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

NYU Tandon School of Engineering,
Prof. Rajesh Jayaram

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

- **Twitter** receives 6,000 tweets every second.
- **Google** receives $\approx$ 10,000 Maps queries every second.
- **NASA** collects 6.4 TB of satallite images every day.
- **Large Synoptic Survey Telescope** will collect 20 TB of images every night.
- **MIT/Harvard Broad Institute** sequences 24 TB of genetic data every day.

New Age of Algorithm Design

Modern demands for enormous data have ushered in a new "golden age" for algorithms research.

- Historically, "polynomial time" was the gold standard for algorithms. Today, this is usually no longer nearly good enough.
- Often, we now need linear, or even *sublinear* time and space algorithms. Commonly, data cannot fit in-memory.
- Motivated the study of new computational modes: streaming, distributed algorithms (i.e. MapReduce, Spark),

Course Topics

(1) Randomized methods.
(2) Optimization.
(3) Spectral methods and linear algebra.
(4) Fourier methods & compressed sensing.

Randomized Methods

Section 1: Randomized Algorithms.

It is hard to find an algorithms paper in 2021 that does not use randomness in some way.

- Probability tools and concentration of random variables (Markov, Chebyshev, Chernoff/Bernstein inequalities).
- Random hashing for fast data search, load balancing, 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, and how to analyze these methods.
- Acceleration, conditioning, preconditioning, adaptive gradient methods.
- Discrete optimization: Clustering, submodularity, and greedy methods.

Spectral 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

- Singular value decompositions and eigendecomposition.
- Spectral graph theory: i.e. linear algebraic tools to analyze large graphs (social networks, co-purchased graph, etc.).
- Spectral clustering and non-linear dimensionality reduction.
- Sketching methods for regression & and low-rank approximation.

Fourier Methods

Section 4: Fourier methods.

- Compressed sensing, sparse recovery, and their applications.
- Fast Fourier Transform-based methods.
- Fourier perspective on machine learning techniques like kernel methods, and the algorithmic benefits.

What we won't cover

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

Machine Learning Models + Techniques. Neural nets, 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 powerful 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.

Course Structure and Logistics

All of this information is on the course webpage `https://rajeshjayaram.com/amlds2022` and in the syllabus posted there! Please take a look.

Class structure:

- 2.5 hour lecture once a week, with 15 minute break mid-way.
- Office hours from me and TAs once a week. Mine will be virtual.

Tech tools:

- **Website** for up-to-date info, lecture notes, readings.
- **Ed discussion** for questions about material.
- **NYU Brightspace** for turning in assignments.

Course Structure and Logistics

Class work:

- **4 problem sets** (50% 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** (15% of course grade).

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

Course Structure and Logistics

Final project <u>or</u> final exam (25% 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. Work alone or in a pair.
- Others can join as well – it's a great opportunity to get better at reading and presenting papers.

Course Structure and Logistics

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. Effort on the project.

Course Structure and Logistics

Important note:

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

Course Assistant
Aarshvi Gajjar

Course Assistant
Teal Witter

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:

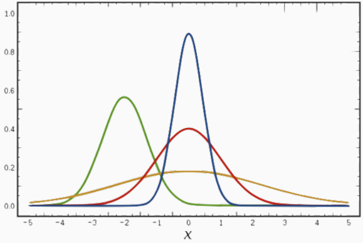
- Group testing for COVID-19.
- Smarter load balancing.
- Estimating Distinct Elements in a Stream

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



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

Probability Review

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: $\Pr(A \mid B) = \Pr(A)$. Equivalently: $\Pr(A \cap B) = \Pr(A) \cdot \Pr(B)$.

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:

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

Related equations

Always, sometimes, or never?

For random variables X, Y:

- $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$.

- $\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$.

- $\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

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

They claim to have a database of $n = \mathbf{1,000,000}$ unique CAPTCHAs in their database, and a random one will be shown on each API call to their service.

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

First Application

First attempt: Count how many unique CAPTCHAs you see, until you find roughly $1,000,000$.

As a function of n, how many queries m do you need to see at least $\frac{9}{10}n$ unique CAPTCHAs?

Bonus: How many queries do you need to see *exactly* n? *This is known as the Coupon Collector Problem*

An Improved 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).

An Improved Approach

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 $\mathbb{E}[D]$?

An Improved Approach

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?

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

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 Inequalities

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.
    - Applies to sums of independent random variables with bounded variance.

Markov's inequality

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

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

Equivalently,

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

Markov's inequality

Theorem (Markov's Inequality): For any random variable X which only takes non-negative values 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:

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{s \in S} \Pr[X = s] \cdot s \\
&= \sum_{s < t} \Pr[X = s] \cdot s + \sum_{s \geq t} \Pr[X = s] \cdot s \\
&\geq 0 + t \cdot \sum_{s \geq t} \Pr[X = s] = t \cdot \Pr[X \geq t]
\end{aligned}$$

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.

General Bound

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

Lemma

Setting $m = \Omega\left(\frac{\sqrt{n}}{\epsilon}\right)$ *and* $\tilde{n} = \frac{m(m-1)}{2D}$, *then with prob.* $\frac{9}{10}$*:*

$$(1-\epsilon)n \leq \tilde{n} \leq (1+\epsilon)n$$

This is a two-sided multiplicative $(1 \pm \epsilon)$ error guarantee — the gold standard in this course.

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

Linearity of Expectation + Markov's Inequality

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

But, cannot bound probability that R.V.'s are small: i.e.

$$\Pr[X \ll \mathbb{E}[X]]$$

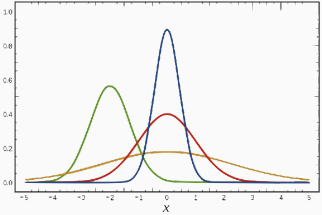
Chebyshev's Inequality

A new concentration inequality:

Lemma (Chebyshev's Inequality)

Let X be a random variable with expectation $\mathbb{E}[X]$ and variance $\sigma^2 = \text{Var}[X]$. Then for any $k > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq k \cdot \sigma] \leq \frac{1}{k^2}$$



$\sigma = \sqrt{\text{Var}[X]}$ is the standard deviation of X. Intuitively this bound makes sense: it is tighter when σ is smaller.

Chebyshev's vs. Markov's

Properties of Chebyshev's inequality:

- **Good:** No requirement of non-negativity. X can be anything.
- **Good:** Two-sided. Bounds the probability that $|X - \mathbb{E}X|$ is large, which means that X isn't too far above <u>or</u> below its expectation. Markov's only bounded probability that X exceeds $\mathbb{E}[X]$.
- **Bad/Good:** Requires a bound on the variance of of X.

No hard rule for which to apply! Both Markov's and Chebyshev's are useful in different settings.

Proof of Chebyshev's Inequality

Idea: Apply Markov's inequality to the (non-negative) random variable $S = (X - \mathbb{E}[X])^2$. Recall $\text{Var}[D] = \mathbb{E}[(X - \mathbb{E}[x])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

Lemma (Chebyshev's Inequality)

Let X be a random variable with expectation $\mathbb{E}[X]$ and variance $\sigma^2 = \text{Var}[X]$. Then for any $k > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq k \cdot \sigma] \leq \frac{1}{k^2}$$

Markov's inequality: for positive r.v. S, $\Pr[S \geq t] \leq \mathbb{E}[S]/t$.

Proof of Captcha Lemma

Lemma

Setting $m = \Omega\left(\frac{\sqrt{n}}{\epsilon}\right)$ *and* $\tilde{n} = \frac{m(m-1)}{2D}$, *then with prob* $\frac{9}{10}$*:*

$$(1-\epsilon)n \leq \tilde{n} \leq (1+\epsilon)n$$

By rearranging, it suffices to show

$$\frac{1}{1+\epsilon} \cdot \binom{m}{2}\frac{1}{n} \leq D \leq \frac{1}{1-\epsilon} \cdot \binom{m}{2}\frac{1}{n}$$

Where

$$D = \sum_{\substack{i,j \in \{1,\ldots,m\} \\ i<j}} \mathbb{1}[x_i = x_j].$$

and $x_1, \ldots, x_m$ are uniformly random Captchas. Recall that $\mathbb{E}[D] = \binom{m}{2}\frac{1}{n}$.

Proof of Captcha Lemma

Using the notation $D_{i,j} = \mathbb{1}[x_i = x_j]$:

$$\mathbb{E}\left[\sum_{i<j\in[m]} D_{i,j}\right]^2 \leq \sum_{i,j\in[m]} \mathbb{E}[D_{i,j}^2] + \sum_{i,j,k\in[m]} \mathbb{E}[D_{i,j}D_{i,k}] + \sum_{i,j,k,t\in[m]} \mathbb{E}[D_{i,j}D_{t,k}]$$

$$= \binom{m}{2}\frac{1}{n} + \binom{m}{3}\frac{1}{n^2} + \binom{m}{4}\frac{1}{n^2}$$

$$\leq \frac{m^2}{2n} + \frac{m^3}{6n^2} + \frac{m^4}{24n^2} \leq \frac{m^2}{2n} + \frac{m^4}{4n^2}$$

Proof of Captcha Lemma

Using the notation $D_{i,j} = \mathbb{1}[x_i = x_j]$:

$$
\begin{aligned}
\mathbb{E}\left[\sum_{i<j\in[m]} D_{i,j}\right]^2 &\le \sum_{i,j\in[m]} \mathbb{E}[D_{i,j}^2] + \sum_{i,j,k\in[m]} \mathbb{E}[D_{i,j}D_{i,k}] + \sum_{i,j,k,t\in[m]} \mathbb{E}[D_{i,j}D_{t,k}] \\
&= \binom{m}{2}\frac{1}{n} + \binom{m}{3}\frac{1}{n^2} + \binom{m}{4}\frac{1}{n^2} \\
&\le \frac{m^2}{2n} + \frac{m^3}{6n^2} + \frac{m^4}{24n^2} \le \frac{m^2}{2n} + \frac{m^4}{4n^2}
\end{aligned}
$$

So

$$
\begin{aligned}
\mathrm{Var}[D] &\le \frac{m^2}{2n} + \frac{m^4}{4n^2} - \mathbb{E}[D]^2 \\
&= \frac{m^2}{2n} + \frac{m^4}{4n^2} - \frac{m^2(m-1)^2}{4n^2} \\
&\le \frac{m^2}{2n}
\end{aligned}
$$

Proof of Captcha Lemma

We have $\mathrm{Var}[D] = \frac{m^2}{2n}$. So setting $m = 10\frac{\sqrt{n}}{\epsilon}$, and plugging this into Chebyshev's Inequality:

$$\Pr\left[\left|D - \binom{m}{2}\frac{1}{n}\right| \geq \epsilon\binom{m}{2}\frac{1}{n}\right] \leq \frac{m^2}{2n} \cdot \left(\frac{2n}{\epsilon m(m-1)}\right)^2$$
$$\leq \frac{4}{\epsilon^2} \cdot \frac{n}{m^2} \leq \frac{1}{25}$$

So

$$\Pr\left[(1-\epsilon)\binom{m}{2}\frac{1}{n} \leq D \leq (1+\epsilon)\binom{m}{2}\frac{1}{n}\right] > \frac{24}{25}$$

From which the Lemma follows.

Chebyshev's Inequality: $\Pr[|X - \mathbb{E}[X]| \geq k] \leq \frac{\mathrm{Var}[X]}{k^2}$

Mark and Recaptcha

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.

Streaming Algorithms

Important Model of Modern Computation: Motivating assumption is the data is too large to fit in memory.

- **Streaming data** which arrives in real time: e.g. IP traffic logs, financial transactions, Google search queries...
- **Large Distributed Databases:** must be processed while making a *single pass* over the data, using a small amount of working memory.

Modeled by a sequence of *updates* to a high-dimensional "frequency" vector $f \in R^n$.

Coordinates (also called "items") $i \in [n]$ can represent e.g. a user, IP address, or Stock, and the frequency f_i stores some associated value for that coordinate (i.e. total network traffic, number of transactions, ect.).

The Streaming Model, Formally

Streaming Model: Let $f \in R^n$ be an implicit, high-dimensional vector, initialized to zero (i.e. $f = \vec{0}$). A *insertion-only* data stream is a sequence of coordinate-wise "updates" $a_1, a_2, \ldots, a_m$, where each $a_t \in [n]$. The update to index $i \in [n]$ causes the change

$$f_{a_t} \leftarrow f_{a_t} + 1$$

- Coordinates can represent IPv6 Addresses, Amazon customers, Google Search Terms.
- n can be very large (e.g. 2^{128} possible IPv6 addresses). Thus, f cannot be maintained explicitly in memory.

A **streaming algorithm** must observe the sequence $a_1, a_2, \ldots, a_m$, and then (approximately) answer queries about the final vector f using *as little space as possible*.

Other examples in practice

Sensor data: GPS or seismometer readings to detect geological anomalies, telescope images, satellite imagery, highway travel time sensors.

Web traffic and data: User data for website, including e.g. click data, web searches and API queries, posts and image uploads on social media.

Training machine learning models: Often done in a streaming setting when training dataset is huge, often with multiple passes.

Lots of software frameworks exist for easy development of streaming algorithms.

Frequent Items/Heavy Hitters

k-**Frequent Items (Heavy-Hitters) Problem**: Consider a stream of m updates $a_1, \ldots, a_m \in [n]$ to the coordinates of $f \in \mathbb{R}^n$. Find all the $(1/k)$-heavy hitters: namely return all items at appears at least $\frac{m}{k}$ times.

- 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.

How much space does a streaming algorithm need to do this?

Frequent Items/Heavy Hitters

k-**Frequent Items (Heavy-Hitters) Problem**: Consider a stream of m updates $a_1, \ldots, a_m \in [n]$ to the coordinates of $f \in \mathbb{R}^n$. Find all the $(1/k)$-heavy hitters: namely return all items at appears at least $\frac{m}{k}$ times.

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
5	12	3	3	4	5	5	10	0	3

- Trivial with $O(n)$ space – store the count for each item and return the one that appears $\geq m/k$ times.
- Possible to do significantly better!
- What is the maximum number of $(1/k)$-Heavy Hitters?

a) m b) k c) n/k d) m/k

Approximate frequent elements

Issue: No algorithm using $o(n)$ space can output just the items with frequency $\geq m/k$.

Intuition: Hard to tell between an item with frequency m/k (should be output) and $m/k - 1$ (should not be output).

Approximate k**-Frequent Items Problem**: Consider a stream of m updates $a_1, \ldots, a_m \in [n]$. Return a set S of items, including all items that appear at least $\frac{m}{k}$ times and no items that appear less than $\frac{1}{2} \cdot \frac{m}{k}$ times.

- For items with frequencies in $[\frac{1}{2} \cdot \frac{m}{k}, \frac{m}{k}]$ no output guarantee.

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.

Has almost 2000 citations!

Hashing

Let h be a random hash function from $[n] \to [B]$. This means that h is a fixed function drawn uniformly from the set of all possible functions $\mathcal{H} = \{g \ : \ g : [n] \to [B]\}$. Once it is fixed, given any 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:

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

$[n] = \{1, \ldots, n\}$ universe of possible keys, $B =$ number of hash buckets.

Hashing

Caveat: It is not possible to efficiently implement uniform random hash functions, would require $\Omega(n)$ space to store the mapping! But:

- In practice "random looking" functions like MD5, SHA256, etc. often suffice.
- If we have time, we will discuss weaker hash functions (in particular, k-wise independent functions) which are hash functions used for theoretical analysis, and which are efficient to implement.

For now, **assume** we have access to a uniformly random hash function h, without worrying about the space needed to store it. This is an assumption we will use in future lectures as well.

Count-Min Sketch

Input: Stream of updates $a_1, \ldots, a_m \in [n]$ to the coordinates of $f \in \mathbb{R}^n$.

array **A**	0	0	0	0	0	0	0	0	0	0

Count-Min Update:

- Choose random hash function $h : [n] \to [B]$
- For each update $i = 1, \ldots, m$
  - Given update a_i, set

$$\mathbf{A}[h(a_i)] = \mathbf{A}[h(a_i)] + 1$$

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

Count-Min Sketch

From small space "sketch" $\mathbf{A} \in \mathbb{R}^B$, we can estimate the frequency of any item f_i, $f_i = \sum_{t=1}^{m} \mathbb{1}[a_t = i]$.

In particular, we simply return $A[h(i)]$.

array **A**	4	2	1	6	20	1	3	41	8	2

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

f_i: frequency of v in the stream. h: random hash function. B: size of Count-Min sketch array.

Count-Min Sketch Accuracy

$$\mathbf{A}[h(i)] = f_i + \underbrace{\sum_{j \neq i} \mathbb{1}[\mathbf{h}(j) = \mathbf{h}(i)] \cdot f_j}_{\text{error in frequency estimate}}$$

Expected Error:

$$\mathbb{E}\left[\sum_{j \neq i} \mathbb{1}[\mathbf{h}(j) = \mathbf{h}(i)] \cdot f(j)\right] =$$

Count-Min Sketch Accuracy

$$\mathbf{A}[h(i)] = f_i + \sum_{j \neq i} \mathbb{1}[\mathbf{h}(j) = \mathbf{h}(i)] \cdot f_j$$

Expected Error:

$$\mathbb{E}\left[\sum_{j \neq i} \mathbb{1}[\mathbf{h}(j) = \mathbf{h}(i)] \cdot f_j\right] \leq \frac{\|f\|_1}{B} = \frac{m}{B}$$

What is a bound on probability that the error is $\geq \frac{2\|f\|_1}{B}$?

Markov's inequality: $\Pr\left[\sum_{j \neq i} \mathbb{1}[\mathbf{h}(j) = \mathbf{h}(i)] \cdot f(j) \geq \frac{2\|f\|_1}{B}\right] \leq$

f_i: frequency of item i in the stream. h: random hash function. B: size of Count-Min sketch array.

Count-Min Sketch Accuracy

array **A**	4	2	1	6	20	1	3	41	8	2

Claim: For any one coordinate $i \in [n]$, with probability $\geq 1/2$,

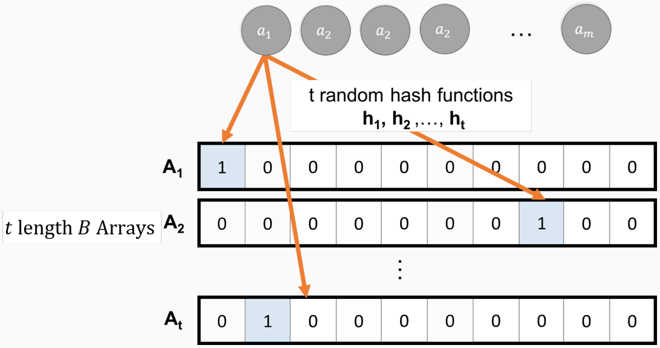
$$f_i \leq \mathbf{A}[\mathbf{h}(i)] \leq f_i + 2\frac{\|f\|_1}{B}$$

To solve the k-Frequent elements problem, set $B =$.

How can we improve the success probability?

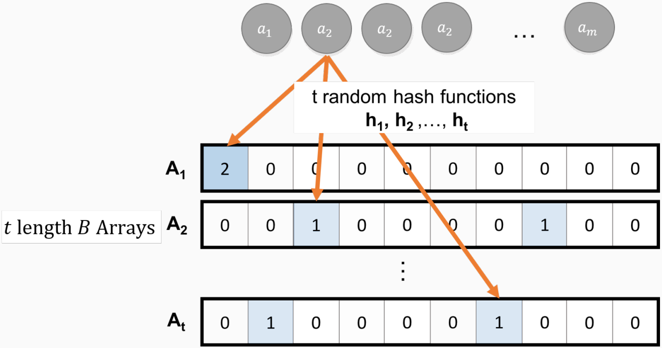
f_i: frequency of item i in the stream. h: random hash function. B: size of Count-Min sketch array.

Count-Min Sketch Accuracy



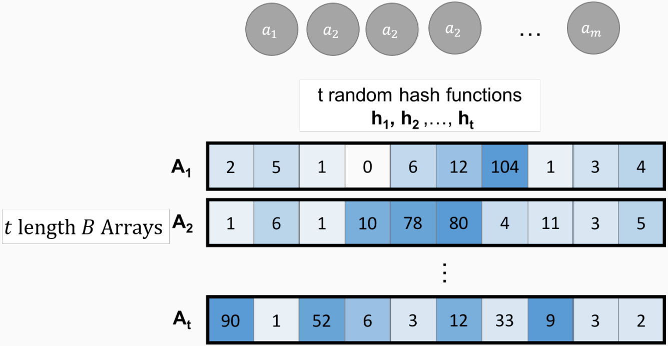
f_i: frequency of item i in the stream. h: random hash function. B: size of Count-Min sketch array.

Count-Min Sketch Accuracy



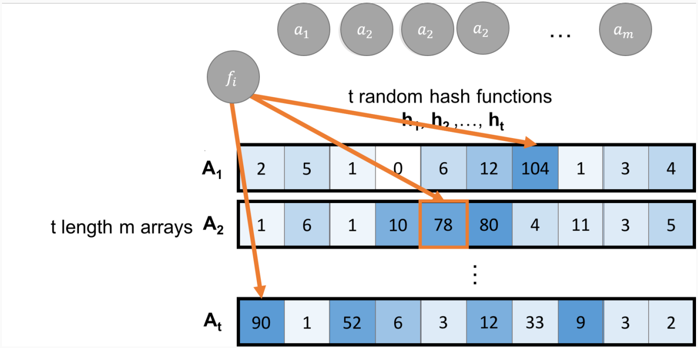
f_i: frequency of item i in the stream. h: random hash function. B: size of Count-Min sketch array.

Count-Min Sketch Accuracy



f_i: frequency of item i in the stream. h: random hash function. B: size of Count-Min sketch array.

Count-Min Sketch Accuracy



Estimate f_i with $\tilde{f}_i = \min_{j \in [t]} \mathbf{A}_j[\mathbf{h}_j(i)]$. (Count-**min** sketch)

Why min instead of mean or median?

Count-Min Sketch Accuracy

Estimate f_i with $\tilde{f}_i = \min_{j \in [t]} \mathbf{A}_j[\mathbf{h}_j(i)]$.

- For every coordinate $i \in [n]$ and Count-Min Array A_ℓ, for $\ell \in [t]$, setting $B = 8k$ we know that with prob. $\geq 1/2$:

$$f_i \leq \mathbf{A}_\ell[\mathbf{h}_\ell(i)] \leq f_i + \frac{\|f\|_1}{4k}$$

- $\Pr\left[f_i \leq \tilde{f}_i \leq f_i + \frac{\|f\|_1}{k}\right] \geq$
- To get a good estimate with probability $\geq 1 - \delta$,

set $t =$

.

Count-Min Sketch

Theorem

For any $\epsilon, \delta \in (0,1)$, for any $i \in [n]$ Count-min sketch yields an estimate $\tilde{f}_i$ of the frequency f_i satisfying:

$$f_i \leq \tilde{f}_i \leq f_i + \epsilon \|f\|_1$$

with probability $\geq 1 - \delta$, using $O\left(\log(1/\delta) \cdot \frac{1}{\epsilon}\right)$ words of space.

- Accurate enough to solve the k-Frequent elements problem – distinquish between items with frequency $\frac{n}{k}$ and those with frequency $\frac{1}{2} \cdot \frac{n}{k}$ – by setting $\epsilon = \Theta(1/k)$.

Identifying frequent items

Observation: Count-min sketch gives an accurate frequency estimate for each item in the stream, but finding heavy hitters ($f_i \geq \|f\|_1/k$) requires $\Omega(n)$ time!

Can we identify heavy hitters without having to compute $\tilde{f}_i$ for each $i \in [n]$?

Yes: Solution is known as the *Dyadic Trick**

*details to be posted in supplementary material on course website

Note on Hash Functions

Can we weaken our assumption that h is uniformly random?

Definition (Universal hash function)

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

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

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 $\{0, \ldots, p\}$ with $a \neq 0$.

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

is universal. Note we only need to store a, b! Proof in supplementary notes to be posted on website.

Note on Hash Functions

Another definition you might come across:

Definition (Pairwise independent hash function)

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

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

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