample an item i proportional to f_i^k / F_k using small memory in one pass over the stream.

(III) Sketching. Given k, create a small size summary (sketch) of the frequency vector providing point query (or other statistics), in one pass over the stream.

Frequency Moments: Questions

(I) Estimation. Given k, estimate F_k exactly/approximately using small memory in one pass over the stream.

(II) Sampling. Given k, sample an item i proportional to f_i^k / F_k using small memory in one pass over the stream.

(III) Sketching. Given k, create a small size summary (sketch) of the frequency vector providing point query (or other statistics), in one pass over the stream.

- With $O(n)$ space; easy: store the vector explicitly
- What if we are restricted to use $\ll n$ words of memory?
  - In particular, $O(\log^c n)$ for some fixed $c \geq 1$ (what we refer to as polylog(n))
  - Note that to store a single word, we require $O(\log n)$bits.
  - Memory consumption is quite optimal.

Mostly, $\Omega(n)$ lower bound for exact answer

Approximate Estimate/Solution

Relative Approximation

An algorithms $\mathcal{A}$ provides an α-relative approximation to a non-negative function g over the stream $\boldsymbol{e} \coloneqq e_1, \dots . e_m$ if

$$\left|\frac{\mathcal{A}(\boldsymbol{e})}{g(\boldsymbol{e})} - 1\right| \leq \alpha$$

- (Maximization: $\frac{\mathcal{A}(\boldsymbol{e})}{g(e)} \geq 1 - \alpha$) & (Minimization: $\frac{\mathcal{A}(\boldsymbol{e})}{g(e)} \leq 1 + \alpha$)

- Randomized: (ϵ, δ)-relative approximation if $\Pr\left[\left|\frac{\mathcal{A}(\boldsymbol{e})}{g(\boldsymbol{e})} - 1\right| \leq \epsilon\right] \geq 1 - \delta$

Also referred to as multiplicative approximation.

Additive Approximation

An algorithms $\mathcal{A}$ provides an α-additive approximation to a function g over the stream $\boldsymbol{e} \coloneqq e_1, \dots . e_m$ if

$$|\mathcal{A}(\boldsymbol{e}) - g(\boldsymbol{e})| \leq \alpha$$

- Randomized: (ϵ, δ)-relative approximation if $\Pr[|\mathcal{A}(e) - g(e)| \leq \epsilon] \geq 1 - \delta$

Typically, useful when some scaling/normalization on g happens.

Estimating Distinct Elements

Distinct Element Problem

Estimate the number of **unique items** in a large dataset w/o storing all the items.

Use cases:

- Tracking the number of unique visitors to a popular website in real-time.
- **Database Query Optimization:** In a complex query, the database's query planner needs to estimate the number of unique values in different columns to decide the most efficient way to execute the query.
- **Online Advertising:** Ad platforms need to measure the **reach** of a campaign, which is the number of unique people who saw an advertisement.

Non-Streaming Solutions

- Use standard **dictionary** data structures:
  - Processing a list of n elements from d distinct items
  - **Binary Search Trees:** $O(d)$ space and total time of $O(n \log d)$
  - **Hashing:** $O(d)$ space and expected total time of $O(n)$

? How to do it much more space efficiently now that we look for an estimate only?

DistinctElements:

Initialize an empty dictionary $\mathcal{D}$
$d \leftarrow 0$
while an item e in stream arrives:
  if $e \notin \mathcal{D}$ **then**
    insert e into $\mathcal{D}$
    $d \leftarrow d + 1$
return d

(Idealized) Flajolet-Martin Algorithm

- Use hash function $h: [n] \to [m]$ for some m polynomial in n.
- Store only the minimum hash value observed so far. I.e., $\min_{i \in [n]} h(e_i)$.
- Space complexity: $O(\log m) = O\big(\log\big(poly(n)\big)\big) = O(\log n)$

For this analysis, we will disregard the space required to store the hash function

Why it works? (analysis of the estimation)

- Consider an ideal hash function $h: [n] \to [0,1]$ that is fully random
- If we have d distinct element, what is the expected value of their minimum hash values?

(Idealized) Flajolet-Martin Algorithm

- Use hash function $h: [n] \to [m]$ for some m polynomial in n.
- Store only the minimum hash value observed so far. I.e., $\min_{i \in [n]} h(e_i)$.
- Space complexity: $O(\log m) = O\left(\log\left(poly(n)\right)\right) = O(\log n)$

Theorem. Suppose $X_1, \ldots, X_d$ are r.v.s that are independent and uniformly distributed in $[0,1]$, and let $Y = \min_{i \in [d]} X_i$.
Then, $\mathbb{E}[Y] = 1/(d+1)$.

DistinctElements:
    ideal hash function h
    $y \leftarrow 1$
    while an item e arrives:
        $y = \min(y, h(e))$
    return $\frac{1}{y} - 1$

Analysis of its Expectation

$$\begin{aligned}\Pr[Y \le t] \\ &= 1 - \Pr[X_1 > t \wedge \cdots \wedge X_d > t] \\ &= 1 - \textstyle\prod_{i\in[d]} \Pr[X_i > t] && \text{(by independence of } X_i\text{)} \\ &= 1 - (1-t)^d.\end{aligned}$$

The pdf of Y is $d(1-t)^{d-1}$. So,

$\mathbb{E}[Y] = \int_0^1 t \cdot d(1-t)^{d-1}\, dt$ (by the definition of $\mathbb{E}$)

$\mathbb{E}[Y] = \frac{1}{d+1}$ (by change of variable $z = 1 - t$)

Concentration

Need to bound variance too. Recall $\mathrm{Var}[Y] = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$

How to compute $\mathbb{E}[Y^2]$? Similar to $\mathbb{E}[Y]$ calculation.

The pdf of Y is $d(1-t)^{d-1}$. So,

$$\mathbb{E}[Y^2] = \int_0^1 t^2 \cdot d(1-t)^{d-1}\, dt \qquad \text{(by the definition of } \mathbb{E}\text{)}$$

$$= \frac{2}{(d+1)(d+2)} \qquad \text{(by change of variable } z = 1-t\text{)}$$

$$\Rightarrow \mathrm{Var}[Y] = \frac{2}{(d+1)(d+2)} - \frac{1}{(d+1)^2} = \frac{d}{(d+1)^2(d+2)} \leq 1/(d+1)^2$$

By Chebyshev's inequality:

$$\Pr[|Y - \mathbb{E}[Y]| \geq \epsilon \mathbb{E}[Y]] \leq \frac{\mathrm{Var}[Y]}{(\epsilon \mathbb{E}[Y])^2} \leq 1/\epsilon^2$$

What does it imply for our final estimate ($\epsilon = 2$)?

How to boost the accuracy? A FAMILIAR RECIPE

- **(Averaging)** Take average of $k = O(1/\epsilon^2)$ independent estimators to reduce variance
  - Apply Chebyshev to get $(\epsilon, O(1))$-relative estimator

$$\mathbb{E}[Y_{\text{avg}}] = \frac{1}{d+1}$$

$$\text{Var}[Y_{\text{avg}}] \le \frac{1}{k(d+1)^2}$$

By Chebyshev's inequality:

$$\Pr[|Y_{\text{avg}} - \mathbb{E}[Y_{\text{avg}}]| \ge \epsilon \mathbb{E}[Y_{\text{avg}}]] \le \frac{\text{Var}[Y_{\text{avg}}]}{(\epsilon \mathbb{E}[Y_{\text{avg}}])^2} \le 1/k\epsilon^2$$

$$k = \frac{1}{4\epsilon^2}$$

- Run k independent copies of the estimator in parallel.
  - *Each run uses its own random hash function h_i.*
- Let $Y^{(1)}, \dots, Y^{(k)}$ be estimators from these k independent runs.
- Output $1/(Y_{\text{avg}}) - 1$ (where $Y_{\text{avg}} = (\sum_{i=1}^{k} Y^{(i)})/k$)

How to boost the accuracy? A FAMILIAR RECIPE

- **(Averaging)** Take average of $k = O(1/\epsilon^2)$ independent estimators to reduce variance
  - Apply Chebyshev to get $(\epsilon, O(1))$-relative estimator

$$\mathbb{E}[Y_{\text{avg}}] = \frac{1}{d+1}$$

$$\text{Var}[\text{Y}_{\text{avg}}] \le \frac{1}{k(d+1)^2}$$

By Chebyshev's inequality:

$$\Pr[|Y_{\text{avg}} - \mathbb{E}[Y_{\text{avg}}]| \ge \epsilon\mathbb{E}[Y_{\text{avg}}]] \le \frac{\text{Var}[Y_{\text{avg}}]}{(\epsilon\mathbb{E}[Y_{\text{avg}}])^2} \le 1/k\epsilon^2$$

$$k = \frac{1}{4\epsilon^2}$$

What does it imply for our final estimate?

$$\Pr\left[Y_{\text{avg}} \in \left(\frac{1-\epsilon}{d+1}, \frac{1+\epsilon}{d+1}\right)\right] \ge 3/4$$

$\Rightarrow \frac{1}{Y_{\text{avg}}} - 1 \in \left(\frac{d+1}{1+\epsilon} - 1, \frac{d+1}{1-\epsilon} - 1\right)$ w.p. at least $3/4$

How to boost the accuracy? A FAMILIAR RECIPE

- **(Averaging)** Take average of $k = O(1/\epsilon^2)$ independent estimators to reduce variance
  - Apply Chebyshev to get $(\epsilon, O(1))$-relative estimator
- **(Median trick)** Use $\ell = O(\log 1/\delta)$ of these averaged estimators and return their median to get $O(\epsilon, \delta)$-relative estimator

○ Repeat $O(\log 1/\delta)$ times

○ Run k independent copies of the estimator in parallel.
  - *Each run uses its own random hash function* h_i.

○ Let $Y^{(1)}, \ldots, Y^{(k)}$ be estimators from these k independent runs.

○ Output $1/(Y_{\text{avg}}) - 1$ (where $Y_{\text{avg}} = (\sum_{i=1}^{k} Y^{(i)})/k$)

○ Output the **median** of the estimators

Practical Considerations: Implementing Hash Functions

- So far, we assume access to a fully random hash function $h: [n] \rightarrow [0,1]$.

How to implement it?

- ❑ Use $h: [n] \rightarrow [m]$ for sufficiently large value of $m = \text{poly}(n)$
- ❑ Use pairwise independent hash families $\mathcal{H}$

Hashing and its role in Streaming

Pairwise Independent Hash Functions

A family $\mathcal{H} = \{h: [n] \to [m]\}$ is pairwise-independent or strongly 2-universal if,

- $\forall x \neq y \in [n], i \neq j \in [m]$: $\Pr_{h \sim \mathcal{H}}[h(x) = i \wedge h(y) = j] = 1/m^2$

What about uniformity? Is such hash function uniform over $[m]$ too?
I.e. Does the following hold?

$$\Pr_{h \sim \mathcal{H}}[h(x) = i] = 1/m$$

Pairwise Independent Hash Functions

A family $\mathcal{H} = \{h: [n] \to [m]\}$ is pairwise-independent or strongly 2-universal if,

- $\forall x \neq y \in [n], i \neq j \in [m]$: $\Pr_{h \sim \mathcal{H}}[h(x) = i \wedge h(y) = j] = 1/m^2$

Construction) Let p be a prime $\in [n, 2n]$. For any $\boldsymbol{a}, \boldsymbol{b} \in \{0, \dots, p-1\}$, define:

- $h_{\boldsymbol{a},\boldsymbol{b}}(x) = (\boldsymbol{a}x + \boldsymbol{b})\ mod\ p$
- The collection of $\mathcal{H} = \{h_{\boldsymbol{a},\boldsymbol{b}} \mid \boldsymbol{a}, \boldsymbol{b} \in [0, p-1]\}$ is pairwise independent

Space complexity for $\mathcal{H}$) a hash function from the family can be specified by three strings of length $\log p = O(\log n)$ to represent $\boldsymbol{a}, \boldsymbol{b}$ and p.

Simialr construction led to k-wise independence with $O(k \log n)$ space representation

Flajolet-Martin (LogLog)

$h(e_1)$	00101101000111011101
$h(e_2)$	11000011011110001000
$h(e_3)$	01101000000100110001
$h(e_4)$	01111001000100111110
$h(e_5)$	00101111011110111000
$h(e_6)$	10111000000101100000

$h: [n] \to [2^L]$ where $L = \lceil \log n \rceil$

Flajolet-Martin (LogLog)

$h(e_1)$	00101101000111011101
$h(e_2)$	11000011011110001000
$h(e_3)$	01101000000100110001
$h(e_4)$	01111001000100111110
$h(e_5)$	00101111011110111000
$h(e_6)$	10111000000101100000

$h: [n] \to [2^L]$ where $L = \lceil \log n \rceil$

$\Pr[h(e_i)$ has $\boldsymbol{Z} = s$ trailing zeros$] = 1/2^s$

In particular,

$\Pr[h(e_i)$ has $\boldsymbol{Z} = \log d$ trailing zeros$] = 1/d$

So, with d distinct hash values (i.e., items), we expect to see one with $\log d$ trailing zeros

Estimate number of distinct elements based on maximum number of trailing zeros.

The more distinct hash values we see, the higher we expect this maximum to be.

Rough Analysis

- If we had truly random h, the same analysis would work here too.
- [Alon, Matias, Szegedy'99] proved that pairwise independence suffices.

How?

- Define $X_{e,r}$ be the indicator r.v. that $h(e)$ has $\geq r$ trailing zeros.
- $Y_r = X_{e_1,r} + \cdots + X_{e_n,r}$
- $\{Y_r \geq 1 \Leftrightarrow \boldsymbol{Z} \geq r\}$ and $\{Y_r = 0 \Leftrightarrow \boldsymbol{Z} \leq r - 1\}$
- For any $r \in [L]$, $\mathbb{E}[Y_r] = \frac{d}{2^r}$ and $\mathrm{Var}[Y_r] = \frac{d}{2^r}\left(1 - \frac{1}{2^r}\right)$
- With probability $\geq \frac{1}{2}$, $d - 2 \leq Z \leq d + 2$