# COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2023.

Lecture 9

## Logistics

- Problem Set 2 is due in roughly 2 weeks – Monday, 10/16 at 11:59pm.
- The midterm is the following Tuesday, 10/24 in class. I will post review material later this week.
- There will be no quizzes due 10/16 or 10/23.
- Next week I am out of town for a conference. There is no class on Tuesday. Thursday, class will be held virtually. Virtual office hours TBD.
- This Thursday I will hold additional in person office hours after class to answer questions on Problem Set 2.
- The class will be filmed this Thursday. The footage will be 'b-roll' with no sound. Let me know if you do not feel comfortable being filmed. Show up 10-15 minutes early to participate in some staged shots (and get two bonus points on the quiz).

## Summary

Last Class:

- Distinct elements counting in streams vis MinHashing.
- The Median Trick to boost success probability.

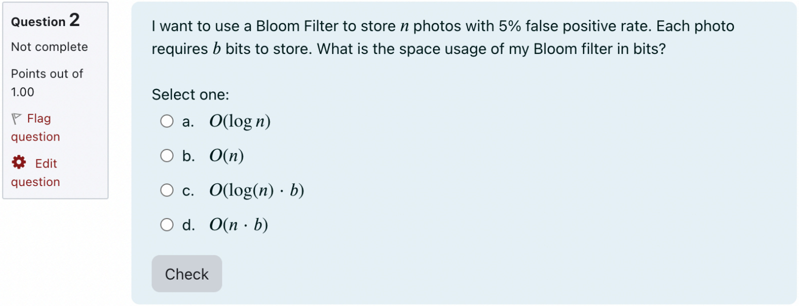
This Class:

- High-level overview of practical distinct elements algorithms.
- Introduction of Jaccard similarity and the similarity search problem.
- Locality sensitive hashing for fast similarity search.
- MinHashing for Jaccard similarity search.

## Quiz

- Many people reported being most interested in Bloom Filters and their applications. I'll try to keep highlighting real world applications moving forward.
- People are a bit concerned/confused about applying concentration bounds, especially exponential concentration bounds. There will be some more practice for this on Problem Set 2, and I will make sure to include plenty of practice questions on this for the midterm review material.

# Quiz



**Question 2**

Not complete

Points out of 1.00

Flag question

Edit question

I want to use a Bloom Filter to store $n$ photos with 5% false positive rate. Each photo requires $b$ bits to store. What is the space usage of my Bloom filter in bits?

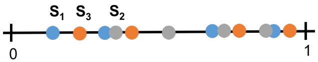
Select one:

- ○ a. $O(\log n)$
- ○ b. $O(n)$
- ○ c. $O(\log(n) \cdot b)$
- ○ d. $O(n \cdot b)$

## 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 $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

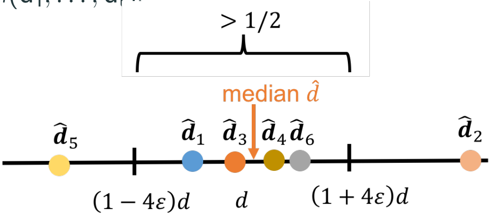


- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\widehat{\mathbf{d}}$ with $|d - \widehat{\mathbf{d}}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$.
- Space complexity is $k = \frac{1}{\epsilon^2 \cdot \delta}$ real numbers $s_1, \ldots, s_k$.
- Can be improved to $O(\log(1/\delta)/\epsilon^2)$ via the median trick.

## Improved Failure Rate

**The median trick:** Run $t = O(\log 1/\delta)$ trials each with failure probability $\delta' = 1/5$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{5}{\epsilon^2}$ hash functions.

- Letting $\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t$ be the outcomes of the $t$ trials, return $\widehat{\mathrm{d}} = \mathit{median}(\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t)$.

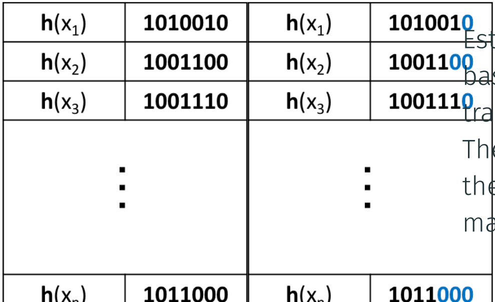


- We expect $4/5$ of the estimates to fall in $[(1-4\epsilon)d, (1+4\epsilon)d]$.
- If $> 1/2$ of estimates fall in $[(1-4\epsilon)d, (1+4\epsilon)d]$, then the median will.
- Can prove this will happen with probability at least $1-\delta$ via a Chernoff bound.

## 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 $\epsilon$ 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!

## 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. $\sim$ 5 million count distinct queries per day. E.g.,

- Count number if 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).

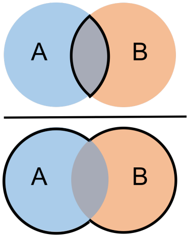
Traditional *COUNT*, *DISTINCT* SQL calls are far too slow, especially when the data is distributed across many servers.

Questions?

## Another Fundamental Problem

**Jaccard Index:** A similarity measure between two sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{\# shared elements}}{\text{\# total elements}}.$$



Natural measure for similarity between bit strings – interpret an $n$ bit string as a set, containing the elements corresponding the positions of its ones. $J(x,y) = \frac{\text{\# shared ones}}{\text{total ones}}$.

## Search with Jaccard Similarity

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\#\text{ shared elements}}{\#\text{ total elements}}.$$

Want Fast Implementations For:

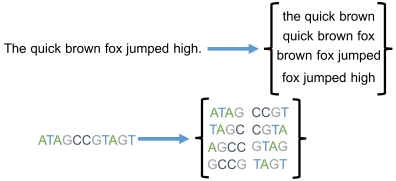
- **Near Neighbor Search:** Have a database of $n$ sets/bit strings and given a set $A$, want to find if it has high Jaccard similarity to anything in the database. $\Omega(n)$ time with a linear scan.
- **All-pairs Similarity Search:** Have $n$ different sets/bit strings and want to find all pairs with high Jaccard similarity. $\Omega(n^2)$ time if we check all pairs explicitly.

Will speed up via randomized locality sensitive hashing.

## Application: Document Similarity

**Document Similarity:**

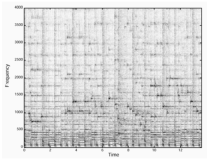
- E.g., to detect plagiarism, copyright infringement, duplicate webpages, spam.
- Use Shingling + Jaccard similarity. ($n$-grams, $k$-mers)

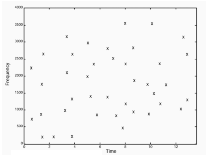


## Application: Audio Search

Audio Fingerprinting:

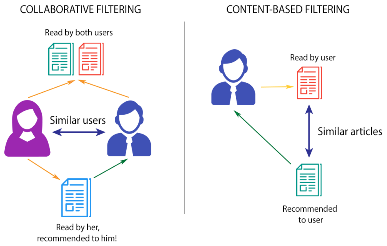
- E.g., in audio search (Shazam), Earthquake detection.
- Represent sound clip via a binary ‘fingerprint’ then compare with Jaccard similarity.





## Application: Collaborative Filtering

Online recommendation systems are often based on **collaborative filtering**. Simplest approach: find similar users and make recommendations based on those users.

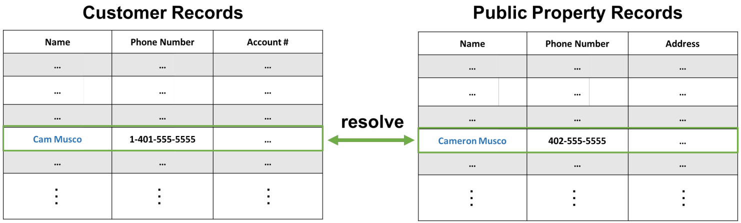


- Twitter: represent a user as the set of accounts they follow. Match users based on the Jaccard similarity of these sets. Recommend that you follow accounts followed by similar users.
- Netflix: look at sets of movies watched. Amazon: look at products purchased, etc.

## Application: Entity Resolution

**Entity Resolution Problem:** Want to combine records from multiple data sources that refer to the same entities.

- E.g. data on individuals from voting registrations, property records, and social media accounts. Names and addresses may not exactly match, due to typos, nicknames, moves, etc.
- Still want to match records that all refer to the same person using all pairs similarity search.



**Customer Records**

| Name | Phone Number | Account # |
|---|---|---|
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| Cam Musco | 1-401-555-5555 | ... |
| ... | ... | ... |
| ⋮ | ⋮ | ⋮ |

**Public Property Records**

| Name | Phone Number | Address |
|---|---|---|
| ... | ... | ... |
| ... | ... | ... |
| ... | ... | ... |
| Cameron Musco | 402-555-5555 | ... |
| ... | ... | ... |
| ⋮ | ⋮ | ⋮ |

See Section 3.8.2 of *Mining Massive Datasets* for a discussion of a real world example involving 1 million customers. Naively this would

## Application: Spam and Fraud Detection

Many applications to spam/fraud detection. E.g.

- Fake Reviews: Very common on websites like Amazon. Detection often looks for (near) duplicate reviews on similar products, which have been copied. 'Near duplicate' measured with shingles + Jaccard similarity.
- Lateral phishing: Phishing emails sent to addresses at a business coming from a legitimate email address at the same business that has been compromised.
    - One method of detection looks at the recipient list of an email and checks if it has small Jaccard similarity with any previous recipient lists. If not, the email is flagged as possible spam.