CS-GY 6763/CS-UY 3943: Lecture 1
Course introduction, concentration of random variable, applications

NYU Tandon School of Engineering, Prof. Christopher Musco

Algorithmic Machine Learning and Data Science

Statistics, machine learning, and data science study how to use data to make better decisions or discoveries.

In this class, we study how to do so as quickly as possible, or with limited computational resources.

APPLICATIONS BY THE NUMBERS

- **ChatGPT** processes 1 billion queries per day, at at a cost of $700,000+ per day for OpenAI.
- **Google** receives $\approx$ 20,000 Maps queries every second.
- **NASA** collects 6.4 TB of satallite images every day.
- **Rubin Observatory in Chile** will collect 20 TB of images every night.
- **MIT/Harvard Broad Institute** sequences 24 TB of genetic data every day.

ROLE OF ALGORITHMS

Growing demands of data science and machine learning have ushered in a new "golden age" for algorithms research.

- Slowing raw performance increases in CPUs + GPUs.
- Parallelization limited by financial and environmental costs. Currently, data centers account for 5% of US electricity use. Expected to double in next 3 years.

Typical data applications require combining a diverse set of algorithmic tools. Most are not heavily covered in your traditional algorithms curriculum.

CLASS TOPICS

(1) Randomized methods.

(2) Optimization.

(3) Spectral methods (linear algebra) and Fourier methods.

Focus is on teaching tools to design algorithms, not just the algorithms themselves.

Section 1: Randomized Algorithms.

It is hard to find an algorithms paper in 2025 that does not use randomness in some way, but this wasn't always the case!

- Probability tools and concentration of random variables (Markovs, Chebyshev, Chernoff/Bernstein inequalities).
- Random hashing for fast data search, load balancing, faster language models, and more. Locality sensitive hashing, MinHash, SimHash, etc.
- Sketching and streaming algorithms for compressing and processing data on the fly.
- High-dimensional geometry and the Johnson-Lindenstrauss lemma for compressing high dimensional vectors.

Section 2: Optimization.

Optimization has become the algorithmic workhorse of modern machine learning.

- Gradient descent, stochastic gradient descent, coordinate descent, and how to analyze these methods.
- Acceleration, conditioning, preconditioning, adaptive gradient methods.
- Constrained optimization, linear programming. Ellipsoid and interior point methods.
- Discrete optimization, relaxation, submodularity and greedy methods.

Section 3: Spectral methods and linear algebra.

"Complex math operations (machine learning, clustering, trend detection) [are] mostly specified as linear algebra on array data" – Michael Stonebraker, Turing Award Winner

- Efficient algorithms for singular value decomposition and eigendecomposition, including randomized methods.
- Spectral graph theory: i.e. how to use linear algebara to understanding large graphs through linear algebra (social networks, interaction graphs, etc.).
- Spectral clustering and non-linear dimensionality reduction.
- Compressed sensing, sparse recovery, and their applications.
- Fast Fourier Transform inspired methods in linear algebra and dimensionality reduction.

WHAT WE WON'T COVER

Software tools or frameworks. Spark, Torch, Tensorflow, HPC, AWS, etc. If you are interested, CS-GY 6513 might be a good course.

Machine Learning Models + Techniques. Neural nets, genertive models, reinforcement learning, Bayesian methods, unsupervised learning, etc. I assume you have already had a course in ML and the focus of this class is on computational considerations.

But if your research is in machine learning, I think you will find the theoretical tools we learn are more broadly applicable than in designing faster algorithms.

OUR APPROACH

This is primarily a **theory** course.

- Emphasis on proofs of correctness, bounding asymptotic runtimes, convergence analysis, etc. *Why?*
- Learn how to model complex problems in simple ways.
- Learn general mathematical tools that can be applied in a wide variety of problems (in your research, in industry, etc.)
- The homework requires **creative problem solving** and thinking beyond what was covered in class. You will not be able to solve many problems on your first try!

You will need a good background in **probability** and **linear algebra**. See the syllabus for more details. Ask me is you are still unsure.

All of this information is on the course webpage `https://www.chrismusco.com/amlds2025/` and in the syllabus posted there! Please take a look.

Class structure:

- Lecture once a week. You can attend on Zoom, but I recommend coming in person.
- Office hours from me and TAs once a week.

Tech tools:

- **Website** for up-to-date info, lecture notes, readings.
- **Ed Discussion** for questions about material.
- **Gradescope** for turning in assignments. Sign up using course code.

Class work:

- **5 problem sets** (45% of course grade).
  - These are challenging, and the most effective way to learn the material. I recommend you start early, work with others, ask questions on Ed, etc.
  - You must write-up solutions on your own.[1]
- **Midterm (Mar. 14th)** 25% of course grade).

[1] 10% bonus on first problem set for using Markdown or LaTex. It should save you time in the long run!

Final project or final exam (20% of grade):

- Final exam will be similar to midterm and problem sets.
- Final project can be based on a recent algorithms paper, and can be either an experimental or theoretical project. Must work in a group of 2 or 3.
- We will hold a **reading group** outside of class for those who decide to complete a final project to workshop topics and papers.
- Others can join as well – it's a great opportunity to get better at reading and presenting papers.

Class participation (10% of grade):

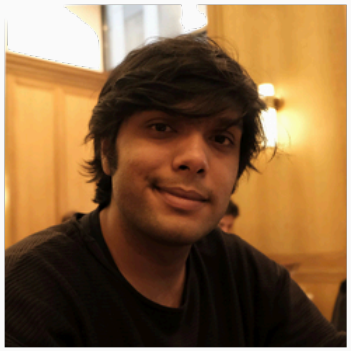
- My goal is to know you all individually by the end of this course.
- Lots of ways to earn the full grade: participation in lecture, office hours, or Ed discussion. Participation in the reading group. Effort on the project.

Important note:

- This is a mixed undergraduate/graduate course.
- Workload is the same, but undergraduates are graded on a different “curve”.

ROCKSTAR TAS

Course Assistant,
Recitation Leader
Noah Amsel



Course Assistant
Pratyush Avi

Both will hold office hours, which are a good place to work on homeworks. Noah will also host “problem solving” recitations for more practice with proof writing. Times TBA.

QUESTIONS?

THIS CLASS

Goal: Demonstrate how even the simplest tools from probability can lead to a powerful algorithmic results.

Lecture applications:

- Estimating set size from samples.
- Finding frequent items with small space.

Problem set applications:

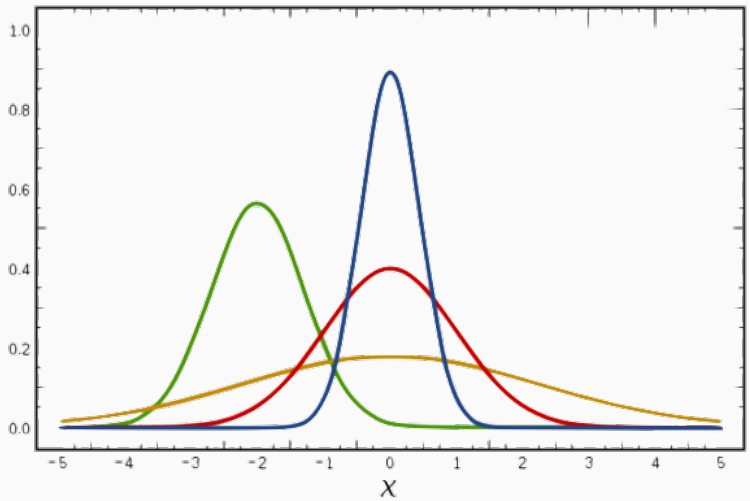
- Group testing for diseases (like bird flu, COVID-19, etc.)
- Efficient packet routing over networks.

Let X be a random variable taking value in some set $\mathcal{S}$. I.e. for a dice, $\mathcal{S} = \{1, \ldots, 6\}$. For a continuous r.v., we might have $\mathcal{S} = \mathbb{R}$.

- **Expectation**: $\mathbb{E}[X] = \sum_{s \in \mathcal{S}} \Pr[X = s] \cdot s$

  For continuous r.v., $\mathbb{E}[X] = \int_{s \in \mathcal{S}} \Pr(s) \cdot s \, ds$.

- **Variance**: $\mathrm{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2]$



For any scalar α, $\mathbb{E}[\alpha X] = \alpha \mathbb{E}[X]$. $\mathrm{Var}[\alpha X] = \alpha^2 \mathrm{Var}[X]$.

Let A and B be random events.

- **Joint Probability**: $\Pr(A \cap B)$. Probability that both events happen.
- **Conditional Probability**: $\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)}$. Probability A happens conditioned on the event that B happens.
- **Independence**: A and B are independent events if: $\Pr(A \mid B) = \Pr(A)$.

Alternative definition of independence:

$$\Pr(A \cap B) = \Pr(A) \cdot \Pr(B).$$

Example: What is the probability that for two independent dice rolls taking values uniformly in $\{1, 2, 3, 4, 5, 6\}$, the first roll comes up odd and the second is < 3?

Let X and Y be random variables. X and Y are independent if, for all events s, t, the random events $[X = s]$ and $[Y = t]$ are independent.

THE MOST POWERFUL THEOREM IN ALL OF PROBABILITY?

Linearity of expectation:

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

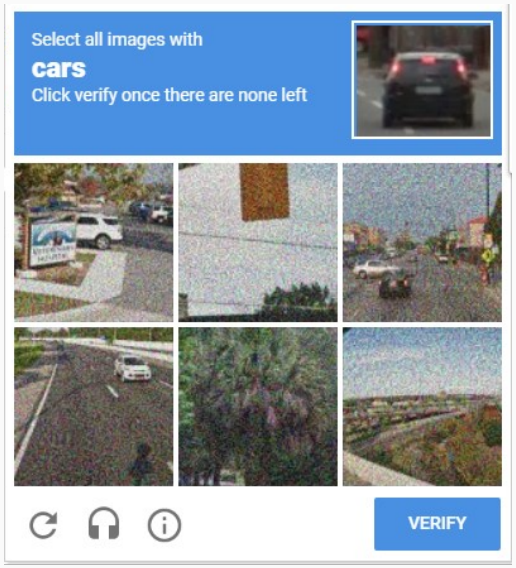
Always, sometimes, or never?

For random variables X, Y:

- $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$.
- $\mathrm{Var}[X + Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]$.
- $\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

FIRST APPLICATION

You run a web company that is considering contracting with a vendor that provides CAPTCHAs for logins.



Claim to have a database of $n = 1,000,000$ unique CAPTCHAs. A random one will be shown on each API call to their service. They give you access to a test API so you can try it out.

Question: Roughly how many queries to the API, m, would you need to independently verify the claim that there are ~ 1 million unique puzzles?

FIRST APPLICATION

First attempt: Count how many unique CAPTCHAs you see, until you find $1,000,000$ or close to it. Declare that you are satisfied.

As a function of n, roughly how many API queries m do you need?

A DIFFERENT APPROACH

Clever alternative: Count how many duplicate CAPTCHAs you see.

If you see the same CAPTCHA on query i and j, that's one duplicate. If you see the same CAPTCHA on queries i, j, and k, that's three duplicates: (i, j), (i, k), (j, k).

Question: How many duplicates do we expect to see?

Let $D_{i,j} = 1$ if queries i, j return the same CAPTCHA, and 0 otherwise.

This is called an indicator random variable.
$D_{i,j} = \mathbb{1}[\text{CAPTCHA } i \text{ equals CAPTCHA } j]$.

Number of duplicates D is :

$$D = \sum_{\substack{i,j \in \{1,\ldots,m\} \\ i<j}} D_{i,j}.$$

What is $\mathbb{E}[D]$?

Question: How many duplicates do we expect to see? Formally, what is $\mathbb{E}[D]$?

$$\mathbb{E}[D] =$$

$n =$ number of CAPTCHAS in database, $m =$ number of test queries.
$D_{i,j} =$ indicator for event CAPTCHA i and j collide.

SOME HARD NUMBERS

Suppose you take $m = 1000$ queries and see 10 duplicates. How does this compare to the expectation if the database actually has $n = 1,000,000$ unique CAPTCHAs?

$$\mathbb{E}[D] =$$

Something seems wrong... this random variable D came up much larger than it's expectation.

Can we say something formally?

$n =$ number of CAPTCHAS in database, $m =$ number of test queries.

CONCENTRATION INEQUALITY

One of the most important tools in analyzing randomized algorithms. Tell us how likely it is that a random variable X deviates a certain amount from its expectation $\mathbb{E}[X]$.

We will learn three fundamental concentration inequalities:

1. Markov's Inequality.
    - Applies to non-negative random variables.
2. Chebyshev's Inequality.
    - Applies to random variables with bounded variance.
3. Hoeffding/Bernstein/Chernoff bounds.
    - Apply to sums of independent random variables.

Theorem (Markov's Inequality): For any random variable X which only takes non-negative values, and any positive t,

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}.$$

Equivalently,

$$\Pr[X \geq \alpha \cdot \mathbb{E}[X]] \leq \frac{1}{\alpha}.$$

Proof:

APPLICATION TO CAPTCHA PROBLEM

Suppose you take $m = 1000$ queries and see 10 duplicates. How does this compare to the expectation if the database actually has $n = 1,000,000$ unique CAPTCHAs?

$$\mathbb{E}[D] = \frac{m(m-1)}{2n} = .4995.$$

By Markov's:

$$\Pr[D \geq 10] \leq \frac{\mathbb{E}[D]}{10} < .05 \text{ if } n \text{ actually equals 1 million.}$$

We can be pretty sure we're being scammed...

$n =$ number of CAPTCHAS in database, $m =$ number of test queries.

Alternative view: If $\mathbb{E}[D] = \frac{m(m-1)}{2n}$, then a natural estimator for n is:

$$\tilde{n} = \frac{m(m-1)}{2D}.$$

With a little more work it is possible to show the following:

Claim: If $m = O\left(\frac{\sqrt{n}}{\epsilon}\right)$, then with probability 9/10, $(1-\epsilon)n \leq \tilde{n} \leq (1+\epsilon)n$. This is a two-sided multiplicative error guarantee. **You will prove this on homework.**

This is a lot better than our original method that required $O(n)$ queries!

Fun facts:

- Known as the “mark-and-recapture” method in ecology.
- Can also be used by webcrawlers to estimate the size of the internet, a social network, etc.

This is also closely related to the birthday paradox.

Linearity of Expectation + Markov's Inequality

Primitive but powerful toolkit, which can be applied to a wide variety of applications!

THE FREQUENT ITEMS PROBLEM

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

$x_1,\ x_2,\ x_3, x_4, x_5,\ x_6, \ldots$

$v_{10}, v_{10}, v_2, v_5, v_{10}, v_2 \ldots$

- 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.

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 of data items.

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

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
5	12	3	3	4	5	5	10	3

- Trivial with $O(u)$ space – store the count for each item and return the one that appears $\geq n/k$ times.

Example where linear dependence on u is too large: Find common subsets within a collection of sets. Each subset is an "item".

- For product recommendations, the number of pairs of products might grow quadratically with the number of products. Amazon has 12 million products. (12 million) $\times$ 4 bytes $=$ 48 megabytes . (12 million)^2 $\times$ 4 bytes $=$ 576 terabytes to maintain counts.
- For social media recommendations, we might have a set of followers for each user and want to count frequent subsets of who they follow. Even higher complexity.

APPROXIMATE FREQUENT ELEMENTS

Issue: Can prove that no algorithm using $o(u)$ space can output just the items with frequency $\geq n/k$. We will only be able to solve the problem approximately.

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

The deterministic **Misra-Gries algorithm** solves this problem using $O(k/\epsilon)$ space. We will see a randomized algorithm that matches this, and is more flexible in many settings.

FREQUENT ELEMENTS WITH COUNT-MIN SKETCH

Today: Count-Min Sketch – a random hashing based method for the frequent elements problem.

Due to a 2005 paper by Graham Cormode and Muthu Muthukrishnan.

Solves the slightly different **point query** problem. Given any value v, let $f(v) = \sum_{i=1}^{n} \mathbb{1}[x_i = v]$ be the number of times v appears in the stream.

Goal: Return estimate $\tilde{f}(v)$ such that $f(v) \leq \tilde{f}(v) \leq f(v) + \frac{\epsilon}{k} n$ with high probability.

Solving Frequent items: Just return all items for which $\tilde{f}(v) \geq \frac{n}{k}$.

RANDOM HASH FUNCTION

Let h be a random function from $|\mathcal{U}| \to \{1, \ldots, m\}$. This means that h is constructed by an algorithm using a seed of random numbers, but then the function is fixed. Given input $x \in \mathcal{U}$, it always returns the same output, $h(x)$.

Definition: Uniformly Random Hash Function. A random function $h : \mathcal{U} \to \{1, \ldots, m\}$ is called uniformly random if:

- $\Pr[h(x) = i] = \frac{1}{m}$ for all $x \in \mathcal{U}$, $i \in \{1, \ldots, m\}$.
- $h(x)$ and $h(y)$ are independent r.v.'s for all $x, y \in \mathcal{U}$.
    - Which implies that $\Pr[h(x) = h(y)] =$

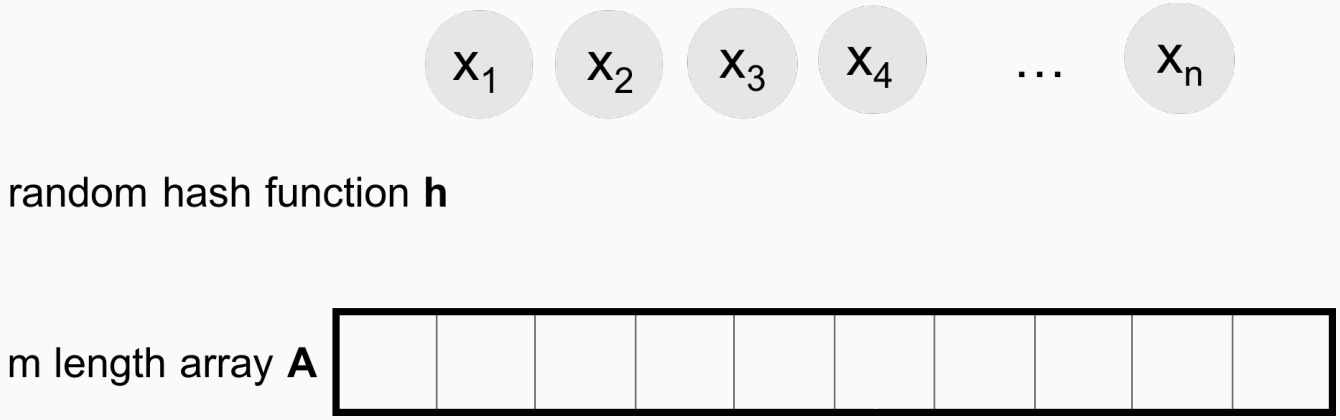
$\mathcal{U}$ = universe of possible keys, m = number of values hashed to.

Caveat: It is not possible to efficiently implement uniform random hash functions! But:

- In practice "random looking" functions suffice.
- We will discuss weaker, efficiently implementable hash functions (in particular, universal hash functions) next week.
- Our analysis will work with these weaker hash functions.

But, we will make our lives easier by **assuming** we have access to a uniformly random hash function. This is an assumption we will use in future lectures as well. The assumption is often made in reseach papers even.

COUNT-MIN SKETCH



Count-Min Update:

- Choose random hash function h mapping to $\{1, \ldots, m\}$.
- For $i = 1, \ldots, n$
  - Given item x_i, set $\mathbf{A}[h(x_i)] = \mathbf{A}[h(x_i)] + 1$

h: random hash function. m: size of Count-Min sketch array.

We want to estimate the frequency of item v, $f(v) = \sum_{i=1}^{n} \mathbb{1}[x_i = v]$. To do this using our small space "sketch" $\mathbf{A}$, return $\tilde{f}(v) = A[\mathbf{h}(v)]$.

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

$f(v)$: frequency of v in the stream. h: random hash function. m: size of Count-Min sketch array.

COUNT-MIN SKETCH ACCURACY

$$\mathsf{A}[\mathsf{h}(v)] = f(v) + \underbrace{\sum_{y \neq v} \mathbb{1}[\mathsf{h}(y) = \mathsf{h}(v)] \cdot f(y)}_{\text{error in frequency estimate}}$$

Expected Error:

$$\mathbb{E}\left[\sum_{y \neq v} \mathbb{1}[\mathsf{h}(y) = \mathsf{h}(v)] \cdot f(y)\right] =$$

COUNT-MIN SKETCH ACCURACY

$$\mathsf{A}[\mathsf{h}(v)] = f(v) + \underbrace{\sum_{y \neq v} \mathbb{1}[\mathsf{h}(y) = \mathsf{h}(v)] \cdot f(y)}_{\text{error in frequency estimate}}$$

Expected Error:

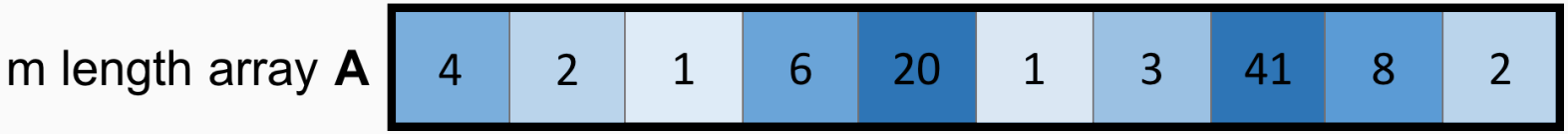
$$\mathbb{E}\left[\sum_{y \neq v} \mathbb{1}[\mathsf{h}(y) = \mathsf{h}(v)] \cdot f(y)\right] \leq \frac{n}{m}$$

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

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

$f(v)$: frequency of v in the stream. h: random hash function. m: size of Count-Min sketch array.

COUNT-MIN SKETCH ACCURACY



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

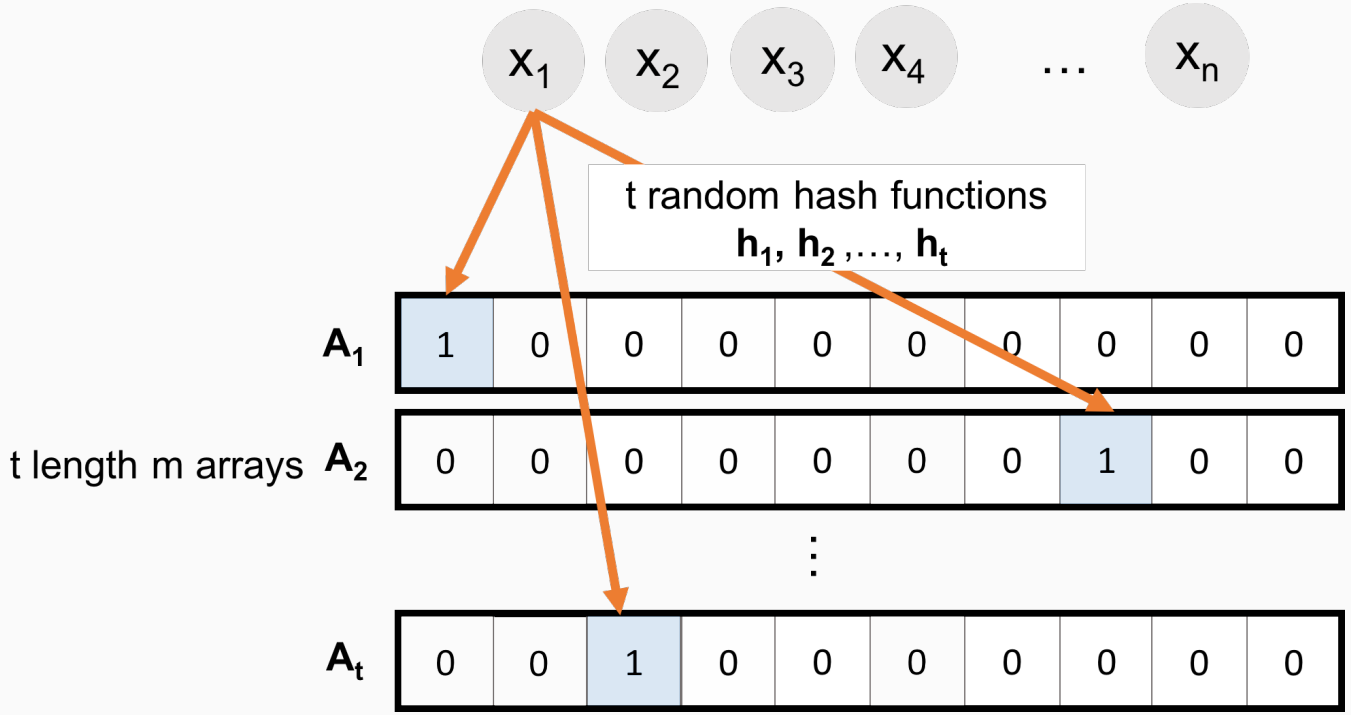
$$f(v) \leq \mathbf{A}[\mathbf{h}(v)] \leq f(v) + \frac{2n}{m}.$$

To solve the point query problem with error $\frac{\epsilon}{k}n$, set $m =$

How can we improve the success probability?

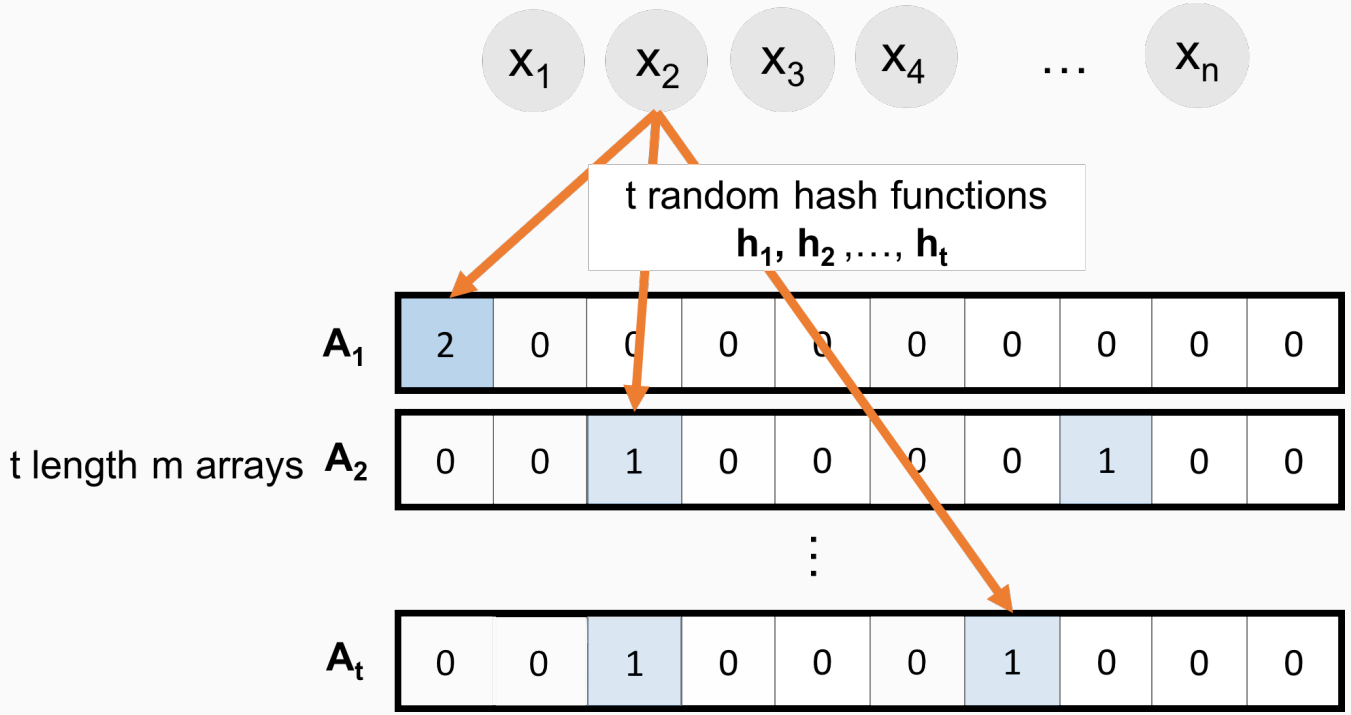
$f(v)$: frequency of v in the stream. h: random hash function. m: size of Count-Min sketch array.

COUNT-MIN SKETCH ACCURACY



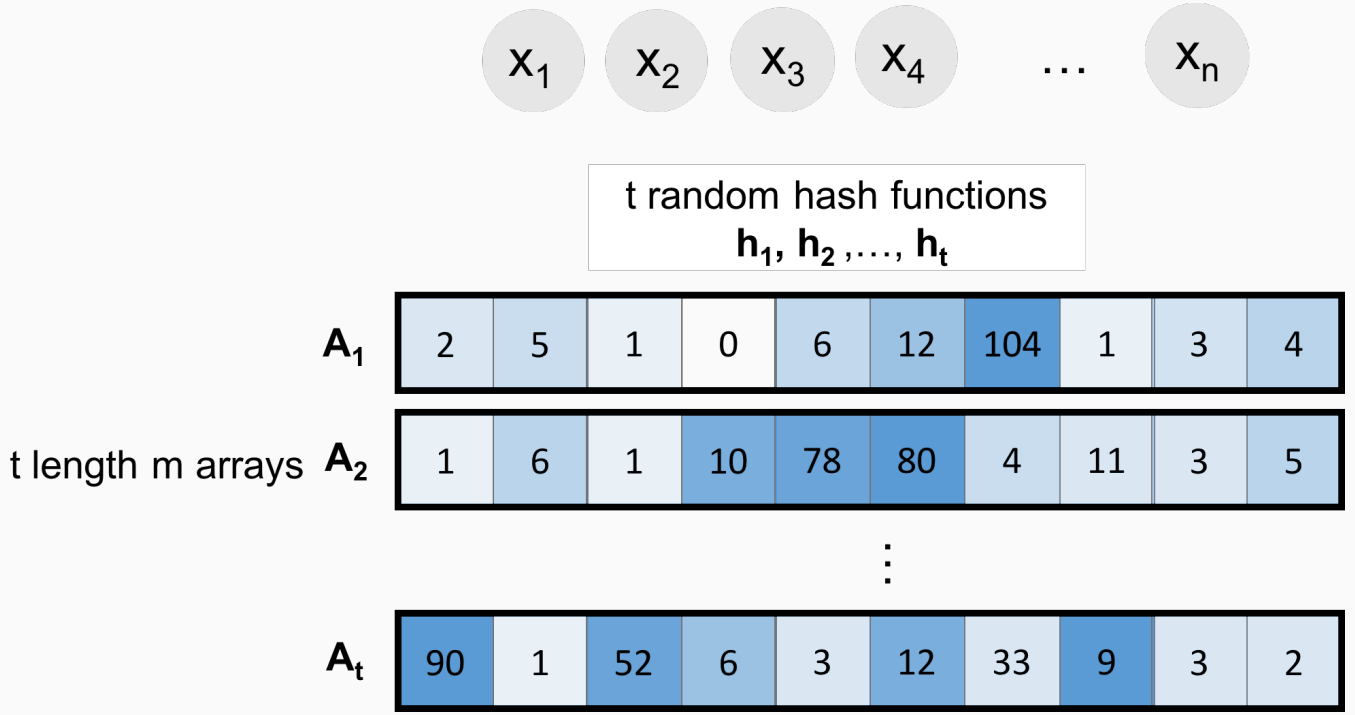
$f(v)$: frequency of v in the stream. $h_1, \ldots, h_t$: multiple random hash functions. m: size of t Count-Min sketch arrays.

COUNT-MIN SKETCH ACCURACY



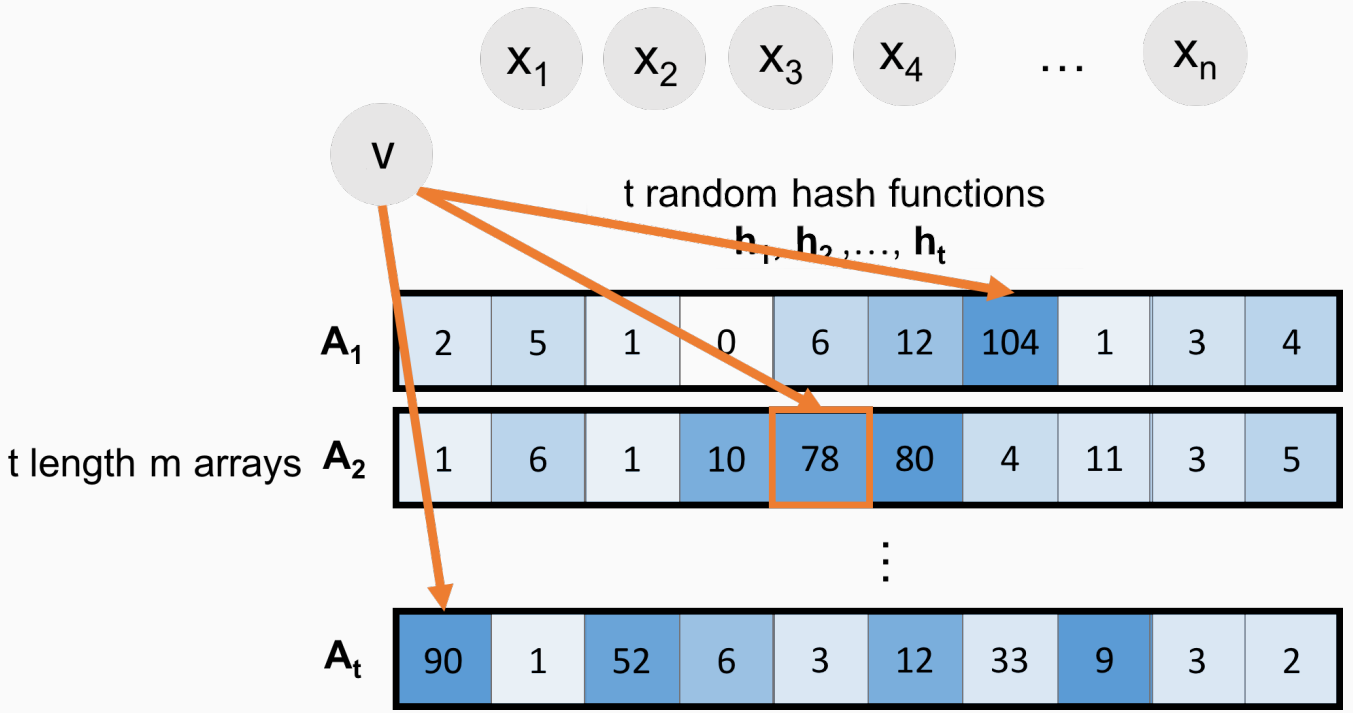
$f(v)$: frequency of v in the stream. $h_1, \ldots, h_t$: multiple random hash functions. m: size of t Count-Min sketch arrays.

COUNT-MIN SKETCH ACCURACY



$f(v)$: frequency of v in the stream. $h_1, \ldots, h_t$: multiple random hash functions. m: size of t Count-Min sketch arrays.

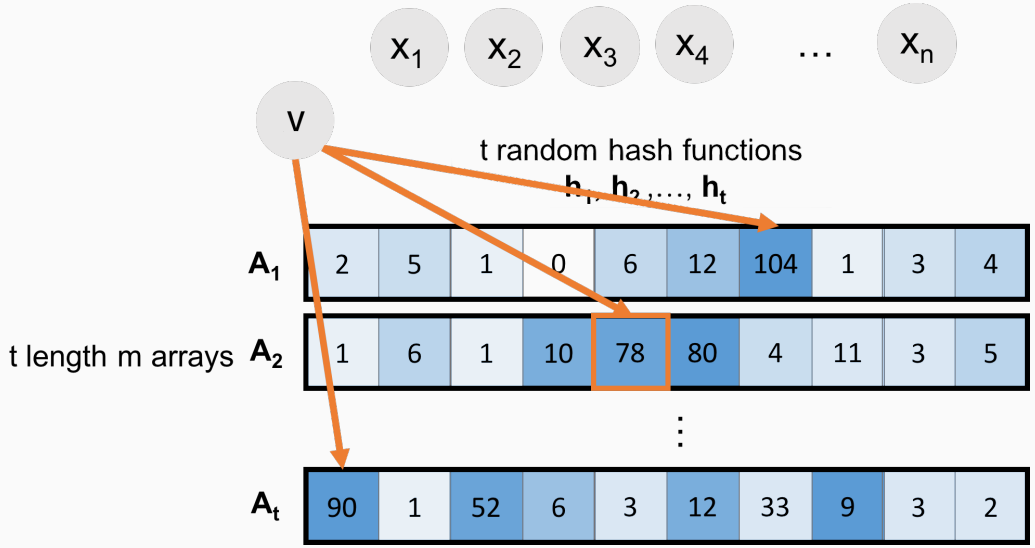
COUNT-MIN SKETCH ACCURACY



Estimate $f(v)$ with $\tilde{f}(v) = \min_{i \in [t]} \mathbf{A}_i[\mathbf{h}_i(v)]$. (Count-Min sketch)

Why min instead of mean or median?

COUNT-MIN SKETCH ACCURACY



Estimate $f(v)$ with $\tilde{f}(v) = \min_{i \in [t]} \mathsf{A}_i[\mathsf{h}_i(v)]$.

- For every v and i and $m = \frac{2k}{\epsilon}$, we know that with prob. $\geq 1/2$:
$$f(v) \leq \mathsf{A}_i[\mathsf{h}_i(v)] \leq f(v) + \frac{\epsilon n}{k}.$$
- $\Pr[f(v) \leq \tilde{f}(v) \leq f(v) + \frac{\epsilon n}{k}] \geq$
- To get a good estimate with probability $\geq 1 - \delta$,

$$\text{set } t =$$

Upshot: Count-Min sketch lets us estimate the frequency of each item in a stream up to error $\frac{\epsilon}{k}n$ with probability $\geq 1 - \delta$ in $O\left(\log(1/\delta) \cdot \frac{k}{\epsilon}\right)$ space.

Caveat: This is a for each v guarantee. We actually want a for all v guarantee: i.e. the bound should hold simultaneously for all v.

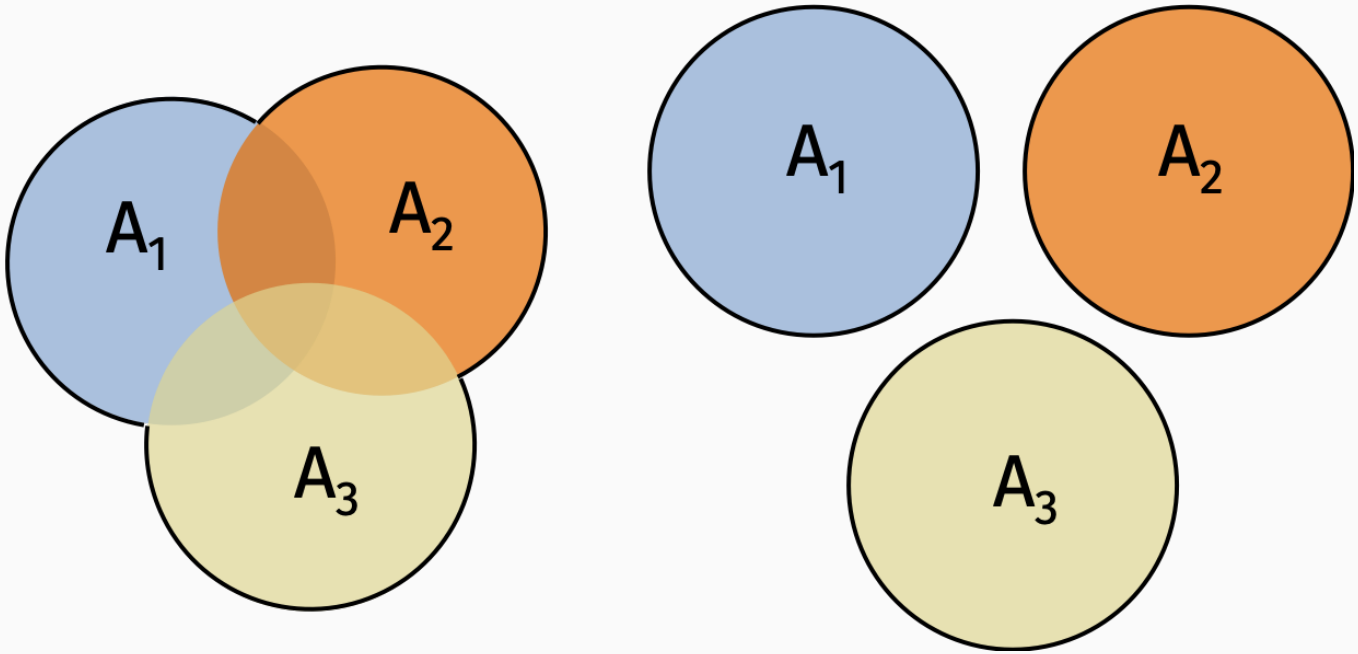
USE A UNION BOUND

Lemma (Union Bound)

For any random events $A_1, \ldots, A_k$:

$$\Pr[A_1 \cup A_2 \cup \ldots \cup A_k] \leq \Pr[A_1] + \Pr[A_2] + \ldots + \Pr[A_k].$$

Here $Pr[A_1 \cup A_2 \cup \ldots \cup A_k]$ means $Pr[A_1$ "or" $A_2 \ldots$ "or" $A_k]$



Proof by picture.

USE A UNION BOUND

The algorithm fails if $|f(v) - \tilde{f}(v)| > \frac{\epsilon}{k} n$ for any $v \in \{v_1, \ldots, v_n\}$.
By union bound:

$$\Pr[(\text{fail for } v_1) \text{ or } (\text{fail for } v_2) \text{ or} \ldots \text{ or} (\text{fail for } v_n)] =$$

FINAL RESULT

Set $\delta = \frac{1}{10n}$. With probability 9/10, Count-Min sketch lets us estimate the frequency of all items in a stream up to error $\frac{\epsilon}{k}n$.

- Accurate enough to solve the (ϵ, k)-Frequent elements problem – just return all v with estimated frequency $\geq n/k$.

IDENTIFYING FREQUENT ITEMS

How do we identify the frequent items without having to look up the estimated frequency for all elements in the stream?

One approach:

- When a new item comes in at step i, check if its estimated frequency is $\geq i/k$ and store it if so.
- At step i remove any stored items whose estimated frequency drops below i/k.
- Store at most $O(k)$ items at once and have all items with estimated frequency $\geq n/k$ stored at the end of the stream.

Can we weaken our assumption that h is uniformly random?

Definition (Universal hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, m\}$ is universal if, for any fixed $x, y \in \mathcal{U}$,

$$\Pr[h(x) = h(y)] \leq \frac{1}{m}.$$

Claim: A uniformly random hash-function is universal.

Efficient alternative: Let p be a prime number between $|\mathcal{U}|$ and $2|\mathcal{U}|$. Let a, b be random numbers in $0, \ldots, p$, $a \neq 0$.

$$h(x) = [a \cdot x + b \pmod{p}] \pmod{m}$$

is universal.

Another definition you might come across:

Definition (Pairwise independent hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, m\}$ is pairwise independent if, for any fixed $x, y \in \mathcal{U}, i, j \in \{1 \ldots, m\}$,

$$\Pr[h(x) = i \cap h(y) = j] = \frac{1}{m^2}.$$

We can naturally extended to k-wise independence for $k > 2$, which is strictly stronger, and needed for some applications.