COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Spring 2020.

Lecture 5

LOGISTICS

- Problem Set 1 was released last Thursday and is due Friday 2/14 at 8pm in Gradescope. Don't leave until the last minute.
- There is **no class** this Thursday.

Last Class We Covered:

Last Class We Covered:

- Bloom Filters:
  - Random hashing to maintain a large set in very small space.
  - Discussed applications and how the false positive rate is determined.

Last Class We Covered:

- **Bloom Filters:**
  - Random hashing to maintain a large set in very small space.
  - Discussed applications and how the false positive rate is determined.
- **Streaming Algorithms and Distinct Elements:**
  - Started on streaming algorithms and one of the most fundamental examples: estimating the number of distinct items in a data stream.
  - Introduced an algorithm for doing this via a min-of-hashes approach.

Finish Distinct Elements:

- Finish hashing-based distinct elements algorithm. Learn the 'median trick' to boost accuracy.
- Discuss variants and practical implementations.

Finish Distinct Elements:

- Finish hashing-based distinct elements algorithm. Learn the ‘median trick’ to boost accuracy.
- Discuss variants and practical implementations.

MinHashing For Set Similarity:

- See how a min-of-hashes approach (MinHash) is used to estimate the overlap between two bit vectors.
- A key idea behind audio fingerprint search (Shazam), document search (plagiarism and copyright violation detection), recommendation systems, etc.

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

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)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

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

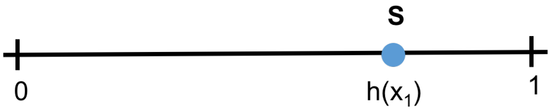
Hashing for Distinct Elements (variant of Flajolet-Martin):

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

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

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

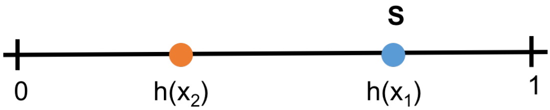
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



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

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

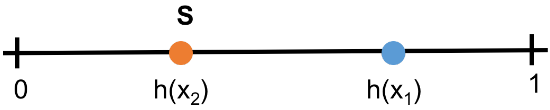
- Let $\mathbf{h} : U \to [0,1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



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

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

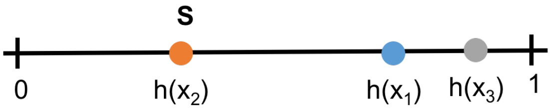
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



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

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

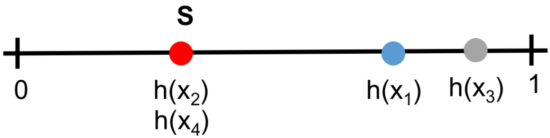
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



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

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

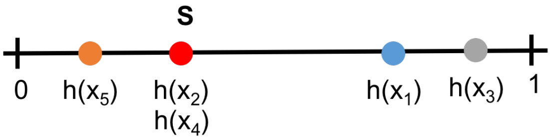
- Let $\mathbf{h} : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



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

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

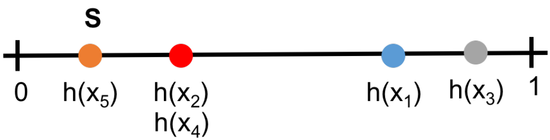
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



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

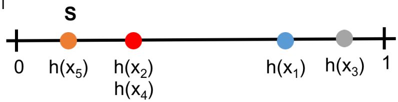
Hashing for Distinct Elements (variant of Flajolet-Martin):

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



Hashing for Distinct Elements:

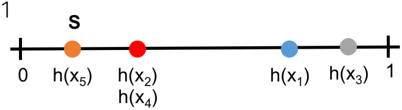
- Let $\mathbf{h} : U \to [0,1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



- After all items are processed, $\mathbf{s}$ 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 $\mathbf{s}$ to be.

Hashing for Distinct Elements:

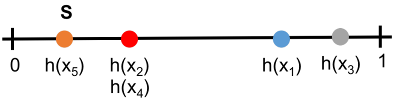
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



- After all items are processed, $\mathbf{s}$ 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 $\mathbf{s}$ to be.
- Same idea as Flajolet-Martin algorithm and HyperLogLog, except they use discrete hash functions.

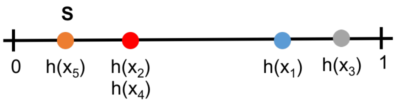
Hashing for Distinct Elements:

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

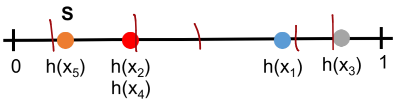


- After all items are processed, $\mathbf{s}$ is the minimum of d points chosen uniformly at random on $[0,1]$. Where $d = \#$ distinct elements.
- **Intuition:** The larger d is, the smaller we expect $\mathbf{s}$ to be.
- Same idea as Flajolet-Martin algorithm and HyperLogLog, except they use discrete hash functions.
- **Notice:** Output does not depend on n at all.

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

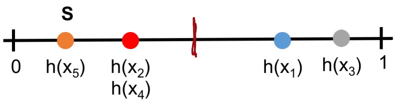


s is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.



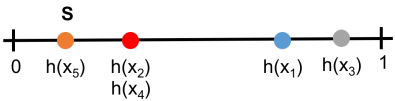
$\mathbb{E}[s] =$

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



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

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

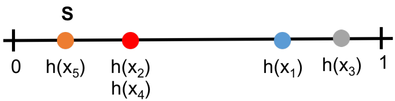


$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (using } \mathbb{E}(\mathbf{s}) = \int_0^\infty \Pr(\mathbf{s} > x)dx) + \text{calculus)}$$

- So estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm is correct if **s** exactly equals its expectation.

$\tilde{d} = \frac{1}{1/d+1} - 1 = d$

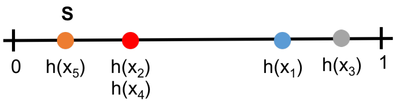
$\mathbf{s}$ is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.



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

- So estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm is correct if $\mathbf{s}$ exactly equals its expectation. Does this mean $\mathbb{E}[\widehat{\mathbf{d}}] = d$?

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

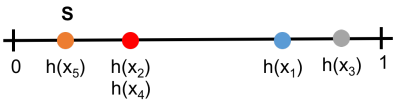


$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (using } \mathbb{E}(\mathbf{s}) = \int_0^\infty \Pr(\mathbf{s} > x)dx) + \text{calculus)}$$

- So estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm is correct if **s** exactly equals its expectation. Does this mean $\mathbb{E}[\widehat{\mathbf{d}}] = d$? No, but:

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 estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm 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 $\epsilon \in (0, 1/2)$:

$$(1-2\epsilon)d \leq \widehat{\mathbf{d}} \leq (1+4\epsilon)d$$

.

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

$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1}$$

.

.

$\mathbf{s}$: 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.

So question is how well s concentrates around its mean.

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

.

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

INITIAL CONCENTRATION BOUND

So question is how well **s** concentrates around its mean.

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

Chebyshev's Inequality:

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

> **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.

So question is how well s concentrates around its mean.

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

Chebyshev's Inequality:

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

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

So question is how well s concentrates around its mean.

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

Chebyshev's Inequality:

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

Bound is vacuous for any $\epsilon < 1$.

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

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

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

Chebyshev's Inequality:

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

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.

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

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

Hashing for Distinct Elements (Improved):

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

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

Hashing for Distinct Elements (Improved):

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

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

Hashing for Distinct Elements (Improved):

- 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 $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

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

Hashing for Distinct Elements (Improved):

- 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, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

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

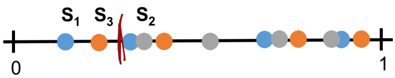
Hashing for Distinct Elements (Improved):

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \rightarrow [0, 1]$ 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 $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

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

Hashing for Distinct Elements (Improved):

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



$|s - \mathbb{E}s| \leq \epsilon \mathbb{E}s$

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

$$\mathbb{E}[s_j] = \frac{1}{d+1}$$

$$\mathrm{Var}[s_j] \leq \frac{1}{(d+1)^2}$$

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

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

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}]$$

$$\text{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2}$$

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.

ANALYSIS

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

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

$$\mathrm{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2}$$

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.

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

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

$$\text{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2} \implies \text{Var}[\mathsf{s}]$$

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.

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

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

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

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

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$. 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:

$$\Pr\left[|\mathsf{s} - \mathbb{E}[\mathsf{s}]| \geq \epsilon \mathbb{E}[\mathsf{s}]\right] \leq \frac{\mathrm{Var}[\mathsf{s}]}{(\epsilon \mathbb{E}[\mathsf{s}])^2}$$

$$\leq \frac{1/k(d+1)^2}{\varepsilon^2 \, 1/(d+1)^2} = \frac{1}{k\varepsilon^2}$$

> 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.

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

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

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

$$d = \frac{1}{s} - 1$$

Chebyshev Inequality:

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

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.

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

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

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

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathbf{d}}\right| \geq 4\epsilon \cdot d\right] \leq \frac{\text{Var}[\mathbf{s}]}{(\epsilon\mathbb{E}[\mathbf{s}])^2} = \frac{\mathbb{E}[\mathbf{s}]^2/k}{\epsilon^2\mathbb{E}[\mathbf{s}]^2} = \frac{1}{k \cdot \epsilon^2}$$

$$s \in \left[\frac{1}{d+1} - \frac{\epsilon}{d+1}, \frac{1}{d+1} + \frac{\epsilon}{d+1}\right] \qquad \hat{d} = \frac{1}{s} - 1$$

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

ANALYSIS

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

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

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

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathbf{d}}\right| \geq 4\epsilon \cdot d\right] \leq \frac{\text{Var}[\mathbf{s}]}{(\epsilon\mathbb{E}[\mathbf{s}])^2} = \frac{\mathbb{E}[\mathbf{s}]^2/k}{\epsilon^2\mathbb{E}[\mathbf{s}]^2} = \frac{1}{k \cdot \epsilon^2}$$

How should we set k if we want $4\epsilon \cdot d$ error with probability $\geq 1 - \delta$?

$K = \frac{1}{\epsilon\sqrt{\delta}}$

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

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

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

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

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathbf{d}}\right| \geq 4\epsilon \cdot d\right] \leq \frac{\mathrm{Var}[\mathbf{s}]}{(\epsilon \mathbb{E}[\mathbf{s}])^2} = \frac{\mathbb{E}[\mathbf{s}]^2/k}{\epsilon^2 \mathbb{E}[\mathbf{s}]^2} = \frac{1}{k \cdot \epsilon^2}$$

How should we set k if we want $4\epsilon \cdot d$ error with probability $\geq 1 - \delta$?

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

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

$\cdot \; \mathrm{Var}(s) = \mathrm{Var}\left(\frac{1}{k}\sum s_j\right) = \frac{1}{k^2}\mathrm{Var}\left(\sum s_j\right) = \frac{1}{k^2}\cdot k \cdot \frac{1}{(d+1)^2} = \frac{1}{k(d+1)^2}$

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

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

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

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathbf{d}}\right| \geq 4\epsilon \cdot d\right] \leq \frac{\mathrm{Var}[\mathbf{s}]}{(\epsilon\mathbb{E}[\mathbf{s}])^2} = \frac{\mathbb{E}[\mathbf{s}]^2/k}{\epsilon^2\mathbb{E}[\mathbf{s}]^2} = \frac{1}{k\cdot\epsilon^2} = \frac{\epsilon^2\cdot\delta}{\epsilon^2} = \delta.$$

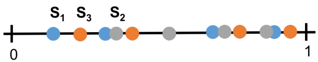
How should we set k if we want $4\epsilon \cdot d$ error with probability $\geq 1-\delta$?

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

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

Hashing for Distinct Elements:

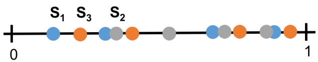
- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - For j=1,..., k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j$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

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

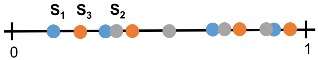


- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\widehat{\mathbf{d}}$ with $|d - \widehat{\mathbf{d}}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$.
- Space complexity is $k = \frac{1}{\epsilon^2 \cdot \delta}$ real numbers $s_1, \ldots, s_k$.

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 $|d - \widehat{\mathbf{d}}| \le 4\epsilon \cdot d$ with probability at least $1 - \delta$.

  $\frac{1}{\delta} = \frac{1}{.05} = 20$ $\quad \frac{20}{\epsilon^2} \quad \frac{100}{\epsilon^2}$

- 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.

How can we decrease the cost of a small failure rate δ?

How can we decrease the cost of a small failure rate δ?

One Thought: Apply stronger concentration bounds. E.g., replace Chebyshev with Bernstein.

How can we decrease the cost of a small failure rate δ?

One Thought: Apply stronger concentration bounds. E.g., replace Chebyshev with Bernstein. This won't work. Why?

How can we decrease the cost of a small failure rate δ?

One Thought: Apply stronger concentration bounds. E.g., replace Chebyshev with Bernstein. This won't work. Why?

Bernstein Inequality (Sample Mean): Consider independent random variables $\mathbf{X}_1, \ldots, \mathbf{X}_k$ all falling in $[-\bar{M}, \bar{M}]$ and let $\mathbf{X} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{X}_i$. Let $\mu = \mathbb{E}[\mathbf{X}]$ and $\bar{\sigma}^2 = \frac{1}{k}\mathrm{Var}[\mathbf{X}]$. For any $t \geq 0$:

$$\Pr\left(|\mathbf{X} - \mu| \geq t\right) \leq 2\exp\left(-\frac{t^2 k}{2\bar{\sigma}^2 + \frac{4}{3}\bar{M}t}\right).$$

How can we decrease the cost of a small failure rate δ?

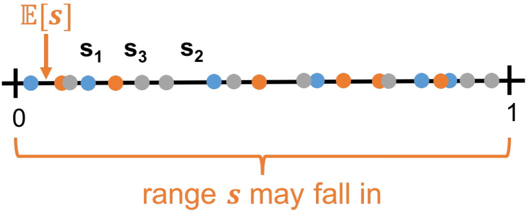
One Thought: Apply stronger concentration bounds. E.g., replace Chebyshev with Bernstein. This won't work. Why?

Bernstein Inequality (Sample Mean): Consider independent random variables $X_1, \ldots, X_k$ all falling in $[-\bar{M}, \bar{M}]$ and let $X = \frac{1}{k}\sum_{i=1}^{k} X_i$. Let $\mu = \mathbb{E}[X]$ and $\bar{\sigma}^2 = \frac{1}{k}\text{Var}[X]$. For any $t \geq 0$:

$$\Pr\left(|X - \mu| \geq t\right) \leq 2\exp\left(-\frac{t^2 k}{2\bar{\sigma}^2 + \frac{4}{3}\bar{M}t}\right).$$

For us, $t = \frac{\epsilon}{d}$ and $\bar{M} = 1$. So $\frac{t^2 k}{\frac{4}{3}\bar{M}t} = \frac{3\epsilon k}{4d}$. So if $k \ll d$ exponent has small magnitude (i.e., bound is bad).

Exponential tail bounds are weak for random variables with very large ranges compared to their expectation.



IMPROVED FAILURE RATE

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

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 $\delta' = 1/5$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{5}{\epsilon^2}$ hash functions.

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 $\delta' = 1/5$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{5}{\epsilon^2}$ hash functions.

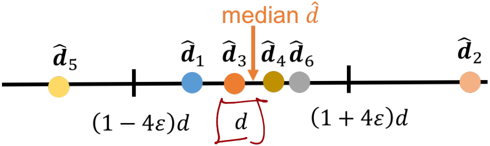
- Letting $\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t$ be the outcomes of the t trials, return $\widehat{\mathbf{d}} = \textit{median}(\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t)$.

t, K

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 $\delta' = 1/5$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{5}{\epsilon^2}$ hash functions.

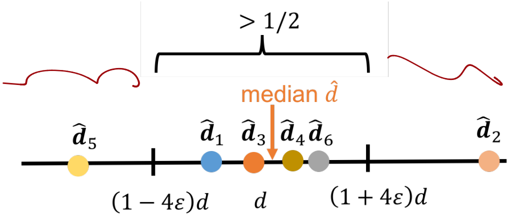
- Letting $\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t$ be the outcomes of the t trials, return $\widehat{\mathrm{d}} = median(\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t)$.



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 $\delta' = 1/5$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{5}{\epsilon^2}$ hash functions.

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

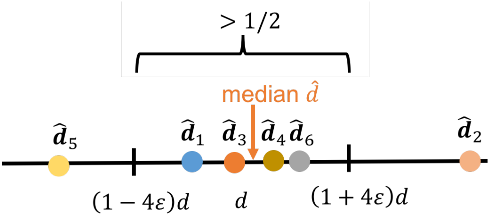


- If $> 1/2$ of trials fall in $[(1-4\epsilon)d, (1+4\epsilon)d]$, then the median will.

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 $\delta' = 1/5$ – 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 $\widehat{\mathrm{d}} = median(\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t)$.



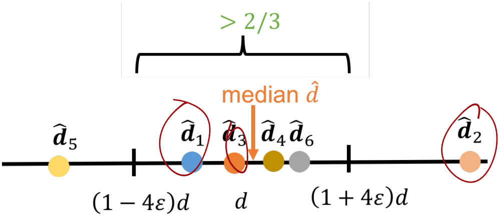
- If $> 1/2$ of trials fall in $[(1-4\epsilon)d, (1+4\epsilon)d]$, then the median will.
- Have $< 1/2$ of trials on both the left and right.

$t \cdot k$
hashes total

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 $\delta' = 1/5$ – 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 $\widehat{\mathrm{d}} = median(\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t)$.



- If $> 2/3$ of trials fall in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$, then the median will.
- Have $< 1/3$ of trials on both the left and right.

- $\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least $4/5$.
- $\widehat{\mathsf{d}} = median(\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t)$.

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

THE MEDIAN TRICK

- $\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least $4/5$.
- $\widehat{\mathrm{d}} = median(\widehat{\mathrm{d}}_1, \ldots, \widehat{\mathrm{d}}_t)$.

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

- Let X be the $\#$ of trials falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$. .

- $\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least $4/5$.
- $\widehat{\mathsf{d}} = median(\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t)$.

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 $[(1-4\epsilon)d, (1+4\epsilon)d]$. .

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

- $\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least $4/5$.
- $\widehat{\mathsf{d}} = median(\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t)$.

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 $[(1-4\epsilon)d, (1+4\epsilon)d]$. $\mathbb{E}[\mathsf{X}] = \frac{4}{5} \cdot t$

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

- $\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least 4/5.
- $\widehat{\mathsf{d}} = median(\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t)$.

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 $[(1-4\epsilon)d, (1+4\epsilon)d]$. $\mathbb{E}[\mathsf{X}] = \frac{4}{5} \cdot t$.

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

- $\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least 4/5.
- $\widehat{\mathsf{d}} = median(\widehat{\mathsf{d}}_1, \ldots, \widehat{\mathsf{d}}_t)$.

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 $[(1-4\epsilon)d, (1+4\epsilon)d]$. $\mathbb{E}[\mathsf{X}] = \frac{4}{5} \cdot t$.

$$\Pr\left(\widehat{\mathsf{d}} \notin [(1-4\epsilon)d, (1+4\epsilon)d]\right) \leq \Pr\left(\mathsf{X} < \frac{5}{6} \cdot \mathbb{E}[\mathsf{X}]\right)$$

- $\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least 4/5.
- $\widehat{\mathbf{d}} = median(\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t)$.

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

- Let $\mathbf{X}$ be the # of trials falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$. $\mathbb{E}[\mathbf{X}] = \frac{4}{5} \cdot t$.

$$\Pr\left(\widehat{\mathbf{d}} \notin [(1-4\epsilon)d, (1+4\epsilon)d]\right) \leq \Pr\left(\mathbf{X} < \frac{5}{6} \cdot \mathbb{E}[\mathbf{X}]\right) \leq \Pr\left(|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq \frac{1}{6}\mathbb{E}[\mathbf{X}]\right)$$

- $\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least 4/5.
- $\widehat{\mathbf{d}} = median(\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t)$.

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

- Let $\mathbf{X}$ be the # of trials falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$. $\mathbb{E}[\mathbf{X}] = \frac{4}{5} \cdot t$.

$$\Pr\left(\widehat{\mathbf{d}} \notin [(1-4\epsilon)d, (1+4\epsilon)d]\right) \leq \Pr\left(\mathbf{X} < \frac{5}{6} \cdot \mathbb{E}[\mathbf{X}]\right) \leq \Pr\left(|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq \frac{1}{6}\mathbb{E}[\mathbf{X}]\right)$$

Apply Chernoff bound:

- $\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least 4/5.
- $\widehat{\mathbf{d}} = median(\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t)$.

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

- Let $\mathbf{X}$ be the # of trials falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$. $\mathbb{E}[\mathbf{X}] = \frac{4}{5} \cdot t$.

$$\Pr\left(\widehat{\mathbf{d}} \notin [(1-4\epsilon)d, (1+4\epsilon)d]\right) \leq \Pr\left(\mathbf{X} < \frac{5}{6} \cdot \mathbb{E}[\mathbf{X}]\right) \leq \Pr\left(|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq \frac{1}{6}\mathbb{E}[\mathbf{X}]\right)$$

Apply Chernoff bound:

$$\Pr\left(|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq \frac{1}{6}\mathbb{E}[\mathbf{X}]\right) \leq 2\exp\left(-\frac{\frac{1}{6}^2 \cdot \frac{4}{5}t}{2+1/6}\right) = O\left(e^{-O(t)}\right).$$

THE MEDIAN TRICK

- $\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t$ are the outcomes of the t trials, each falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$ with probability at least $4/5$.
- $\widehat{\mathbf{d}} = median(\widehat{\mathbf{d}}_1, \ldots, \widehat{\mathbf{d}}_t)$.

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

- Let $\mathbf{X}$ be the $\#$ of trials falling in $[(1-4\epsilon)d, (1+4\epsilon)d]$. $\mathbb{E}[\mathbf{X}] = \frac{4}{5} \cdot t$.

$$\Pr\left(\widehat{\mathbf{d}} \notin [(1-4\epsilon)d, (1+4\epsilon)d]\right) \leq \Pr\left(\mathbf{X} < \frac{5}{6} \cdot \mathbb{E}[\mathbf{X}]\right) \leq \Pr\left(|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq \frac{1}{6}\mathbb{E}[\mathbf{X}]\right)$$

Apply Chernoff bound:

$$\Pr\left(|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq \frac{1}{6}\mathbb{E}[\mathbf{X}]\right) \leq 2\exp\left(-\frac{\frac{1}{6}^2 \cdot \frac{4}{5}t}{2 + 1/6}\right) = O\left(e^{-O(t)}\right).$$

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

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 $1-\delta$.

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

Total Space Complexity: t trials, each using $k = \frac{1}{\epsilon^2\delta'}$ hash functions, for $\delta' = 1/5$. Space is $\frac{5t}{\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ real numbers (the minimum value of each hash function).

$O\left(\frac{1}{\delta\epsilon^2}\right)$

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 $\delta' = 1/5$. Space is $\frac{5t}{\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ 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.

MEDIAN TRICK

$ES = \frac{1}{d+1}$

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 $\delta' = 1/5$. Space is $\frac{5t}{\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ 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).

Our algorithm uses continuous valued fully random hash functions.

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

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.

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.

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_2)	**1001100**
h(x_3)	**1001110**
⋮	
h(x_n)	**1011000**

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

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

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**
$h(x_2)$	**1001100**
$h(x_3)$	**1001110**
⋮	
$h(x_n)$	**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.

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**.

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?

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? lg(d)

$$\Pr(\mathbf{h}(x_i) \text{ has } x \text{ trailing zeros}) =$$

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?

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

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?

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

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?

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

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?

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

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

Expect $\mathbf{m} \approx \log d$.

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?

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

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.

Flajolet-Martin (LogLog) algorithm and HyperLogLog.

$\mathbf{h}(x_1)$	**1010010**
$\mathbf{h}(x_2)$	**1001100**
$\mathbf{h}(x_3)$	**1001110**
⋮	
$\mathbf{h}(x_n)$	**1011000**

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

$\frac{\log(1/\delta)}{\epsilon^2}$

With *d* distinct elements what do we expect **m** to be?

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

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

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

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

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?

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

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

Note: Careful averaging of estimates from multiple hash functions.

LOGLOG SPACE GUARANTEES

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

$$\text{space used } = O\left(\frac{\log\log d}{\epsilon^2} + \log d\right)$$

LOGLOG SPACE GUARANTEES

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

$$\begin{aligned} \text{space used } &= O\left(\frac{\log\log d}{\epsilon^2} + \log d\right) \\ &= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1 \end{aligned}$$

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

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

$$\begin{aligned} \text{space used} &= O\left(\frac{\log\log d}{\epsilon^2} + \log d\right) \\ &= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1 \\ &= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6\ kB! \end{aligned}$$

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

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

$$\begin{aligned}\text{space used } &= O\left(\frac{\log\log d}{\epsilon^2} + \log d\right)\\ &= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1\\ &= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6\ kB!\end{aligned}$$

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

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

LOGLOG SPACE GUARANTEES

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

$$\text{space used} = O\left(\frac{\log\log d}{\epsilon^2} + \log d\right)$$

$$= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1$$

$$= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6\ kB!$$

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

$S_1, S_2 \ldots S_k$ $\quad \bar{S}_1, \bar{S}_2, \ldots \bar{S}_k$

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

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

LOGLOG SPACE GUARANTEES

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

$$\begin{aligned}\text{space used} &= O\left(\frac{\log\log d}{\epsilon^2} + \log d\right) \\ &= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1 \\ &= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6\ kB!\end{aligned}$$

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?

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

LOGLOG SPACE GUARANTEES

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

$$\begin{aligned} \text{space used } &= O\left(\frac{\log\log d}{\epsilon^2} + \log d\right) \\ &= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1 \\ &= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6\ kB! \end{aligned}$$

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!

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

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

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.

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).

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.

Estimate number of search 'sessions' that happened in the last month (i.e., a single user making possibly many searches at one time, likely surrounding a specific topic.)

Estimate number of search 'sessions' that happened in the last month (i.e., a single user making possibly many searches at one time, likely surrounding a specific topic.)

IP Address	Time Stamp	Query
127.65.48.31	10:10:26	"bloom filter"
62.54.16.001	10:10:28	"united airlines"
16.578.32.12	10:13:34	"china news"
192.68.001.1	10:14:05	"bmw"
174.15.254.1	10:14:54	"khalid tour"
127.65.48.31	10:15:45	"hashing"
192.68.001.1 ⋮	10:16:18 ⋮	"car loans" ⋮

Estimate number of search 'sessions' that happened in the last month (i.e., a single user making possibly many searches at one time, likely surrounding a specific topic.)

IP Address	Time Stamp	Query
127.65.48.31	10:10:26	"bloom filter"
62.54.16.001	10:10:28	"united airlines"
16.578.32.12	10:13:34	"china news"
192.68.001.1	10:14:05	"bmw"
174.15.254.1	10:14:54	"khalid tour"
127.65.48.31	10:15:45	"hashing"
192.68.001.1	10:16:18	"car loans"
⋮	⋮	⋮

- Count distinct keys where key is $(IP, Hr, Min \mod 10)$.

IN PRACTICE

Estimate number of search 'sessions' that happened in the last month (i.e., a single user making possibly many searches at one time, likely surrounding a specific topic.)

IP Address	Time Stamp	Query
127.65.48.31	10:10:26	"bloom filter"
62.54.16.001	10:10:28	"united airlines"
16.578.32.12	10:13:34	"china news"
192.68.001.1	10:14:05	"bmw"
174.15.254.1	10:14:54	"khalid tour"
127.65.48.31	10:15:45	"hashing"
192.68.001.1	10:16:18	"car loans"
⋮	⋮	⋮

- Count distinct keys where key is $(IP, Hr, Min \mod 10)$.

Estimate number of search ‘sessions’ that happened in the last month (i.e., a single user making possibly many searches at one time, likely surrounding a specific topic.)

IP Address	Time Stamp	Query
127.65.48.31	10:10:26	“bloom filter”
62.54.16.001	10:10:28	“united airlines”
16.578.32.12	10:13:34	“china news”
192.68.001.1	10:14:05	“bmw”
174.15.254.1	10:14:54	“khalid tour”
127.65.48.31	10:15:45	“hashing”
192.68.001.1	10:16:18	“car loans”
⋮	⋮	⋮

- Count distinct keys where key is $(IP, Hr, Min \mod 10)$.
- Using HyperLogLog, cost is roughly that of a (distributed) linear scan (to stream through all items in table).

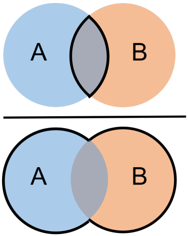
Questions on distinct elements counting?

Jaccard Index: A similarity measure between two sets.

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

Jaccard Index: A similarity measure between two sets.

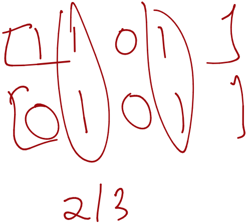
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\#\text{ shared elements}}{\#\text{ total elements}}.$$

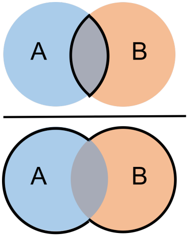


ANOTHER FUNDAMENTAL PROBLEM

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. $J(x,y) = \frac{\#\text{ shared ones}}{\text{total ones}}$.

COMPUTING JACCARD SIMILARITY

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

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

- Computing exactly requires roughly linear time in $|A| + |B|$ (using a hash table or binary search). Not bad.

COMPUTING JACCARD SIMILARITY

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

- Computing exactly requires roughly linear time in $|A| + |B|$ (using a hash table or binary search). Not bad.
- **Near Neighbor Search:** Have a database of n sets/bit strings and given a set A, want to find if it has high similarity to anything in the database. $O(n \cdot \text{ average set size})$ time.

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

- Computing exactly requires roughly linear time in $|A| + |B|$ (using a hash table or binary search). Not bad.
- **Near Neighbor Search:** Have a database of n sets/bit strings and given a set A, want to find if it has high similarity to anything in the database. $O(n \cdot \text{ average set size})$ time.
- **All-pairs Similarity Search:** Have n different sets/bit strings and want to find all pairs with high similarity. $O(n^2 \cdot \text{ average set size})$ time.

COMPUTING JACCARD SIMILARITY

Locality Sensitive Hashing

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

- Computing exactly requires roughly linear time in $|A| + |B|$ (using a hash table or binary search). Not bad.
- **Near Neighbor Search:** Have a database of n sets/bit strings and given a set A, want to find if it has high similarity to anything in the database. $O(n \cdot \text{ average set size})$ time.
- **All-pairs Similarity Search:** Have n different sets/bit strings and want to find all pairs with high similarity. $O(n^2 \cdot \text{ average set size})$ time.

Prohibitively expensive when n is very large. We'll see how to significantly improve on these runtimes with random hashing.

How should you measure similarity between two-documents?

How should you measure similarity between two-documents?

E.g., to detect plagiarism and copyright infringement, to see if an email message is similar to previously seen spam, to detect duplicate webpages in search results, etc.

How should you measure similarity between two-documents?

E.g., to detect plagiarism and copyright infringement, to see if an email message is similar to previously seen spam, to detect duplicate webpages in search results, etc.

- If the documents are not identical, doing a word-by-word comparison typically gives nothing. Can compute edit distance, but this is very expensive if you are comparing many documents.

How should you measure similarity between two-documents?

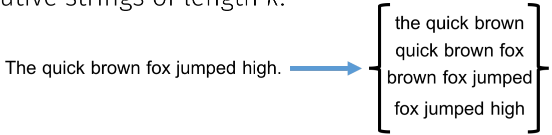
E.g., to detect plagiarism and copyright infringement, to see if an email message is similar to previously seen spam, to detect duplicate webpages in search results, etc.

- If the documents are not identical, doing a word-by-word comparison typically gives nothing. Can compute edit distance, but this is very expensive if you are comparing many documents.
- Shingling + Jaccard Similarity: Represent a document as the set of all consecutive strings of length k.

How should you measure similarity between two-documents?

E.g., to detect plagiarism and copyright infringement, to see if an email message is similar to previously seen spam, to detect duplicate webpages in search results, etc.

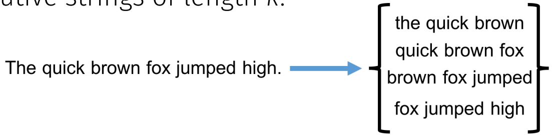
- If the documents are not identical, doing a word-by-word comparison typically gives nothing. Can compute edit distance, but this is very expensive if you are comparing many documents.
- **Shingling + Jaccard Similarity:** Represent a document as the set of all consecutive strings of length k.



How should you measure similarity between two-documents?

E.g., to detect plagiarism and copyright infringement, to see if an email message is similar to previously seen spam, to detect duplicate webpages in search results, etc.

- If the documents are not identical, doing a word-by-word comparison typically gives nothing. Can compute edit distance, but this is very expensive if you are comparing many documents.
- **Shingling + Jaccard Similarity:** Represent a document as the set of all consecutive strings of length k.

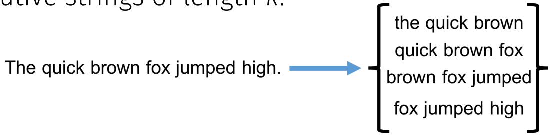


- Measure similarity as Jaccard similarity between shingle sets.

How should you measure similarity between two-documents?

E.g., to detect plagiarism and copyright infringement, to see if an email message is similar to previously seen spam, to detect duplicate webpages in search results, etc.

- If the documents are not identical, doing a word-by-word comparison typically gives nothing. Can compute edit distance, but this is very expensive if you are comparing many documents.
- **Shingling + Jaccard Similarity:** Represent a document as the set of all consecutive strings of length k.



- Measure similarity as Jaccard similarity between shingle sets.
- Also used to measure word similarity. E.g., in spell checkers.

How should you measure similarity between two audio clips?

How should you measure similarity between two audio clips?

E.g. in audio search engines like Shazam, for detecting copyright infringement, for search in sound effect libraries, etc.

How should you measure similarity between two audio clips?

E.g. in audio search engines like Shazam, for detecting copyright infringement, for search in sound effect libraries, etc.

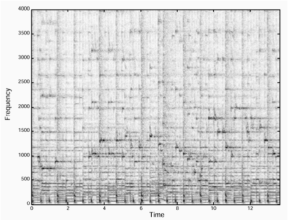
Audio Fingerprinting + Jaccard Similarity:

How should you measure similarity between two audio clips?

E.g. in audio search engines like Shazam, for detecting copyright infringement, for search in sound effect libraries, etc.

Audio Fingerprinting + Jaccard Similarity:

Step 1: Compute the spectrogram: representation of frequency intensity over time.

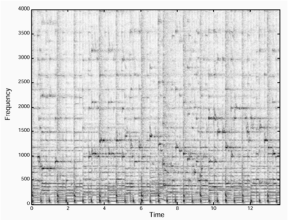


How should you measure similarity between two audio clips?

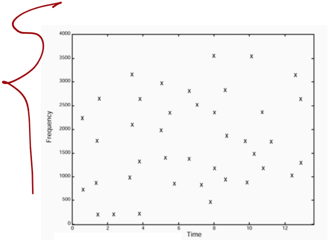
E.g. in audio search engines like Shazam, for detecting copyright infringement, for search in sound effect libraries, etc.

Audio Fingerprinting + Jaccard Similarity:

Step 1: Compute the spectrogram: representation of frequency intensity over time.



Step 2: Threshold the spectrogram to a binary matrix representing the sound clip.

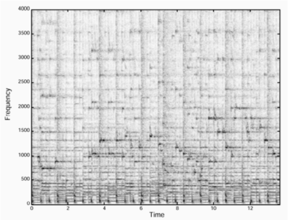


How should you measure similarity between two audio clips?

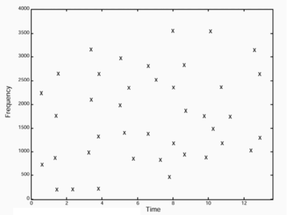
E.g. in audio search engines like Shazam, for detecting copyright infringement, for search in sound effect libraries, etc.

Audio Fingerprinting + Jaccard Similarity:

Step 1: Compute the spectrogram: representation of frequency intensity over time.

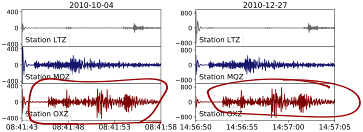


Step 2: Threshold the spectrogram to a binary matrix representing the sound clip.

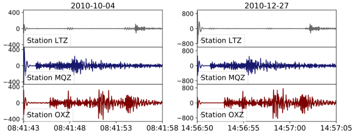


Compare thresholded spectrograms with Jaccard similarity.

Small earthquakes make consistent signatures on seismographs that repeat over time. Detecting repeated signatures lets you detect these otherwise undetectable events.

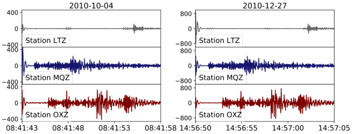


Small earthquakes make consistent signatures on seismographs that repeat over time. Detecting repeated signatures lets you detect these otherwise undetectable events.

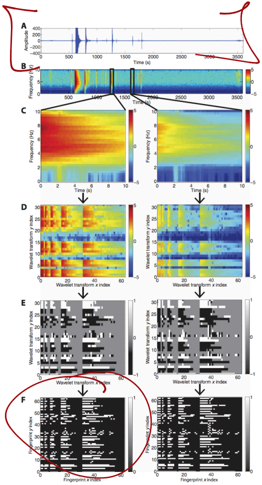


- Split data into overlapping windows of 10 seconds

Small earthquakes make consistent signatures on seismographs that repeat over time. Detecting repeated signatures lets you detect these otherwise undetectable events.

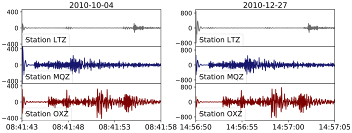


- Split data into overlapping windows of 10 seconds
- Fingerprint each window using the spectrogram (i.e., compute a binary string representing the reading in the window).

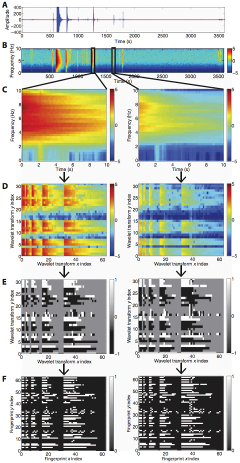


APPLICATION: EARTHQUAKE DETECTION

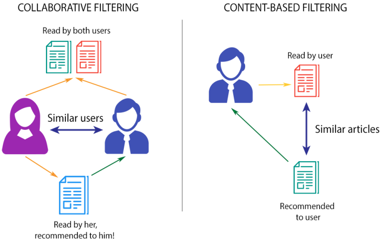
Small earthquakes make consistent signatures on seismographs that repeat over time. Detecting repeated signatures lets you detect these otherwise undetectable events.



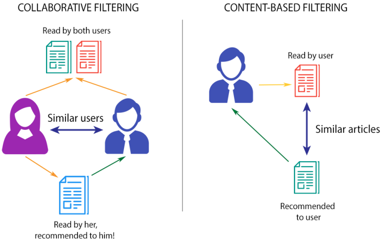
- Split data into overlapping windows of 10 seconds
- Fingerprint each window using the spectrogram (i.e., compute a binary string representing the reading in the window).
- All-pairs search for windows with high Jaccard similarity.



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

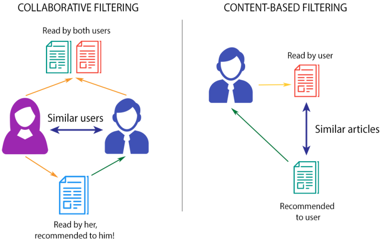


Online recommendation systems are often based on **collaborative filtering**. Simplest approach: find similar users and make recommendations based on those users.



- Twitter: represent a user as the set of accounts they follow. Match similar users based on the Jaccard similarity of these sets. Recommend that you follow accounts followed by similar users.

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 similar 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.

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

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' can be measured with shingles + Jaccard similarity.

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' can be 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.

Why use Jaccard similarity over other metrics like: Hamming distinct (bit strings), correlation (sound waves, seismograms), edit distance (text, genome sequences, etc.)?

Why use Jaccard similarity over other metrics like: Hamming distinct (bit strings), correlation (sound waves, seismograms), edit distance (text, genome sequences, etc.)?

Two Reasons:

- Depending on the application, often is the right measure.
- Even when not ideal, very efficient to compute and implement near neighbor search and all-pairs similarity search with.

Why use Jaccard similarity over other metrics like: Hamming distinct (bit strings), correlation (sound waves, seismograms), edit distance (text, genome sequences, etc.)?

Two Reasons:

- Depending on the application, often is the right measure.
- Even when not ideal, very efficient to compute and implement near neighbor search and all-pairs similarity search with.
- This is what we will cover next time. Using more random hashing!

Questions?