

COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2026.

Lecture 9

- Problem Set 2 is due Friday 10/16 at 11:59pm.
- The midterm is Monday, 10/19 in class.
- No class this upcoming Monday. Quiz due Wednesday at 9am instead.
- At least part of next Wednesday's class will be midterm review.
- Material from today may be on the midterm.

Summary

Last Class:

- Finish up Count-min sketch and frequent items counting.
- Distinct elements counting in streams via MinHashing.
- Track minimum hash value of incoming items and use it as a proxy for the number of distinct items.

This Class:

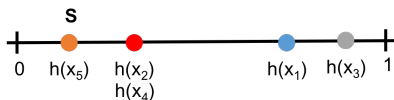
- Analysis of MinHashing.
- Boosting success probability with the median trick.
- Start on the similarity search problem and locality sensitive hashing.

Distinct Elements Counting

Hashing for Distinct Elements

Min-Hashing for Distinct Elements:

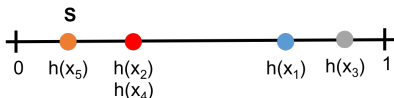
- Let $h : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \dots, n$
 - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



- After all items are processed, s is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.
- Intuition: The larger d is, the smaller we expect s to be.

Performance in Expectation

s is the minimum of d points chosen uniformly at random on $[0, 1]$.
Where $d = \#$ distinct elements.



$$\mathbb{E}[s] = \frac{1}{d+1} \text{ (using } \mathbb{E}(s) = \int_0^\infty \Pr(s > x) dx \text{) + calculus)}$$

- So our estimate $\hat{d} = \frac{1}{s} - 1$ is correct if s exactly equals its expectation. Does this mean $\mathbb{E}[\hat{d}] = d$? No, but:
- **Approximation is robust:** if $|s - \mathbb{E}[s]| \leq \epsilon \cdot \mathbb{E}[s]$ for any $\epsilon \in (0, 1/2)$ then:

$$(1 - 4\epsilon)d \leq \hat{d} \leq (1 + 4\epsilon)d$$

Initial Concentration Bound

So question is how well $\mathbf{s}$ concentrates around its mean.

$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ and } \text{Var}[\mathbf{s}] \leq \frac{1}{(d+1)^2} \text{ (also via calculus).}$$

Chebyshev's Inequality:

$$\Pr[|\mathbf{s} - \mathbb{E}[\mathbf{s}]| \geq \epsilon \mathbb{E}[\mathbf{s}]] \leq \frac{\text{Var}[\mathbf{s}]}{(\epsilon \mathbb{E}[\mathbf{s}])^2} = \frac{1}{\epsilon^2}.$$

Bound is vacuous for any $\epsilon < 1$. How can we improve accuracy?

$\mathbf{s}$: minimum of d distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\hat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$: estimate of # distinct elements d .

Improving Performance

Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $h : U \rightarrow [0, 1]$ be a random hash function
Let $h_1, h_2, \dots, h_k : U \rightarrow [0, 1]$ be random hash functions
- $s := 1$
- $s_1, s_2, \dots, s_k := 1$
- For $i = 1, \dots, n$
 - $s := \min(s, h(x_i))$
 - For $j=1, \dots, k, s_j := \min(s_j, h_j(x_i))$
- $s := \frac{1}{k} \sum_{j=1}^k s_j$
- Return $\hat{d} = \frac{1}{s} - 1$



Analysis

$\mathbf{s} = \frac{1}{k} \sum_{j=1}^k \mathbf{s}_j$. Have already shown that for $j = 1, \dots, k$:

$$\mathbb{E}[\mathbf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\text{Var}[\mathbf{s}_j] \leq \frac{1}{(d+1)^2} \implies \text{Var}[\mathbf{s}] \leq \frac{1}{k \cdot (d+1)^2} \text{ (linearity of variance)}$$

Chebyshev Inequality:

$$\Pr[|\mathbf{s} - \mathbb{E}[\mathbf{s}]| \geq \epsilon \mathbb{E}[\mathbf{s}]] \leq \frac{\text{Var}[\mathbf{s}]}{(\epsilon \mathbb{E}[\mathbf{s}])^2} = \frac{\mathbb{E}[\mathbf{s}]^2/k}{\epsilon^2 \mathbb{E}[\mathbf{s}]^2} = \frac{1}{k \cdot \epsilon^2} = \frac{\epsilon^2 \cdot \delta}{\epsilon^2} = \delta.$$

How should we set k if we want an error with probability at most δ ?

$$k = \frac{1}{\epsilon^2 \cdot \delta}.$$

$\mathbf{s}_j$: minimum of d distinct hashes chosen randomly over $[0, 1]$. $\mathbf{s} = \frac{1}{k} \sum_{j=1}^k \mathbf{s}_j$.
 $\hat{d} = \frac{1}{\mathbf{s}} - 1$: estimate of # distinct elements d .

Space Complexity

Hashing for Distinct Elements:

- Let $h_1, h_2, \dots, h_k : U \rightarrow [0, 1]$ be random hash functions
- $s_1, s_2, \dots, s_k := 1$
- For $i = 1, \dots, n$
 - For $j=1, \dots, k$, $s_j := \min(s_j, h_j(x_i))$
- $s := \frac{1}{k} \sum_{j=1}^k s_j$
- Return $\hat{d} = \frac{1}{s} - 1$



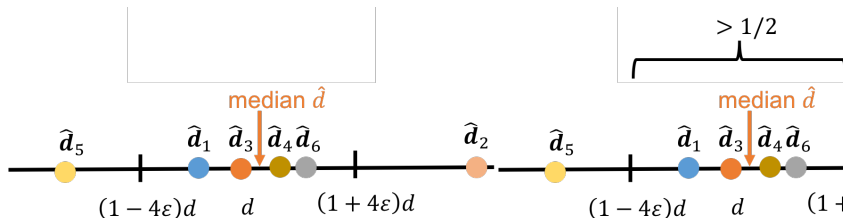
- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\hat{d}$ with $|d - \hat{d}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$.
- Space complexity is $k = \frac{1}{\epsilon^2 \cdot \delta}$ real numbers $s_1, \dots, s_k$.
- $\delta = 5\%$ failure rate gives a factor 20 overhead in space complexity.

Improved Failure Rate

How can we improve our dependence on the failure rate δ ?

The median trick: Run $t = O(\log 1/\delta)$ trials each with failure probability $\delta' = 1/4$ – each using $k = \frac{1}{\delta'\epsilon^2} = \frac{4}{\epsilon^2}$ hash functions.

- Letting $\hat{d}_1, \dots, \hat{d}_t$ be the outcomes of the t trials, return $\hat{d} = \text{median}(\hat{d}_1, \dots, \hat{d}_t)$.



- If $> 1/2$ of trials fall in $[(1-4\epsilon)d, (1+4\epsilon)d]$, then the median will.
- Have $< 1/2$ of trials on both the left and right.

The Median Trick

- $\hat{\mathbf{d}}_1, \dots, \hat{\mathbf{d}}_t$ are the outcomes of the t trials, each falling in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$ with probability at least $3/4$.
- $\hat{\mathbf{d}} = \text{median}(\hat{\mathbf{d}}_1, \dots, \hat{\mathbf{d}}_t)$.

What is the probability that the median $\hat{\mathbf{d}}$ falls in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$?

- Let X be the # of trials falling in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$.
 $\mathbb{E}[X] = \frac{3}{4} \cdot t$.

$$\Pr(\hat{\mathbf{d}} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d]) \leq \Pr\left(X < \frac{1}{2} \cdot t \cdot \frac{2}{3} \cdot \mathbb{E}[X]\right) \leq \Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{3}\mathbb{E}[X]\right)$$

Apply Chernoff bound:

$$\Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{3}\mathbb{E}[X]\right) \leq 2 \exp\left(-\frac{\frac{1}{3}^2 \cdot \frac{3}{4}t}{2 + 1/3}\right) = O(e^{-ct}).$$

- Setting $t = O(\log(1/\delta))$ gives failure probability $e^{-\log(1/\delta)} = \delta$.

Median Trick

Upshot: The median of $t = O(\log(1/\delta))$ independent runs of the hashing algorithm for distinct elements returns $\hat{d} \in [(1 - 4\epsilon)d, (1 + 4\epsilon)d]$ with probability at least $1 - \delta$.

Total Space Complexity: t trials, each using $k = \frac{1}{\epsilon^2 \delta'}$ hash functions, for $\delta' = 1/4$. Space is $\frac{4t}{\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ real numbers (the minimum value of each hash function).

No dependence on the number of distinct elements d or the number of items in the stream n ! Both of these numbers are typically very large.

A note on the median: The median is often used as a robust alternative to the mean, when there are outliers (e.g., heavy tailed distributions, corrupted data).

Distinct Elements in Practice

Our algorithm uses continuous valued fully random hash functions.
Can't be implemented...

- The idea of using the minimum hash value of $x_1, \dots, x_n$ to estimate the number of distinct elements naturally extends to when the hash functions map to discrete values.
- Flajolet-Martin (LogLog) algorithm and **HyperLogLog**.

$h(x_1)$	1010010	$h(x_1)$	1010010
$h(x_2)$	1001100	$h(x_2)$	1001100
$h(x_3)$	1001110	$h(x_3)$	1001110
⋮		⋮	
$h(x_n)$	1011000	$h(x_n)$	1011000

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

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

LogLog Counting of Distinct Elements

Flajolet-Martin (LogLog) algorithm and HyperLogLog.

$h(x_1)$	1010010
$h(x_2)$	1001100
$h(x_3)$	1001110
⋮	
$h(x_n)$	1011000

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

With d distinct elements, what do we expect m to be? $O(\log d)$.

a) $O(1)$ b) $O(\log d)$ c) $O(\sqrt{d})$ d) $O(d)$

$$\Pr(h(x_i) \text{ has } m \text{ trailing zeros}) = \frac{1}{2^m}.$$

- So with d distinct hashes, expect to see $\frac{d}{2^m}$ with m trailing zeros.
- Setting $m \approx \log d$ means we expect roughly one hash to have $\log d$ trailing zeros, i.e. $m \approx \log d$.

LogLog Space Guarantees

Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

$$\begin{aligned}\text{space used} &= O\left(\frac{\log \log d}{\epsilon^2}\right) \\ &= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} \text{ bits}^1 \\ &= \frac{1.04 \cdot 5}{.02^2} = 13000 \text{ bits} \approx 1.6 \text{ kB!}\end{aligned}$$

Mergeable Sketch: Consider the case (essentially always in practice) that the items are processed on different machines.

- Given data structures (sketches) $HLL(x_1, \dots, x_n)$, $HLL(y_1, \dots, y_n)$ is easy to merge them to give $HLL(x_1, \dots, x_n, y_1, \dots, y_n)$. **How?**
- Set the maximum # of trailing zeros to the maximum in the two sketches.

1. 1.04 is the constant in the HyperLogLog analysis. Not important!

HyperLogLog In Practice

Implementations: Google PowerDrill, Facebook Presto, Twitter Algebird, Amazon Redshift.

Use Case: Exploratory SQL-like queries on tables with 100s billions of rows. ~ 5 million count distinct queries per day. E.g.,

- **Count** number of **distinct** users in Germany that made at least one search containing the word 'auto' in the last month.
- **Count** number of **distinct** subject lines in emails sent by users that have registered in the last week, in comparison to number of emails sent overall (to estimate rates of spam accounts).

Traditional *COUNT*, *DISTINCT* SQL calls are far too slow, especially when the data is distributed across many servers.

Questions?