COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2024.

Lecture 7

Summary

Last Class:

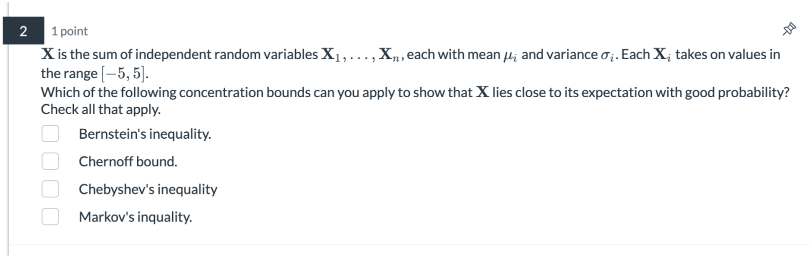
- Finish up exponential concentration bounds. Application to max load in hashing/load balancing.
- Bloom filters for storing a set with a small false positive rate.
- Start on Bloom filter analysis.

This Class:

- Finish Bloom filters.
- Start on streaming algorithms
- Frequent items estimation via Count-Min sketch

Quiz

- Average time spent on homework: approx. 20 hours. If you spent way more than this and would like to chat, send me a message.
- 13 people worked alone, 149 worked in groups. Mix of approaches to splitting up work in groups.
- A fair number of people reported just splitting up the problems – I strongly recommend not doing this.



2 1 point

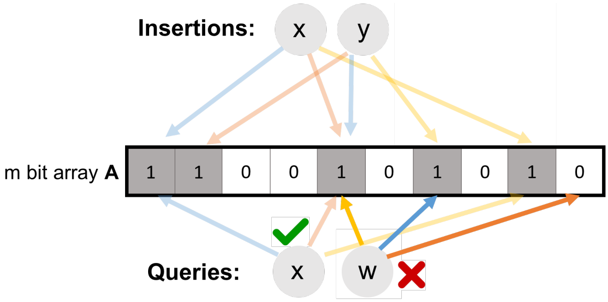
$\mathbf{X}$ is the sum of independent random variables $\mathbf{X}_1, \ldots, \mathbf{X}_n$, each with mean μ_i and variance σ_i. Each $\mathbf{X}_i$ takes on values in the range $[-5, 5]$.
Which of the following concentration bounds can you apply to show that $\mathbf{X}$ lies close to its expectation with good probability? Check all that apply.

- [ ] Bernstein's inequality.
- [ ] Chernoff bound.
- [ ] Chebyshev's inequality
- [ ] Markov's inquality.

Bloom Filters

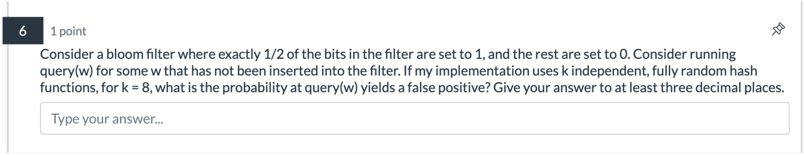
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



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

Bloom Filter Quiz Question


6 1 point

Consider a bloom filter where exactly 1/2 of the bits in the filter are set to 1, and the rest are set to 0. Consider running query(w) for some w that has not been inserted into the filter. If my implementation uses k independent, fully random hash functions, for k = 8, what is the probability at query(w) yields a false positive? Give your answer to at least three decimal places.

Type your answer...

Analysis

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

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

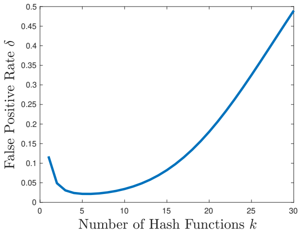
Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}\Pr\left(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1\right) &= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\ &= \left(1 - e^{-\frac{kn}{m}}\right)^k\end{aligned}$$

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

Optimizing Parameters

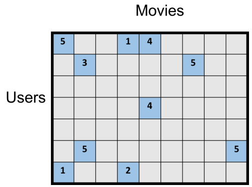
False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$. How should we set k to minimize the FPR given a fixed amount of space m?



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



- 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 $\approx \left(1 - e^{-k \cdot \frac{n}{m}}\right)^k \approx \frac{1}{2^k} \approx \frac{1}{2^{5.54}} = .021$.

Bloom Filter Note

An observation about 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: $k = \ln 2 \cdot \frac{m}{n}$.

Think Pair Share: 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:

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

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

Questions on Bloom Filters?

Streaming Algorithms

Stream Processing: Have a massive dataset X with n items $x_1, x_2, \ldots, x_n$ 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 (Smart Cities), 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

The Frequent Items Problems

k-**Frequent Items (Heavy-Hitters) Problem**: Consider a stream of n items $x_1, \ldots, x_n$ (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	x_1
5	12	3	3	4	5	5	10	3	**5**

- 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?

The Frequent Items Problem

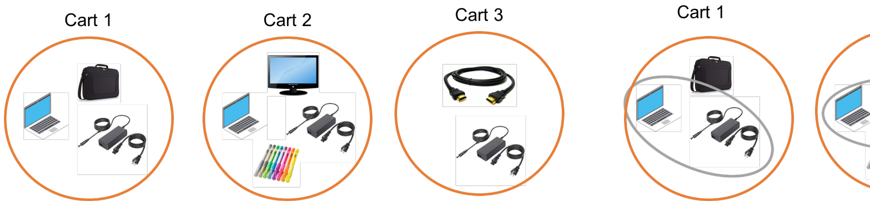
Applications of Frequent Items:

- 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

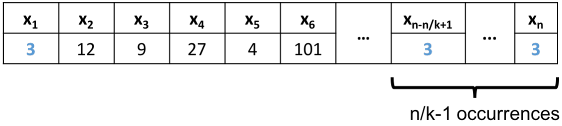
Association rule learning: A very common task in data mining is to identify common associations between different events.



- 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.

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

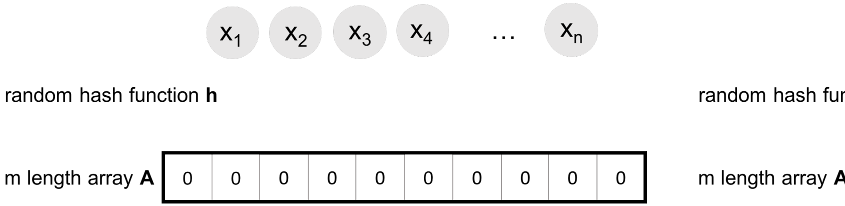
n/k-1 occurrences

(ϵ, k)-Frequent Items Problem: Consider a stream of n items $x_1, \ldots, x_n$. 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.

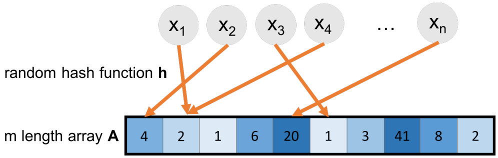
Frequent Elements with Count-Min Sketch

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



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

Count-Min Sketch Accuracy



Use $A[\mathbf{h}(x)]$ to estimate $f(x)$.

Claim 1: We always have $A[\mathbf{h}(x)] \geq f(x)$. Why?

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

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