COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Spring 2026.
Lecture 8

Logistics

- Problem Set 2 is due Sunday 3/8 at 11:59pm.
- Problem Set 1 grades will be released later today.
- Midterm is **in class** next Thursday, 3/12.
- Study guide is posted on the course webpage (under the midterm row in the schedule tab).
- Past midterms are posted in Canvas. Note that some cover a bit more material than we have seen – e.g., we will not cover the Johnson-Lindenstrauss lemma or low-distortion embeddings before Midterm 1.
- At least some of next Tuesday's lecture will be for midterm review. I will also hold additional review office hours next **Wednesday, 11am-12pm**.
- If you need extended time on the midterm, please reach out by this Friday at the latest to make a plan.

Summary

Last Class:

- Overview of streaming algorithms.
- The (ϵ, k)-frequent items problem and its applications.
- Count-Min sketch algorithm for frequent items.

This Class:

- Distinct items counting via min-hashing.
- Success probability boosting via the median trick.

Distinct Elements

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Applications:

- Distinct IP addresses clicking on an ad or visiting a site.
- Distinct values in a database column (for estimating sizes of joins and group bys).
- Number of distinct search engine queries.
- Counting distinct motifs in large DNA sequences.

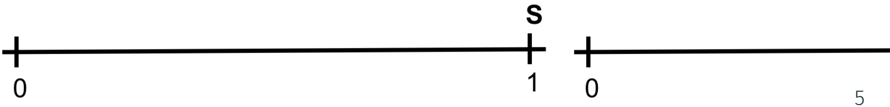
Google Sawzall, Facebook Presto, Apache Drill, Twitter Algebird

Hashing for Distinct Elements

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

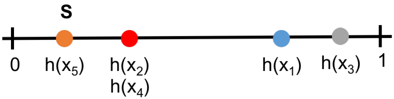
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



Hashing for Distinct Elements

Min-Hashing for Distinct Elements:

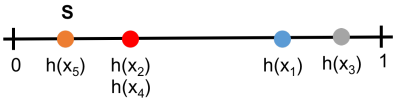
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



- 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.
- Same idea as Flajolet-Martin algorithm and HyperLogLog, except they use discrete hash functions.

Performance in Expectation

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



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

- So our estimate $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ is correct if **s** exactly equals its expectation. Does this mean $\mathbb{E}[\widehat{\mathbf{d}}] = d$? No, but:
- **Approximation is robust:** if $|\mathbf{s} - \mathbb{E}[\mathbf{s}]| \leq \epsilon \cdot \mathbb{E}[\mathbf{s}]$ for any $\epsilon \in (0, 1/2)$ and a small constant $c \leq 4$:

$$(1 - c\epsilon)d \leq \widehat{\mathbf{d}} \leq (1 + c\epsilon)d$$

.

Initial Concentration Bound

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

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

Chebyshev's Inequality:

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

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

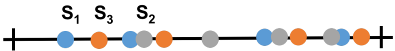
$\mathbf{s}$: minimum of d distinct hashes chosen randomly over $[0,1]$, computed by hashing algorithm. $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$: estimate of # distinct elements d.

Improving Performance

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

Hashing for Distinct Elements (Improved):

- Let $\mathbf{h} : U \to [0,1]$ be a random hash functionLet $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0,1]$ be random hash functions
- $\mathbf{s} := 1$
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
  - For j=1,...,k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



Analysis

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

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

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

Chebyshev Inequality:

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

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

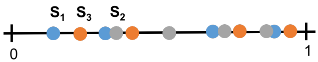
$k = \frac{1}{\epsilon^2 \cdot \delta}$.

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

Space Complexity

Hashing for Distinct Elements:

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0,1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
    - For j=1,..., k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



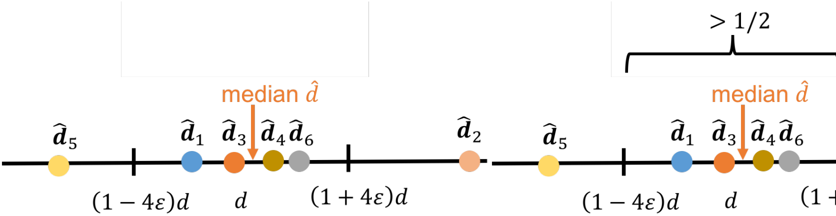
- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\widehat{\mathbf{d}}$ with $|d - \widehat{\mathbf{d}}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$.
- Space complexity is $k = \frac{1}{\epsilon^2 \cdot \delta}$ real numbers $s_1, \ldots, s_k$.
- $\delta = 5\%$ 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 $\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t$ be the outcomes of the t trials, return $\widehat{\mathbf{d}} = median(\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t)$.



- 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

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

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

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

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

Apply Chernoff bound:

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

- 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 $\widehat{\mathbf{d}} \in [(1-4\epsilon)d, (1+4\epsilon)d]$ 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 $x_1, \ldots, x_n$ 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, roughly what do we expect **m** to be?

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

$$\Pr(\mathbf{h}(x_i) \text{ has } x \log d \text{ trailing zeros}) = \frac{1}{2^{x \log d}} = \frac{1}{d}.$$

So with d distinct hashes, expect to see 1 with $\log d$ trailing zeros.

Expect $\mathbf{m} \approx \log d$. **m** takes $\log \log d$ bits to store.

Total Space: $O\left(\frac{\log \log d}{\epsilon^2}\right)$ for an ϵ approximate count.

LogLog Space Guarantees

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

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

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

- Given data structures (sketches) $HLL(x_1, \ldots, x_n)$, $HLL(y_1, \ldots, y_n)$ is is easy to merge them to give $HLL(x_1, \ldots, x_n, y_1, \ldots, y_n)$. 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!