

COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2026.

Lecture 7

- Problem Set 1 is due this Friday. Gradescope assignment should be open for submission.
- Online quiz due Monday 9am, and in class quiz Monday.

Summary

Last Class:

- Finish exponential concentration bounds
- Bloom filters for storing a set with a small false positive rate.

Summary

Last Class:

- Bloom filters for storing a set with a small false positive rate.

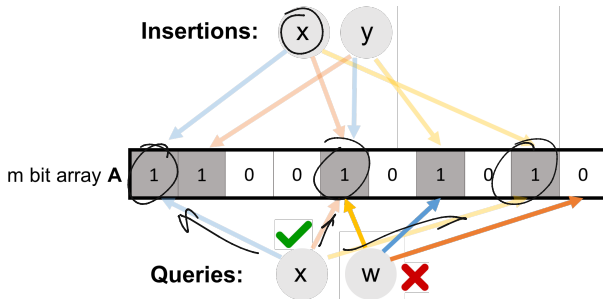
This Class:

- Finish up Bloom filters
- Intro to streaming algorithms – computation when space is limited.
- Frequent items estimation via Count-Min sketch

Bloom Filters

Chose k independent random hash functions $h_1, \dots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array A containing m bits, all initially 0.
- $insert(x)$: set all bits $A[h_1(x)] = \dots = A[h_k(x)] := 1$.
- $query(x)$: return 1 only if $A[h_1(x)] = \dots = A[h_k(x)] = 1$.



No false negatives. False positives more likely with more insertions.

Analysis

How does the false positive rate δ depend on m , k , and # of items?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}\Pr(A[h_1(w)] = \dots = A[h_k(w)] = 1) \\&= \Pr(A[h_1(w)] = 1) \times \dots \times \Pr(A[h_k(w)] = 1) \\&= \left(1 - e^{-\frac{kn}{m}}\right)^k\end{aligned}$$

n : total number items in filter, m : number of bits in filter, k : number of random hash functions, $h_1, \dots, h_k$: hash functions, A : bit array, δ : false positive rate.

Analysis

How does the false positive rate δ depend on m , k , and # of items?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\Pr(A[h_1(w)] = \dots = A[h_k(w)] = 1)$$

$$= \Pr(A[\underline{h_1(w)}] = 1) \times \dots \times \Pr(A[\underline{h_k(w)}] = 1)$$

Handwritten notes: $h_i(w)$ and $h_j(w)$ are independent. $\dots$ will depend on state A .

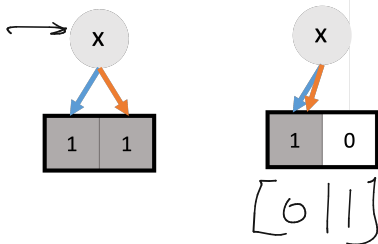
$$= \left(1 - e^{-\frac{kn}{m}}\right)^k$$

Actually Incorrect! Dependent events.

n : total number items in filter, m : number of bits in filter, k : number of random hash functions, $h_1, \dots, h_k$: hash functions, A : bit array, δ : false positive rate.

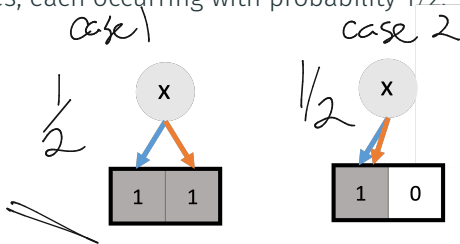
Dependencies

Consider a bloom filter with $k = 2$ and $m = 2$ and $n = 1$. Two possible states, each occurring with probability $1/2$.



Dependencies

Consider a bloom filter with $k = 2$ and $m = 2$ and $n = 1$. Two possible states, each occurring with probability $1/2$.

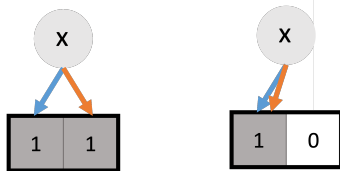


For a non-inserted item w , $\Pr(A[h_1(w)] = 1) = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$.

case 1 case 2

Dependencies

Consider a bloom filter with $k = 2$ and $m = 2$ and $n = 1$. Two possible states, each occurring with probability $1/2$.



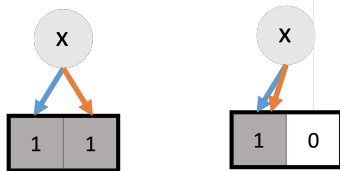
For a non-inserted item w , $\Pr(A[h_1(w)] = 1) = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$. But...

$$\Pr(A[h_1(w)] = 1 | A[h_2(w)] = 1) = \frac{\Pr(A[h_1(w)] = 1 \cap A[h_2(w)] = 1)}{\Pr(A[h_2(w)] = 1)}$$

3/4
dependent

Dependencies

Consider a bloom filter with $k = 2$ and $m = 2$ and $n = 1$. Two possible states, each occurring with probability $1/2$.



For a non-inserted item w , $\Pr(A[h_1(w)] = 1) = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$. But...

$$\begin{aligned}\Pr(A[h_1(w)] = 1 | A[h_2(w)] = 1) &= \frac{\Pr(A[h_1(w)] = 1 \cap A[h_2(w)] = 1)}{\Pr(A[h_2(w)] = 1)} \\ &= \frac{\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{4}}{\frac{3}{4}} = \frac{5}{6} \neq \frac{3}{4} \quad \text{dependent events}\end{aligned}$$

Correct Analysis Sketch

Need to show that the number of zeros t in A after n insertions is bounded by $O\left(e^{-\frac{kn}{m}}\right)$ with high probability.

Can apply Theorem 2 of:

<http://cglab.ca/~morin/publications/ds/bloom-submitted.pdf>

I.e., knowing that $A[h_i(w)] = 1$ for some i doesn't tell you much about the number of zeros in the array, and thus doesn't tell you much about $\Pr(A[h_j(w)] = 1)$ for $j \neq i$.

Optimizing Parameters

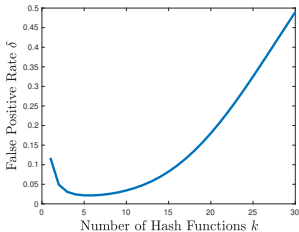
False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$.

Optimizing Parameters

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$. How should we set k to minimize the FPR given a fixed amount of space m ?

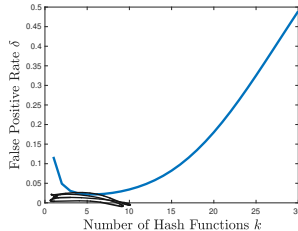
Optimizing Parameters

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$. How should we set k to minimize the FPR given a fixed amount of space m ?



Optimizing Parameters

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$. How should we set k to minimize the FPR given a fixed amount of space m ?



- Can differentiate to show optimal number of hashes is

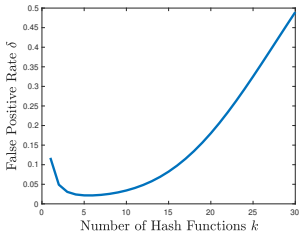
$$k = \ln 2 \cdot \frac{m}{n}$$

$$\left(1 - e^{-\frac{\ln 2 \cdot m}{n} \cdot \frac{n}{m}}\right)^k = (1 - e^{-\ln 2})^k = \left(\frac{1}{2}\right)^k$$

aring has $\approx 1/2$
entries equal to
or 0

Optimizing Parameters

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$. How should we set k to minimize the FPR given a fixed amount of space m ?



- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$.
- Balances filling up the array vs. having enough hashes so that even when the array is pretty full, a new item is unlikely to yield a false positive.

False Positive Rate

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$.

Users

Movies

5			1	4			
	3					5	
				4			
	5						5
1			2				

- Say we have 100 million users, each who have rated 10 movies.
- $n = 10^9 = n$ (user,movie) pairs with non-empty ratings.



False Positive Rate

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$.

Users

Movies

5			1	4				
	3					5		
				4				
	5							5
1			2					

- Say we have 100 million users, each who have rated 10 movies.
- $n = 10^9 = n$ (user,movie) pairs with non-empty ratings.
- Allocate $m = 8n = \underline{8 \times 10^9}$ bits for a Bloom filter (1 GB).

False Positive Rate

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$.

Movies

	1	2	3	4	5	6	7	8
Users	5			1	4			
		3					5	
				4				
		5						5
	1			2				

- Say we have 100 million users, each who have rated 10 movies.
- $n = 10^9 = n$ (user,movie) pairs with non-empty ratings.
- Allocate $m = 8n = 8 \times 10^9$ bits for a Bloom filter (1 GB).
- Set $k = \ln 2 \cdot \frac{m}{n} = 5.54 \approx 6$.

$$\ln 2 \cdot 8$$

False Positive Rate

False Positive Rate: with m bits of storage, k hash functions, and n

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

$k \uparrow$
fraction of 1s $\uparrow$

Users

Movies

5			1	4				
	3					5		
				4				
		5						5
1			2					

- Say we have 100 million users, each who have rated 10 movies.
- $n = 10^9 = n$ (user,movie) pairs with non-empty ratings.
- Allocate $m = 8n = 8 \times 10^9$ bits for a Bloom filter (1 GB).
- Set $k = \ln 2 \cdot \frac{m}{n} = 5.54 \approx 6$.
- False positive rate is $\approx \left(1 - e^{-k \cdot \frac{n}{m}}\right)^k \approx \frac{1}{2^k} \approx \frac{1}{2^{5.54}} = .021$.

8-bits per item

$$\left(\frac{1}{2}\right)^k$$

Bloom Filter Note

General formula for Bloom filter space complexity:

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

For an m -bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$. So to achieve false positive rate δ we need:

Bloom Filter Note

General formula for Bloom filter space complexity:

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

For an m -bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$. So to achieve false positive rate δ we need:

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

Bloom Filter Note

General formula for Bloom filter space complexity:

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

For an m -bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$. So to achieve false positive rate δ we need:

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

$$\left(1 - e^{-\ln 2}\right)^k \leq \delta$$

Bloom Filter Note

General formula for Bloom filter space complexity:

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

For an m -bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$. So to achieve false positive rate δ we need:

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

$$\left(1 - e^{-\ln 2}\right)^k \leq \delta$$

$$\left(\frac{1}{2}\right)^k \leq \delta.$$

Bloom Filter Note

General formula for Bloom filter space complexity:

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

For an m -bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$. So to achieve false positive rate δ we need:

revised formula for k : $\frac{kn}{\ln 2} = m$

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

$$\left(1 - e^{-\ln 2}\right)^k \leq \delta$$

$$\left(\frac{1}{2}\right)^k \leq \delta.$$

same

$$k \log(1/2) \leq \log(\delta)$$
$$k \geq \frac{\log(\delta)}{\log(1/2)} = \log_2(\delta)$$

i.e., we need $k = \log(1/\delta)$ and so $m = \frac{kn}{\ln 2} = O(n \log(1/\delta))$.

$$\frac{\log_2(1/\delta) \cdot n}{\ln 2}$$

Bloom Filter Note

General formula for Bloom filter space complexity:

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

For an m -bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$. So to achieve false positive rate δ we need:

$$\begin{aligned}\left(1 - e^{-\frac{kn}{m}}\right)^k &\leq \delta \\ \left(1 - e^{-\ln 2}\right)^k &\leq \delta \\ \left(\frac{1}{2}\right)^k &\leq \delta.\end{aligned}$$

I.e., we need $k = \log(1/\delta)$ and so $m = \frac{kn}{\ln 2} = O(n \log(1/\delta))$.

So storing n items in a bloom filter requires $O(n)$ space. So what's the point?

Bloom Filter Note

General formula for Bloom filter space complexity:

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

For an m -bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$. So to achieve false positive rate δ we need:

$$\begin{aligned}\left(1 - e^{-\frac{kn}{m}}\right)^k &\leq \delta \\ \left(1 - e^{-\ln 2}\right)^k &\leq \delta \\ \left(\frac{1}{2}\right)^k &\leq \delta.\end{aligned}$$

I.e., we need $k = \log(1/\delta)$ and so $m = \frac{kn}{\ln 2} = O(n \log(1/\delta))$.

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

Questions on Bloom Filters?

Streaming Algorithms

Streaming Algorithms

Stream Processing: Have a massive dataset X with n items $\underline{x_1}, \underline{x_2}, \dots, \underline{x_n}$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
- Typically must compress the data on the fly, storing a data structure from which you can still learn useful information.
- Often the compression is randomized. E.g., bloom filters.
- Compared to traditional algorithm design, which focuses on minimizing runtime, the big question here is how much space is needed to answer queries of interest.



Some Examples

- **Sensor data:** images from telescopes (30 terabytes per night from the Vera C. Rubin Observatory), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors, electrical grid monitoring.



Some Examples

- **Sensor data:** images from telescopes (30 terabytes per night from the Vera C. Rubin Observatory), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors, electrical grid monitoring.
- **Internet Traffic:** 8.5 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...

Some Examples

- **Sensor data:** images from telescopes (30 terabytes per night from the Vera C. Rubin Observatory), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors, electrical grid monitoring.
- **Internet Traffic:** 8.5 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...
- **Datasets in Machine Learning:** When training e.g. a neural network on a large dataset (ImageNet with 14 million images or LLaMA-2 on trillions of tokens of text), the data is typically processed in a stream due to storage limitations.

Some Examples

- **Sensor data:** images from telescopes (30 terabytes per night from the Vera C. Rubin Observatory), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors, electrical grid monitoring.
- **Internet Traffic:** 8.5 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...
- **Datasets in Machine Learning:** When training e.g. a neural network on a large dataset (ImageNet with 14 million images or LLaMA-2 on trillions of tokens of text), the data is typically processed in a stream due to storage limitations.



The Frequent Items Problems

k -Frequent Items (Heavy-Hitters) Problem: Consider a stream of n items $x_1, \dots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.

The Frequent Items Problems

k -Frequent Items (Heavy-Hitters) Problem: Consider a stream of n items $x_1, \dots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
5	12	3	3	4	5	5	10	3

—

—

—

The Frequent Items Problems

k -Frequent Items (Heavy-Hitters) Problem: Consider a stream of n items $x_1, \dots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

$$k=3$$

Return items that appear $\geq 9/3 = 3$

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
5	12	3	3	4	5	5	10	3

The Frequent Items Problems

k -Frequent Items (Heavy-Hitters) Problem: Consider a stream of n items $x_1, \dots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
5	12	3	3	4	5	5	10	3

- What is the maximum number of items that can be returned?

a) n b) k c) n/k d) $\log n$

The Frequent Items Problems

k -Frequent Items (Heavy-Hitters) Problem: Consider a stream of n items $x_1, \dots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.

Buyer-moore

Misra-Gries counter

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
5	12	3	3	4	5	5	10	3

k

- What is the maximum number of items that can be returned?
a) n b) k c) n/k d) $\log n$
- Trivial with $O(n)$ space – store the count for each item and return the one that appears $\geq n/k$ times.

- Can we do it with less space? I.e., without storing all n items?

- store ^{running} list of k most frequent items

$k=2$ 5, 1, 2, 1, 2, 5, 3, 4, 3, 4, 5, 6, 7, 6, 7, 5 ¹⁴

The Frequent Items Problem

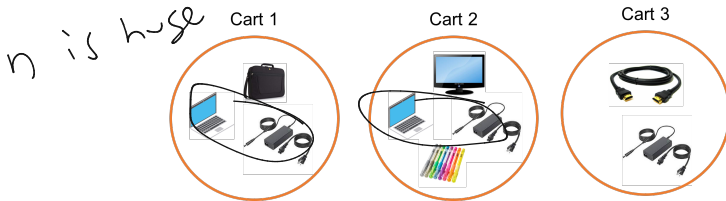
Applications of Frequent Items:

- Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
- Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
- 'Iceberg queries' for all items in a database with frequency above some threshold.

Generally want very fast detection, without having to scan through database/logs. I.e., want to maintain a running list of frequent items that appear in a stream.

Frequent Itemset Mining

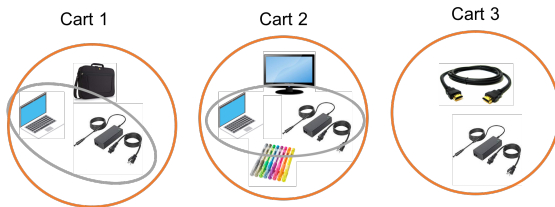
Association rule learning: A very common task in data mining is to identify common associations between different events.



- Identified via **frequent itemset** counting. Find all sets of t items that appear many times in the same basket.
- Frequency of an itemset is known as its support.
- A single basket includes many different itemsets, and with many different baskets an efficient approach is critical. E.g., baskets are Twitter users and itemsets are subsets of who they follow.

Frequent Itemset Mining

Association rule learning: A very common task in data mining is to identify common associations between different events.



- Identified via **frequent itemset** counting. Find all sets of t items that appear many times in the same basket.
- Frequency of an itemset is known as its support.
- A single basket includes many different itemsets, and with many different baskets an efficient approach is critical. E.g., baskets are Twitter users and itemsets are subsets of who they follow.

Approximate Frequent Elements

Issue: No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency n/k (should be output) and $n/k - 1$ (should not be output).

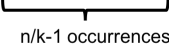
x_1	x_2	x_3	x_4	x_5	x_6	...	$x_{n-n/k+1}$	...	x_n
3	12	9	27	4	101	...	3	...	3

n/k-1 occurrences

Approximate Frequent Elements

Issue: No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency n/k (should be output) and $n/k - 1$ (should not be output).

x_1	x_2	x_3	x_4	x_5	x_6	...	$x_{n-n/k+1}$	...	x_n
3	12	9	27	4	101	...	3	...	3


n/k-1 occurrences

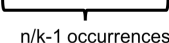
(ϵ, k) -Frequent Items Problem: Consider a stream of n items $x_1, \dots, x_n$. Return a set F of items, including all items that appear at least $\frac{n}{k}$ times and only items that appear at least $(1 - \epsilon) \cdot \frac{n}{k}$ times.

$$\left[(1 - \epsilon) \frac{n}{k}, \frac{n}{k} \right]$$

Approximate Frequent Elements

Issue: No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency n/k (should be output) and $n/k - 1$ (should not be output).

x_1	x_2	x_3	x_4	x_5	x_6	...	$x_{n-n/k+1}$	...	x_n
3	12	9	27	4	101	...	3	...	3


n/k-1 occurrences

(ϵ, k) -Frequent Items Problem: Consider a stream of n items $x_1, \dots, x_n$. Return a set F of items, including all items that appear at least $\frac{n}{k}$ times and only items that appear at least $(1 - \epsilon) \cdot \frac{n}{k}$ times.

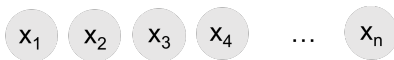
- An example of relaxing to a ‘promise problem’: for items with frequencies in $[(1 - \epsilon) \cdot \frac{n}{k}, \frac{n}{k})$ no output guarantee.

Frequent Elements with Count-Min Sketch

Today: Count-min sketch – a random hashing based method closely related to bloom filters.

Frequent Elements with Count-Min Sketch

Today: Count-min sketch – a random hashing based method closely related to bloom filters.



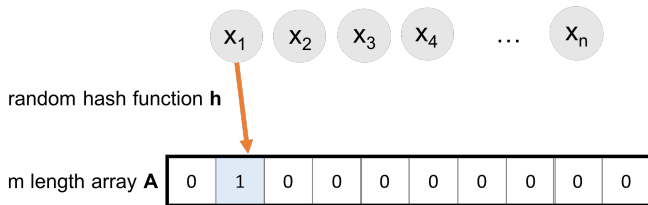
random hash function h

m length array **A**

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

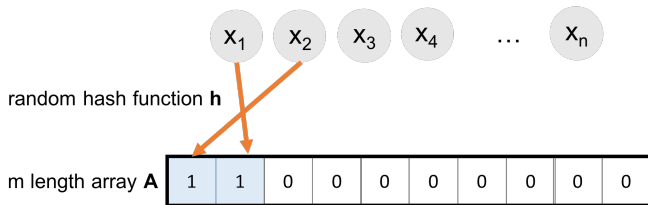
Frequent Elements with Count-Min Sketch

Today: Count-min sketch – a random hashing based method closely related to bloom filters.



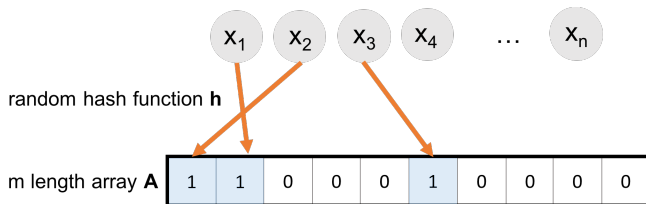
Frequent Elements with Count-Min Sketch

Today: Count-min sketch – a random hashing based method closely related to bloom filters.



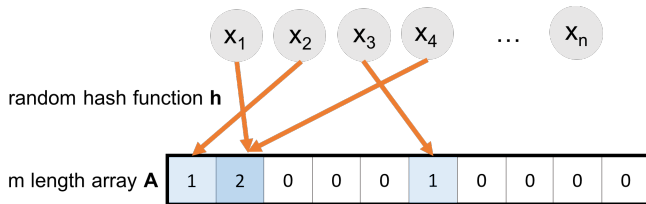
Frequent Elements with Count-Min Sketch

Today: Count-min sketch – a random hashing based method closely related to bloom filters.



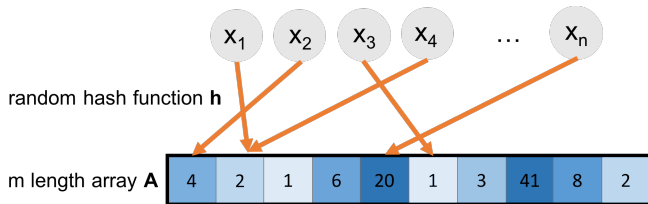
Frequent Elements with Count-Min Sketch

Today: Count-min sketch – a random hashing based method closely related to bloom filters. $X_1 = X_4$ or $X_1 \neq X_4$



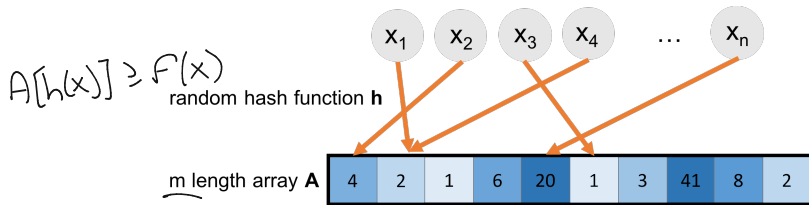
Frequent Elements with Count-Min Sketch

Today: Count-min sketch – a random hashing based method closely related to bloom filters.



Frequent Elements with Count-Min Sketch

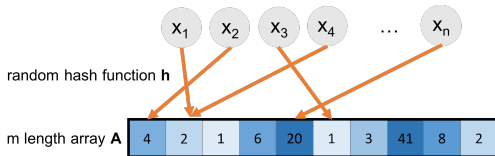
Today: Count-min sketch – a random hashing based method closely related to bloom filters.



Will use $A[h(x)]$ to estimate $f(x)$, the frequency of x in the stream. I.e., $|\{x_i : x_i = x\}|$.

For now: focus on storing approximate counts for any item.
- will store heavy items separately

Count-Min Sketch Accuracy



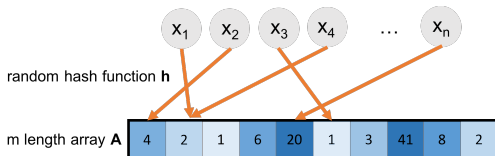
Use $A[h(x)]$ to estimate $f(x)$.

Claim 1: We always have $A[h(x)] \geq f(x)$.

- $A[h(x)]$ counts the number of occurrences of any y with $\underline{h(y) = h(x)}$, including x itself.

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy



Use $A[h(x)]$ to estimate $f(x)$.

Claim 1: We always have $A[h(x)] \geq f(x)$.

- $A[h(x)]$ counts the number of occurrences of any y with $h(y) = h(x)$, including x itself. */ error*
- $A[h(x)] = f(x) + \sum_{y \neq x: h(y)=h(x)} f(y)$.

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}} .$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}} .$$

Expected Error:

$$\mathbb{E} \left[\sum_{y \neq x: h(y)=h(x)} f(y) \right] =$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}}.$$

Expected Error:

$$\mathbb{E} \left[\sum_{y \neq x: h(y)=h(x)} f(y) \right] = \sum_{y \neq x} \underbrace{\Pr(h(y) = h(x))}_{\frac{1}{m}} \cdot \underbrace{f(y)}$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}} .$$

Expected Error:

$$\begin{aligned} \mathbb{E} \left[\sum_{y \neq x: h(y)=h(x)} f(y) \right] &= \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\ &= \sum_{y \neq x} \frac{1}{m} \cdot f(y) \end{aligned}$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}}.$$

Expected Error:

$$\begin{aligned} \mathbb{E} \left[\sum_{y \neq x: h(y)=h(x)} f(y) \right] &= \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\ &= \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m} \end{aligned}$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}}.$$

Expected Error:

$$\begin{aligned}\mathbb{E} \left[\sum_{y \neq x: h(y)=h(x)} f(y) \right] &= \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\ &= \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}\end{aligned}$$

What is a bound on probability that the error is $\geq \frac{2n}{m}$?

$$\Pr\left(\text{error} \geq \frac{2n}{m}\right) \leq \frac{\mathbb{E}[\text{error}]}{\frac{2n}{m}} = \frac{1}{2}$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}}.$$

Expected Error:

$$\begin{aligned}\mathbb{E} \left[\sum_{y \neq x: h(y)=h(x)} f(y) \right] &= \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\ &= \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}\end{aligned}$$

What is a bound on probability that the error is $\geq \frac{2n}{m}$?

Markov's inequality: $\Pr \left[\sum_{y \neq x: h(y)=h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2}.$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

$$A[h(x)] = f(x) + \underbrace{\sum_{y \neq x: h(y)=h(x)} f(y)}_{\text{error in frequency estimate}}.$$

Expected Error:

$$\begin{aligned} \mathbb{E} \left[\sum_{y \neq x: h(y)=h(x)} f(y) \right] &= \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\ &= \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m} \end{aligned}$$

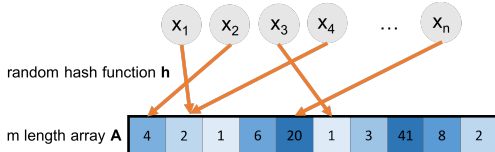
What is a bound on probability that the error is $\geq \frac{2n}{m}$?

Markov's inequality: $\Pr \left[\sum_{y \neq x: h(y)=h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2} \cdot \Pr(h(x) = h(y)) \leq \frac{1}{m}$

What property of h is required to show this bound? a) fully random
b) pairwise independent c) 2-universal d) ~~locality sensitive~~

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy

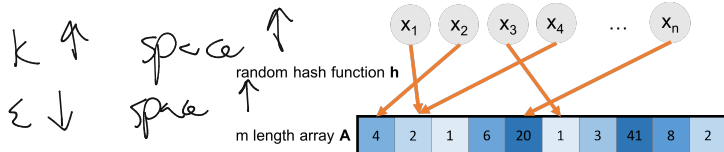


Claim: For any x , with probability at least $1/2$,

$$\widetilde{f(x)} \\ f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy



Claim: For any x , with probability at least $\frac{1}{2}$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$

$$\frac{2n}{m} = \frac{2n}{\frac{2k}{\epsilon}} = \frac{\epsilon n}{k}$$

To solve the (ϵ, k) -Frequent elements problem, set $m = \frac{2k}{\epsilon}$.

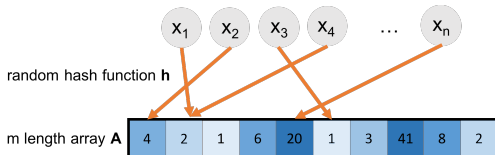
return items where $A[h(x)] \geq \frac{n}{k}$

$$\geq \frac{n}{k} < \frac{n}{k} - \frac{\epsilon n}{k}$$

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

$\rightarrow$ if $f(x) \geq \frac{n}{k}$ $A[h(x)] \geq \frac{n}{k}$ so I return x
 $\rightarrow$ if $f(x) < (1-\epsilon)\frac{n}{k}$ $A[h(x)] \leq f(x) + \frac{\epsilon n}{k} < \frac{n}{k}$ so I don't return x

Count-Min Sketch Accuracy



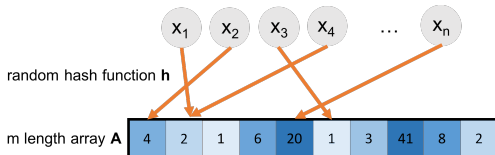
Claim: For any x , with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$

To solve the (ϵ, k) -Frequent elements problem, set $m = \frac{2k}{\epsilon}$. { How can we improve the success probability? }

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.

Count-Min Sketch Accuracy



Claim: For any x , with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$

To solve the (ϵ, k) -Frequent elements problem, set $m = \frac{2k}{\epsilon}$. How can we improve the success probability? **Repetition.**

$f(x)$: frequency of x in the stream (i.e., number of items equal to x). h : random hash function. m : size of Count-min sketch array.