COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco
University of Massachusetts Amherst. Fall 2019.
Lecture 5
• Problem Set 1 was posted on Friday. Due next Thursday 9/26 in Gradescope, before class.
• Don’t leave until the last minute.
Last Class We Covered:

- Bloom Filters:
  - Random hashing to maintain a large sets 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.
Last Class We Covered:

- **Bloom Filters:**
  - Random hashing to maintain a large sets 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 sets 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 analyzing distinct elements algorithm. Learn the ‘median trick’.
- Discuss variants and practical implementations.
Finish Distinct Elements:

- Finish analyzing distinct elements algorithm. Learn the ‘median trick’.
- 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.
First an observation about Bloom filters:

For an \( m \)-bit bloom filter holding \( n \) items, optimal number of hash functions \( k \) is:

\[
    k = \frac{\ln 2}{m/n}
\]

If we want a false positive rate \(< 1/2\) how big does \( m \) need to be in comparison to \( n \)?

\[
    m = O(\log n)
\]

\[
    m = O(p n)
\]

\[
    m = O(n)
\]

\[
    m = O(n^2)
\]

If \( m = n \ln 2 \), optimal \( k = 1 \), and failure rate is:

\[
    \text{Pr}[\text{false positive}] = \left(1 - e^{-n/ln 2} \right)^k = \left(1 - \frac{1}{2} \right)^k = \frac{1}{2^k}
\]

I.e., storing \( n \) items in a bloom filter requires \( O(n) \) space. So what's the point? Truly \( O(n) \) bits, rather than \( O(n \text{ item size}) \).
First an observation about Bloom filters:

$$\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k.$$ 

For an $m$-bit bloom filter holding $n$ items, optimal number of hash functions $k$ is: $k = \ln 2 \cdot \frac{m}{n}$. 

I.e., storing $n$ items in a bloom filter requires $O(n)$ space. So what's the point? Truly $O(n)$ bits, rather than $O(n \cdot \text{item size})$. 

4
First an observation about Bloom filters:

False Positive Rate: \( \delta \approx \left(1 - e^{-\frac{kn}{m}} \right)^k \).

For an \( m \)-bit bloom filter holding \( n \) items, optimal number of hash functions \( k \) is: \( k = \ln 2 \cdot \frac{m}{n} \).

If we want a false positive rate \(< \frac{1}{2}\) how big does \( m \) need to be in comparison to \( n \)?
First an observation about Bloom filters:

False Positive Rate: \( \delta \approx \left( 1 - e^{-\frac{kn}{m}} \right)^k \).

For an \( m \)-bit bloom filter holding \( n \) items, optimal number of hash functions \( k \) is: \( k = \ln 2 \cdot \frac{m}{n} \).

If we want a false positive rate < \( \frac{1}{2} \) how big does \( m \) need to be in comparison to \( n \)?

\[ m = O(\log n), \quad m = O(\sqrt{n}), \quad m = O(n), \quad m = O(n^2)? \]
First an observation about Bloom filters:

False Positive Rate: \( \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \).

For an \( m \)-bit bloom filter holding \( n \) items, optimal number of hash functions \( k \) is: \( k = \ln 2 \cdot \frac{m}{n} \).

If we want a false positive rate \( < \frac{1}{2} \) how big does \( m \) need to be in comparison to \( n \)?

\[ m = O(\log n), \quad m = O(\sqrt{n}), \quad m = O(n), \quad m = O(n^2) ? \]
First an observation about Bloom filters:

False Positive Rate: \( \delta \approx \left( 1 - e^{-\frac{kn}{m}} \right)^k \).

For an \( m \)-bit bloom filter holding \( n \) items, optimal number of hash functions \( k \) is: \( k = \ln 2 \cdot \frac{m}{n} \).

If we want a false positive rate < \( \frac{1}{2} \) how big does \( m \) need to be in comparison to \( n \)?

\( m = O(\log n), \ m = O(\sqrt{n}), \ m = O(n), \ m = O(n^2)? \)

If \( m = \frac{n}{\ln 2} \), optimal \( k = 1 \), and failure rate is:

\[
\delta = \left( 1 - e^{-\frac{n/\ln 2}{n}} \right)^1 = \left( 1 - \frac{1}{2} \right)^1 = \frac{1}{2}.
\]
First an observation about Bloom filters:

False Positive Rate: \( \delta \approx (1 - e^{-\frac{kn}{m}})^k \).

For an \( m \)-bit bloom filter holding \( n \) items, optimal number of hash functions \( k \) is: \( k = \ln 2 \cdot \frac{m}{n} \).

If we want a false positive rate \(< \frac{1}{2} \) how big does \( m \) need to be in comparison to \( n \)?

\[
\begin{align*}
m &= O(\log n), \quad m = O(\sqrt{n}), \quad m = O(n), \quad m = O(n^2) ?
\end{align*}
\]

If \( m = \frac{n}{\ln 2} \), optimal \( k = 1 \), and failure rate is:

\[
\begin{align*}
\delta &= \left(1 - e^{-\frac{n}{m\ln 2}}\right)^1 = \left(1 - \frac{1}{2}\right)^1 = \frac{1}{2}.
\end{align*}
\]

I.e., storing \( n \) items in a bloom filter requires \( O(n) \) space. So what's the point?
First an observation about Bloom filters:

False Positive Rate: \( \delta \approx \left( 1 - e^{-\frac{kn}{m}} \right)^k \).

For an \( m \)-bit bloom filter holding \( n \) items, optimal number of hash functions \( k \) is: \( k = \ln 2 \cdot \frac{m}{n} \).

If we want a false positive rate \(< \frac{1}{2} \) how big does \( m \) need to be in comparison to \( n \)?

\[
m = O(\log n), \ m = O(\sqrt{n}), \ m = O(n), \ m = O(n^2)?
\]

If \( m = \frac{n}{\ln 2} \), optimal \( k = 1 \), and failure rate is:

\[
\delta = \left( 1 - e^{-\frac{n}{\ln 2}} \right)^1 = \left( 1 - \frac{1}{2} \right)^1 = \frac{1}{2}.
\]

I.e., storing \( n \) items in a bloom filter requires \( O(n) \) space. So what’s the point? Truly \( O(n) \) bits, rather than \( O(n \cdot \text{item size}) \).
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Hashing for Distinct Elements:

- Let $h : U \rightarrow [0, 1]$ be a random hash function (continuous output).
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\hat{d} = \frac{1}{s} - 1$
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Hashing for Distinct Elements:

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

- After all items are processed, $s$ is the minimum of $d$ points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Hashing for Distinct Elements:

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

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

Hashing for Distinct Elements:

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

- 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.
- **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.
**PERFORMANCE IN EXPECTATION**

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

\[
\mathbb{E}[s] = \int_0^1 \Pr(s > x) \, dx
\]

So estimate of \( b_d = s \) output by the algorithm is correct if \( s \) exactly equals its expectation.

Does this mean \( \mathbb{E}[b_d] = d \)? No, but:

Approximation is robust: if for any \( \epsilon > 0 \): \( 1 / 4d b_d (1 + 4\epsilon) ).
**Performance in Expectation**

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

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

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

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

- So estimate of \( \hat{d} = \frac{1}{s} - 1 \) output by the algorithm is correct if \( s \) exactly equals its expectation.
**PERFORMANCE IN EXPECTATION**

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

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

- So estimate of $\hat{d} = \frac{1}{s} - 1$ output by the algorithm is correct if $s$ exactly equals its expectation. **Does this mean $\mathbb{E}[\hat{d}] = d$?**
$s$ is the minimum of $d$ points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.

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

• So estimate of $\hat{d} = \frac{1}{s} - 1$ output by the algorithm is correct if $s$ exactly equals its expectation. **Does this mean $E[\hat{d}] = d$?** No, but:
s is the minimum of d points chosen uniformly at random on [0, 1]. Where d = # distinct elements.

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

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

• Approximation is robust: if \(|s - \mathbb{E}[s]| \leq \epsilon \cdot \mathbb{E}[s]\) for any \( \epsilon \in (0, 1/2)\):

\[ (1 - 2\epsilon)d \leq \hat{d} \leq (1 + 4\epsilon)d \]
So question is how well $s$ concentrates around its mean.

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

$s$: minimum of $d$ distinct hashes chosen randomly over $[0,1]$, computed by hashing algorithm. $\hat{d} = \frac{1}{s} - 1$: estimate of # distinct elements $d$. 
So question is how well $s$ concentrates around its mean.

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

**s:** minimum of $d$ distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\hat{d} = \frac{1}{s} - 1$: estimate of # distinct elements $d$. 
So question is how well $s$ concentrates around its mean.

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

$$\Pr[|s - \mathbb{E}[s]| \geq \epsilon \mathbb{E}[s]] \leq .$$

$s$: minimum of $d$ distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\hat{d} = \frac{1}{s} - 1$: estimate of # distinct elements $d$. 
So question is how well $s$ concentrates around its mean.

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

Chebyshev’s Inequality:

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

$s$: minimum of $d$ distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\hat{d} = \frac{1}{s} - 1$: estimate of # distinct elements $d$. 
So question is how well $s$ concentrates around its mean.

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

Chebyshev’s Inequality:

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

$s$: minimum of $d$ distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\hat{d} = \frac{1}{s} - 1$: estimate of $\#$ distinct elements $d$. 
So question is how well $s$ concentrates around its mean. 

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

Chebyshev’s Inequality: 

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

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

$s$: minimum of $d$ distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\hat{d} = \frac{1}{s} - 1$: estimate of # distinct elements $d$. 
So question is how well $s$ concentrates around its mean.

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

Chebyshev’s Inequality:

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

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

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

---

*INITIAL CONCENTRATION BOUND*
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 $h : U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\hat{d} = \frac{1}{s} - 1$
Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $h_1, h_2, \ldots, h_k : U \rightarrow [0, 1]$ be random hash functions
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\hat{d} = \frac{1}{s} - 1$
IMPROVING PERFORMANCE

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

Hashing for Distinct Elements (Improved):

- Let $h_1, h_2, \ldots, h_k : U \rightarrow [0, 1]$ be random hash functions
- $s_1, s_2, \ldots, s_k := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\hat{d} = \frac{1}{s} - 1$
Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $h_1, h_2, \ldots, h_k : U \rightarrow [0, 1]$ be random hash functions
- $s_1, s_2, \ldots, s_k := 1$
- For $i = 1, \ldots, n$
  - For $j=1,\ldots, k$, $s_j := \min(s_j, h_j(x_i))$
- Return $\hat{d} = \frac{1}{s} - 1$
Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $h_1, h_2, \ldots, h_k : U \rightarrow [0, 1]$ be random hash functions
- $s_1, s_2, \ldots, s_k := 1$
- For $i = 1, \ldots, n$
  - For $j=1,\ldots, k$, $s_j := \min(s_j, h_j(x_i))$
- $s := \frac{1}{k} \sum_{j=1}^{k} s_j$
- Return $\hat{d} = \frac{1}{s} - 1$
Leverage the law of large numbers: improve accuracy via repeated independent trials.

**Hashing for Distinct Elements (Improved):**

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

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

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

\( s_j \): minimum of \( d \) distinct hashes chosen randomly over \([0, 1]\). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).

\( \hat{d} = \frac{1}{s} - 1 \): estimate of \# distinct elements \( d \).
s = \frac{1}{k} \sum_{j=1}^{k} s_j. Have already shown that for j = 1, \ldots, k:

\[ E[s_j] = \frac{1}{d+1} \implies E[s] \]

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

\[ s_j: \text{ minimum of } d \text{ distinct hashes chosen randomly over } [0, 1]. \]

\[ \hat{d} = \frac{1}{s} - 1: \text{ estimate of } \# \text{ distinct elements } d. \]
\[ s = \frac{1}{k} \sum_{j=1}^{k} s_j. \] Have already shown that for \( j = 1, \ldots, k: \]

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

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

\( s_j: \) minimum of \( d \) distinct hashes chosen randomly over \([0, 1] \). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j. \)

\( \hat{d} = \frac{1}{s} - 1: \) estimate of \# distinct elements \( d. \)
\( s = \frac{1}{k} \sum_{j=1}^{k} s_j \). Have already shown that for \( j = 1, \ldots, k \):

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

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

- \( s_j \): minimum of \( d \) distinct hashes chosen randomly over \([0, 1]\).
- \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).
- \( \hat{d} = \frac{1}{s} - 1 \): estimate of \# distinct elements \( d \).
\[ s = \frac{1}{k} \sum_{j=1}^{k} s_j. \] Have already shown that for \( j = 1, \ldots, k: \]

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

\[
\text{Var}[s_j] \leq \frac{1}{(d + 1)^2} \implies \text{Var}[s] \leq \frac{1}{k \cdot (d + 1)^2}
\]

\( s_j \): minimum of \( d \) distinct hashes chosen randomly over \([0, 1]\). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).

\( \hat{d} = \frac{1}{s} - 1 \): estimate of \# distinct elements \( d \).
\[ s = \frac{1}{k} \sum_{j=1}^{k} s_j. \] Have already shown that for \( j = 1, \ldots, k \):

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

\[
\text{Var}[s_j] \leq \frac{1}{(d+1)^2} \implies \text{Var}[s] \leq \frac{1}{k \cdot (d+1)^2}
\]

Chebyshev Inequality:

\[
\Pr \left[ |s - \mathbb{E}[s]| \geq \epsilon \mathbb{E}[s] \right] \leq
\]

\( s_j \): minimum of \( d \) distinct hashes chosen randomly over \( [0, 1] \). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).

\( \hat{d} = \frac{1}{s} - 1 \): estimate of \# distinct elements \( d \).
\[ \mathbf{s} = \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j. \] Have already shown that for \( j = 1, \ldots, k \):

\[
\mathbb{E}[\mathbf{s}_j] = \frac{1}{d+1} \quad \implies \quad \mathbb{E}[\mathbf{s}] = \frac{1}{d+1}
\]

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

**Chebyshev Inequality:**

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

\( \mathbf{s}_j \): minimum of \( d \) distinct hashes chosen randomly over \([0,1]\). \( \mathbf{s} = \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j \).

\( \hat{d} = \frac{1}{\mathbf{s}} - 1 \): estimate of \# distinct elements \( d \).
\[ s = \frac{1}{k} \sum_{j=1}^{k} s_j. \] Have already shown that for \( j = 1, \ldots, k: \]

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

\[ \text{Var}[s_j] \leq \frac{1}{(d+1)^2} \implies \text{Var}[s] \leq \frac{1}{k \cdot (d+1)^2} \]

**Chebyshev Inequality:**

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

\( s_j \): minimum of \( d \) distinct hashes chosen randomly over \([0, 1]\). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).

\( \hat{d} = \frac{1}{s} - 1 \): estimate of \( \# \) distinct elements \( d \).
\[ s = \frac{1}{k} \sum_{j=1}^{k} s_j. \] Have already shown that for \( j = 1, \ldots, k: \]

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

\[ \text{Var}[s_j] \leq \frac{1}{(d+1)^2} \implies \text{Var}[s] \leq \frac{1}{k \cdot (d+1)^2} \]

**Chebyshev Inequality:**

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

**s_j:** minimum of \( d \) distinct hashes chosen randomly over \([0, 1]\). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).

\( \hat{d} = \frac{1}{s} - 1 \): estimate of \# distinct elements \( d \).
s = \frac{1}{k} \sum_{j=1}^{k} s_j. Have already shown that for j = 1, \ldots, k:

\begin{align*}
\mathbb{E}[s_j] &= \frac{1}{d+1} \implies \mathbb{E}[s] = \frac{1}{d+1} \\
\text{Var}[s_j] &\leq \frac{1}{(d+1)^2} \implies \text{Var}[s] \leq \frac{1}{k \cdot (d+1)^2}
\end{align*}

Chebyshev Inequality:

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

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

\[ s_j: \text{minimum of } d \text{ distinct hashes chosen randomly over } [0, 1]. \quad s = \frac{1}{k} \sum_{j=1}^{k} s_j. \]

\[ \hat{d} = \frac{1}{s} - 1: \text{estimate of } \# \text{ distinct elements } d. \]
\[ s = \frac{1}{k} \sum_{j=1}^{k} s_j. \] Have already shown that for \( j = 1, \ldots, k \):

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

\[ \text{Var}[s_j] \leq \frac{1}{(d+1)^2} \implies \text{Var}[s] \leq \frac{1}{k \cdot (d+1)^2} \]

**Chebyshev Inequality:**

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

How should we set \( k \) if we want \( \epsilon \) error with probability \( \geq 1 - \delta \)?

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

\( s_j \): minimum of \( d \) distinct hashes chosen randomly over \([0, 1]\). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).

\( \hat{d} = \frac{1}{s} - 1 \): estimate of \# distinct elements \( d \).
\( s = \frac{1}{k} \sum_{j=1}^{k} s_j \). Have already shown that for \( j = 1, \ldots, k \):

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

\[
\text{Var}[s_j] \leq \frac{1}{(d + 1)^2} \implies \text{Var}[s] \leq \frac{1}{k \cdot (d + 1)^2}
\]

Chebyshev Inequality:

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

How should we set \( k \) if we want \( \epsilon \) error with probability \( \geq 1 - \delta \)?

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

\( s_j \): minimum of \( d \) distinct hashes chosen randomly over \([0, 1]\). \( s = \frac{1}{k} \sum_{j=1}^{k} s_j \).

\( \hat{d} = \frac{1}{s} - 1 \): estimate of \# distinct elements \( d \).
Hashing for Distinct Elements:

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

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

- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\hat{d}$ with $|d - \hat{d}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$. 
Hashing for Distinct Elements:

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

• Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\hat{d}$ with $|d - \hat{d}| \leq 4 \epsilon \cdot d$ with probability at least $1 - \delta$.
• Space complexity is $k = \frac{1}{\epsilon^2 \cdot \delta}$ real numbers $s_1, \ldots, s_k$. 
Hashing for Distinct Elements:

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

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

**One Thought:** Apply stronger concentration bounds. E.g., replace Chebyshev with Bernstein.
How can we decrease the cost of a small failure rate $\delta$?

**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 \( \delta \)?

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

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

\[
\Pr (|X - \mu| \geq t) \leq 2 \exp \left( -\frac{t^2}{2\sigma^2 + \frac{4Mt}{3k}} \right).
\]
How can we decrease the cost of a small failure rate $\delta$?

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

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

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

For us, $t^2 = O\left(\frac{\epsilon}{d^2}\right)$ and $\frac{4Mt}{3k} = O\left(\frac{\epsilon}{dk}\right)$ 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.
How can we improve our dependence on the failure rate $\delta$?
How can we improve our dependence on the failure rate $\delta$?

**The median trick:** Run $t = O(\log 1/\delta)$ trials each with failure probability $\delta' = 1/5$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{1}{5\epsilon^2}$ hash functions.
How can we improve our dependence on the failure rate \( \delta \)?

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

- Letting \( \hat{d}_1, \ldots, \hat{d}_t \) be the outcomes of the \( t \) trials, return \( \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t) \).
How can we improve our dependence on the failure rate $\delta$?

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

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

![Diagram](image.png)
How can we improve our dependence on the failure rate $\delta$?

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

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

- If $> 1/2$ of trials fall in $[(1 - \epsilon)d, (1 + \epsilon)d]$, then the median will.
How can we improve our dependence on the failure rate $\delta$?

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

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

- If $\frac{1}{2}$ of trials fall in $[(1 - \epsilon)d, (1 + \epsilon)d]$, then the median will.

- Have $< \frac{1}{2}$ of trials on both the left and right.
How can we improve our dependence on the failure rate $\delta$?

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

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

- If $> 2/3$ of trials fall in $[(1 - \epsilon)d, (1 + \epsilon)d]$, then the median will.
- Have $< 1/3$ of trials on both the left and right.
• $\hat{d}_1, \ldots, \hat{d}_t$ are the outcomes of the $t$ trials, each falling in $[(1 - \epsilon)d, (1 + \epsilon)d]$ with probability at least 4/5.

• $\hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t)$.

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

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

- \( \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t) \).

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

- Let \( X \) be the \# of trials falling in \([(1 - \epsilon)d, (1 + \epsilon)d]\).
• \( \hat{d}_1, \ldots, \hat{d}_t \) are the outcomes of the \( t \) trials, each falling in 
\( [(1 - \epsilon)d, (1 + \epsilon)d] \) with probability at least \( 4/5 \).

• \( \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t) \).

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

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

\[
\Pr \left( \hat{d} \notin [(1 - \epsilon)d, (1 + \epsilon)d] \right) \leq \Pr \left( X < \frac{2}{3} \cdot t \right)
\]
\cdot \hat{d}_1, \ldots, \hat{d}_t \text{ are the outcomes of the } t \text{ trials, each falling in } [(1 - \epsilon)d, (1 + \epsilon)d] \text{ with probability at least } 4/5.
\cdot \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t).

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

\cdot \text{Let } X \text{ be the } \# \text{ of trials falling in } [(1 - \epsilon)d, (1 + \epsilon)d]. \quad \mathbb{E}[X] = \ldots.

\Pr\left(\hat{d} \notin [(1 - \epsilon)d, (1 + \epsilon)d]\right) \leq \Pr\left(X < \frac{2}{3} \cdot t\right)
THE MEDIAN TRICK

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

• \( \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t) \).

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

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

\[
\Pr \left( \hat{d} \notin [ (1 - \epsilon)d, (1 + \epsilon)d ] \right) \leq \Pr \left( X < \frac{2}{3} \cdot t \right)
\]
• $\hat{d}_1, \ldots, \hat{d}_t$ are the outcomes of the $t$ trials, each falling in $[(1 - \epsilon)d, (1 + \epsilon)d]$ with probability at least $4/5$.

• $\hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t)$.

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

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

$$\Pr\left(\hat{d} \notin [(1 - \epsilon)d, (1 + \epsilon)d]\right) \leq \Pr\left(X < \frac{5}{6} \cdot \mathbb{E}[X]\right)$$
• \( \hat{d}_1, \ldots, \hat{d}_t \) are the outcomes of the \( t \) trials, each falling in 
\([(1 - \epsilon)d, (1 + \epsilon)d]\) with probability at least \( 4/5 \).

• \( \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t) \).

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

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

\[
\Pr \left( \hat{d} \notin [(1 - \epsilon)d, (1 + \epsilon)d] \right) \leq \Pr \left( X < \frac{5}{6} \cdot \mathbb{E}[X] \right) \leq \Pr \left( |X - \mathbb{E}[X]| \geq \frac{1}{6} \mathbb{E}[S] \right).
\]
THE MEDIAN TRICK

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

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

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

\[
\begin{align*}
\Pr \left( \hat{d} \notin [(1 - \epsilon)d, (1 + \epsilon)d] \right) & \leq \Pr \left( X < \frac{5}{6} \cdot \mathbb{E}[X] \right) \\
& \leq \Pr \left( \left| X - \mathbb{E}[X] \right| \geq \frac{1}{6} \mathbb{E}[S] \right).
\end{align*}
\]

Apply Chernoff bound:
THE MEDIAN TRICK

• \( \hat{d_1}, \ldots, \hat{d_t} \) are the outcomes of the \( t \) trials, each falling in \([ (1 - \epsilon)d, (1 + \epsilon)d] \) with probability at least \( 4/5 \).

• \( \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t) \).

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

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

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

Apply Chernoff bound:

\[
\Pr \left( |X - \mathbb{E}[X]| \geq \frac{1}{6} \mathbb{E}[X] \right) \leq 2 \exp \left( -\frac{1^2 \cdot 4t}{2 + 1/6} \right) = O \left( e^{-O(t)} \right).
\]
• $\hat{d}_1, \ldots, \hat{d}_t$ are the outcomes of the $t$ trials, each falling in $[(1 - \epsilon)d, (1 + \epsilon)d]$ with probability at least $4/5$.

• $\hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t)$.

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

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

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

Apply Chernoff bound:

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

• 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 $\hat{d} \in [(1 - \epsilon)d, (1 + \epsilon)]$ 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 $\hat{d} \in [(1 - \epsilon)d, (1 + \epsilon)]$ 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 $\frac{t}{5\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ real numbers (the minimum value of each hash function).
Upshot: The median of \( t = O(\log(1/\delta)) \) independent runs of the hashing algorithm for distinct elements returns \( \hat{d} \in [(1 - \epsilon)d, (1 + \epsilon)] \) 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 \( \frac{t}{5\epsilon^2} = O \left( \frac{\log(1/\delta)}{\epsilon^2} \right) \) 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 very large.
Upshot: The median of $t = O(\log(1/\delta))$ independent runs of the hashing algorithm for distinct elements returns $\hat{d} \in [(1 - \epsilon)d, (1 + \epsilon)]$ 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 $\frac{t}{5\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ 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.

<table>
<thead>
<tr>
<th>$h(x_1)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>
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 |
| $\vdots$ | $\vdots$ |
| $h(x_n)$ | 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.

<table>
<thead>
<tr>
<th>$h(x_1)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>$h(x_1)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
</tbody>
</table>

...:

| $h(x_n)$ | 1011000 |

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

Note: Careful averaging of estimates from multiple hash functions.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_1)$</td>
<td>1010010</td>
</tr>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

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

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

Note: Careful averaging of estimates from multiple hash functions.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>$h(x_i)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

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

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

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

Total Space: $O(\log \log d \epsilon^2 + \log d)$ for an $\epsilon$ approximate count.

Note: Careful averaging of estimates from multiple hash functions.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x_i)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x_2)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x_3)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h(x_n)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

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

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

$$\Pr(h(x_i) \text{ has } x \text{ trailing zeros}) = \frac{1}{2^x}$$
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x₁)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x₂)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x₃)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h(xₙ)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

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

With \( d \) distinct elements what do we expect \( m \) to be?

\[
\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^{\log d}}
\]
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

| $h(x_1)$ | 1010010 |
| $h(x_2)$ | 1001100 |
| $h(x_3)$ | 1001110 |
| \vdots   | \vdots   |
| $h(x_n)$ | 1011000 |

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

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

$$\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^\log d} = \frac{1}{d}.$$
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

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

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

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

So with $d$ distinct hashes, expect to see 1 with $\log d$ trailing zeros. Expect $m \approx \log d$. 
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x_i)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x_2)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x_3)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h(x_n)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

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

With \( d \) distinct elements what do we expect \( m \) to be?

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

So with \( d \) distinct hashes, expect to see 1 with \( \log d \) trailing zeros. Expect \( m \approx \log d \). \( m \) takes \( \log \log d \) bits to store.

Total Space:

\[
O\left(\log \log d \epsilon^2 + \log d\right)
\]

for an \( \epsilon \) approximate count.

Note: Careful averaging of estimates from multiple hash functions.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

| \( h(x_i) \) | \( 1010010 \) |
|----------------|
| \( h(x_2) \) | \( 1001100 \) |
| \( h(x_3) \) | \( 1001110 \) |
| \vdots | \vdots |
| \( h(x_n) \) | \( 1011000 \) |

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

With \( d \) distinct elements what do we expect \( m \) to be?

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

So with \( d \) distinct hashes, expect to see 1 with \( \log d \) trailing zeros. Expect \( m \approx \log d \). \( m \) takes \( \log \log d \) bits to store.

**Total Space:** \( O \left( \frac{\log \log d}{\epsilon^2} + \log d \right) \) for an \( \epsilon \) approximate count.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>$h(x_i)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

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

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

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

So with $d$ distinct hashes, expect to see 1 with log $d$ trailing zeros. Expect $m \approx \log d$. $m$ takes log log $d$ bits to store.

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

Note: Careful averaging of estimates from multiple hash functions.
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O \left( \frac{\log \log d}{\epsilon^2} + \log d \right)
\]
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

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

\[
= 1.04 \cdot \lceil \log_2 \log_2 d \rceil + \lceil \log_2 d \rceil \text{ bits}
\]
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O \left( \frac{\log \log d}{\epsilon^2} + \log d \right) = \frac{1.04 \cdot [\log_2 \log_2 d]}{\epsilon^2} + [\log_2 d] \text{ bits} = \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6 \text{ kB}!
\]
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

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

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

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

**Mergeable Sketch:** Consider the case (essentially always in practice) that the items are processed on different machines.
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

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

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

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

**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 easy to merge them to give \(HLL(x_1, \ldots, x_n, y_1, \ldots, y_n)\).
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

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

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

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

**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 easy to merge them to give \( HLL(x_1, \ldots, x_n, y_1, \ldots, y_n) \). How?
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

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

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

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

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

- Given data structures (sketches) \(HLL(x_1, \ldots, x_n), HLL(y_1, \ldots, y_n)\) it 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.
**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. ~ 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. ~ 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).
Hyperloglog in Practice

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

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

- Count number 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.)

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Time Stamp</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.65.48.31</td>
<td>10:10:26</td>
<td>“bloom filter”</td>
</tr>
<tr>
<td>62.54.16.001</td>
<td>10:10:28</td>
<td>“united airlines”</td>
</tr>
<tr>
<td>16.578.32.12</td>
<td>10:13:34</td>
<td>“china news”</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:14:05</td>
<td>“bmw”</td>
</tr>
<tr>
<td>174.15.254.1</td>
<td>10:14:54</td>
<td>“khalid tour”</td>
</tr>
<tr>
<td>127.65.48.31</td>
<td>10:15:45</td>
<td>“hashing”</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:16:18</td>
<td>“car loans”</td>
</tr>
</tbody>
</table>
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.)

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Time Stamp</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.65.48.31</td>
<td>10:10:26</td>
<td>“bloom filter”</td>
</tr>
<tr>
<td>62.54.16.001</td>
<td>10:10:28</td>
<td>“united airlines”</td>
</tr>
<tr>
<td>16.578.32.12</td>
<td>10:13:34</td>
<td>“china news”</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:14:05</td>
<td>“bmw”</td>
</tr>
<tr>
<td>174.15.254.1</td>
<td>10:14:54</td>
<td>“khalid tour”</td>
</tr>
<tr>
<td>127.65.48.31</td>
<td>10:15:45</td>
<td>“hashing”</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:16:18</td>
<td>“car loans”</td>
</tr>
</tbody>
</table>

- 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.)

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Time Stamp</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.65.48.31</td>
<td>10:10:26</td>
<td>“bloom filter”</td>
</tr>
<tr>
<td>62.54.16.001</td>
<td>10:10:28</td>
<td>“united airlines”</td>
</tr>
<tr>
<td>16.578.32.12</td>
<td>10:13:34</td>
<td>“china news”</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:14:05</td>
<td>“bmw”</td>
</tr>
<tr>
<td>174.15.254.1</td>
<td>10:14:54</td>
<td>“khalid tour”</td>
</tr>
<tr>
<td>127.65.48.31</td>
<td>10:15:45</td>
<td>“hashing”</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:16:18</td>
<td>“car loans”</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

- 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.)

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Time Stamp</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.65.48.31</td>
<td>10:10:26</td>
<td>&quot;bloom filter&quot;</td>
</tr>
<tr>
<td>62.54.16.001</td>
<td>10:10:28</td>
<td>&quot;united airlines&quot;</td>
</tr>
<tr>
<td>16.578.32.12</td>
<td>10:13:34</td>
<td>&quot;china news&quot;</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:14:05</td>
<td>&quot;bmw&quot;</td>
</tr>
<tr>
<td>174.15.254.1</td>
<td>10:14:54</td>
<td>&quot;khalid tour&quot;</td>
</tr>
<tr>
<td>127.65.48.31</td>
<td>10:15:45</td>
<td>&quot;hashing&quot;</td>
</tr>
<tr>
<td>192.68.001.1</td>
<td>10:16:18</td>
<td>&quot;car loans&quot;</td>
</tr>
</tbody>
</table>

- 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.

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

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

- 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?

- 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-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.

- **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 $k$.

  The quick brown fox jumped high.

  \[
  \text{the quick brown fox jumped high, quick brown fox jumped,}
  \]

  brown fox jumped
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 $k$.

  The quick brown fox jumped high.

  \[
  \text{the quick brown fox jumped high}
  \]

  \[
  \text{quick brown fox}
  \]

• 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 $k$.

  The quick brown fox jumped high.

  \[
  \text{the quick brown fox jumped high}
  \]

- 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?

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.
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:**

**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.
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.
- 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 represents a user as the set of accounts that they follow. They match similar uses 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 represents a user as the set of accounts that they follow. They match similar uses based on the Jaccard similarity of these sets. Recommend that you follow accounts followed by similar users.
- Netflix looks at sets of movies watched, Amazon at products purchased, etc.
Goal: Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).
Goal: Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

Strategy: Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.
Goal: Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

Strategy: Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

MinHash(A):

- Let \( h : U \rightarrow [0, 1] \) be a random hash function
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):**

- Let $h : U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):**

- Let \( h : U \rightarrow [0, 1] \) be a random hash function
- \( s := 1 \)
- For \( x_1, \ldots, x_{|A|} \in A \)
  - \( s := \min(s, h(x_R)) \)
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):**

- Let $h : U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
- For $x_1, \ldots, x_{|A|} \in A$
  - $s := \min(s, h(x_R))$
- Return $s$
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):**

- Let $h : U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
- For $x_1, \ldots, x_{|A|} \in A$
  - $s := \min(s, h(x_k))$
- Return $s$
For two sets \( A \) and \( B \), what is \( \Pr(\text{MinHash}(A) = \text{MinHash}(B)) \)?
For two sets \( A \) and \( B \), what is \( \Pr(\text{MinHash}(A) = \text{MinHash}(B)) \)?

- Since we are hashing into the continuous range \([0, 1]\), we will never have \( \Pr(h(x) = h(y)) \) for \( x \neq y \).
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $\Pr(h(x) = h(y))$ for $x \neq y$.
- So, we will only have $\text{MinHash}(A) = \text{MinHash}(B)$ if the same item $x$ has the minimum hash value in both sets. I.e., if $x$ has the lowest hash value of all item in both sets (in $A \cup B$).
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $\Pr(h(x) = h(y))$ for $x \neq y$.
- So, we will only have $\text{MinHash}(A) = \text{MinHash}(B)$ if the same item $x$ has the minimum hash value in both sets. I.e., if $x$ has the lowest hash value of all item in both sets (in $A \cup B$).

$$\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = \sum_{x \in A \cap B} \Pr(\text{MinHash}(A) = h(x) \cap \text{MinHash}(B) = h(x))$$
For two sets $A$ and $B$, what is $Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $Pr(h(x) = h(y))$ for $x \neq y$.
- So, we will only have $\text{MinHash}(A) = \text{MinHash}(B)$ if the same item $x$ has the minimum hash value in both sets. I.e., if $x$ has the lowest hash value of all item in both sets (in $A \cup B$).

$$Pr(\text{MinHash}(A) = \text{MinHash}(B)) =$$

$$\sum_{x \in A \cap B} Pr(\text{MinHash}(A) = h(x) \cap \text{MinHash}(B) = h(x))$$

$$= \sum_{x \in A \cap B} Pr(x = \arg \min_{y \in A \cup B} h(y))$$
For two sets $A$ and $B$, what is $\text{Pr}(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $\text{Pr}(h(x) = h(y))$ for $x \neq y$.
- So, we will only have $\text{MinHash}(A) = \text{MinHash}(B)$ if the same item $x$ has the minimum hash value in both sets. I.e., if $x$ has the lowest hash value of all item in both sets (in $A \cup B$).

$$\text{Pr}(\text{MinHash}(A) = \text{MinHash}(B)) =$$

$$\sum_{x \in A \cap B} \text{Pr}(\text{MinHash}(A) = h(x) \cap \text{MinHash}(B) = h(x))$$

$$= \sum_{x \in A \cap B} \text{Pr}(x = \arg \min_{y \in A \cup B} h(y))$$

$$= \frac{|A \cap B|}{|A \cup B|}$$
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $\Pr(h(x) = h(y))$ for $x \neq y$.
- So, we will only have $\text{MinHash}(A) = \text{MinHash}(B)$ if the same item $x$ has the minimum hash value in both sets. I.e., if $x$ has the lowest hash value of all item in both sets (in $A \cup B$).

\[
\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = \\
\sum_{x \in A \cap B} \Pr(\text{MinHash}(A) = h(x) \cap \text{MinHash}(B) = h(x))
\]

\[
= \sum_{x \in A \cap B} \Pr(x = \arg \min_{y \in A \cup B} h(y))
\]

\[
= \frac{|A \cap B|}{|A \cup B|} = J(A, B)
\]
**Upshot**: MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*. 
**Upshot:** MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

- An instance of *locality sensitive hashing*. Viewing MinHash as a hash function on sets, the collision probability between \textit{MinHash}(A) and \textit{MinHash}(B) is higher when A and B are more similar.
**Upshot:** MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

- An instance of **locality sensitive hashing**. Viewing MinHash as a hash function on sets, the collision probability between $\text{MinHash}(A)$ and $\text{MinHash}(B)$ is higher when $A$ and $B$ are more similar.

**Next Time:**

- Further analysis of MinHash and how it used to speed up similarity search. Locality sensitive hashing more generally.
**Upshot:** MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

- An instance of *locality sensitive hashing*. Viewing MinHash as a hash function on sets, the collision probability between \( \text{MinHash}(A) \) and \( \text{MinHash}(B) \) is higher when \( A \) and \( B \) are more similar.

**Next Time:**

- Further analysis of MinHash and how it used to speed up similarity search. Locality sensitive hashing more generally.
- Start on the Frequent Items problem in stream processing.
Questions?