COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Fall 2020.

Lecture 2

By Next Thursday 9/3:

- Sign up for Piazza.
- Sign up for Gradescope (code on course website) and fill out the Gradescope consent poll on Piazza. Contact me via email if you don't consent to use Gradescope.

By Next Monday 8/31, 8pm:

- Complete Moodle Quiz – posted under Assignments tab on course website.

Last Class We Covered:

- Basic probability review. See course site for links to resources to refresh your probability background.
- Linearity of expectation: $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$ always.
- Linearity of variance: $\text{Var}[X+Y] = \text{Var}[X] + \text{Var}[Y]$ if X and Y are independent.

Today:

- An algorithmic application of of linearity of expectation and variance.
- Introduce Markov's inequality a fundamental concentration bound that let us prove that a random variable lies close to its expectation with good probability.
- Learn about random hash functions, which are a key tool in randomized methods for data processing. Probabilistic analysis via linearity of expectation.

$\text{Var}[X+Y] = \text{Var}[X] + \text{Var}[Y]$ when X and Y are independent.

Claim 1: (exercise) $\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ (via linearity of expectation)

Claim 2: (exercise) $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$ when X, Y are independent.

Together give:

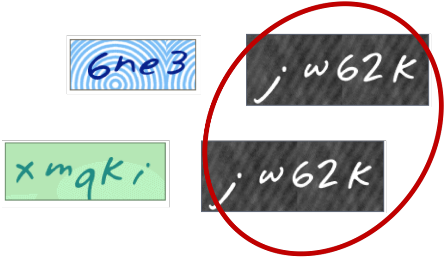
$$\begin{aligned}
\text{Var}[X+Y] &= \mathbb{E}[(X+Y)^2] - \mathbb{E}[X+Y]^2 \\
&= \mathbb{E}[X^2] + 2\mathbb{E}[XY] + \mathbb{E}[Y^2] - (\mathbb{E}[X] + \mathbb{E}[Y])^2 \\
&\qquad \text{(linearity of expectation)} \\
&= \mathbb{E}[X^2] + 2\mathbb{E}[XY] + \mathbb{E}[Y^2] - \mathbb{E}[X]^2 - 2\mathbb{E}[X] \cdot \mathbb{E}[Y] - \mathbb{E}[Y]^2 \\
&= \mathbb{E}[X^2] + \mathbb{E}[Y^2] - \mathbb{E}[X]^2 - \mathbb{E}[Y]^2 \\
&= \text{Var}[X] + \text{Var}[Y].
\end{aligned}$$

You have contracted with a new company to provide CAPTCHAS for your website.

- They claim that they have a database of $1,000,000$ unique CAPTCHAS. A random one is chosen for each security check.
- You want to independently verify this claimed database size.
- You could make test checks until you see $1,000,000$ unique CAPTCHAS: would take $\geq 1,000,000$ checks!

AN ALGORITHMIC APPLICATION

An Idea: You run some test security checks and see if any duplicate CAPTCHAS show up. If you're seeing duplicates after not too many checks, the database size is probably not too big.

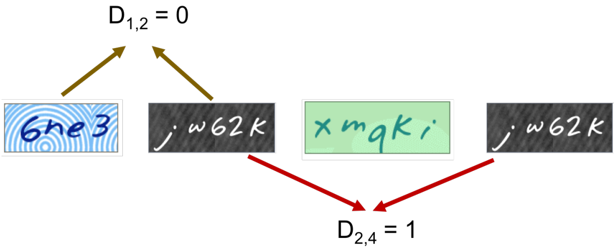


'Mark and recapture' method in ecology.

Breakout: If you run m security checks, and there are n unique CAPTCHAS, how many pairwise duplicates do you see in expectation?

If e.g. the same CAPTCHA shows up three times, on your i^{th}, j^{th}, and k^{th} test, this is three duplicates: (i, j), (i, k) and (j, k).

Let $\mathbf{D}_{i,j} = 1$ if tests i and j give the same CAPTCHA, and 0 otherwise. An indicator random variable.



The number of pairwise duplicates (a random variable) is:

$$\mathbf{D} = \sum_{i,j\in[m],i\neq j} \mathbf{D}_{i,j} . \mathbb{E}[\mathbf{D}] = \sum_{i,j\in[m],i\neq j} \mathbb{E}[\mathbf{D}_{i,j}].$$

For any pair $i, j \in [m], i \neq j$: $\quad \mathbb{E}[\mathbf{D}_{i,j}] = \Pr[\mathbf{D}_{i,j} = 1] = \frac{1}{n}$.

You take $m = 1000$ samples. If the database size is as claimed ($n = 1,000,000$) then expected number of duplicates is:

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

You see 10 pairwise duplicates and suspect that something is up. But how confident can you be in your test?

Concentration Inequalities: Bounds on the probability that a random variable deviates a certain distance from its mean.

- Useful in understanding how statistical tests perform, the behavior of randomized algorithms, the behavior of data drawn from different distributions, etc.

n: number of CAPTCHAS in database, m: number of random CAPTCHAS drawn to check database size, $\mathbf{D}$: number of pairwise duplicates in m random CAPTCHAS.

The most fundamental concentration bound: **Markov's inequality.**

For any non-negative random variable X and any $t > 0$:

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

Proof:

$$\begin{aligned}\mathbb{E}[X] = \sum_s \Pr(X = s) \cdot s &\geq \sum_{s \geq t} \Pr(X = s) \cdot s \\ &\geq \sum_{s \geq t} \Pr(X = s) \cdot t \\ &= t \cdot \Pr(X \geq t).\end{aligned}$$

The larger the deviation t, the smaller the probability.

Expected number of duplicate CAPTCHAS:

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

You see $\mathbf{D} = 10$ duplicates.

Applying Markov's inequality, if the real database size is $n = 1,000,000$ the probability of this happening is:

$$\Pr[\mathbf{D} \geq 10] \leq \frac{\mathbb{E}[\mathbf{D}]}{10} = \frac{.4995}{10} \approx .05$$

This is pretty small – you feel pretty sure the number of unique CAPTCHAS is much less than $1,000,000$. But how can you boost your confidence? We'll discuss later.

n: number of CAPTCHAS in database ($n = 1,000,000$ claimed) , m: number of random CAPTCHAS drawn to check database size ($m = 1000$ in this example), $\mathbf{D}$: number of pairwise duplicates in m random CAPTCHAS.

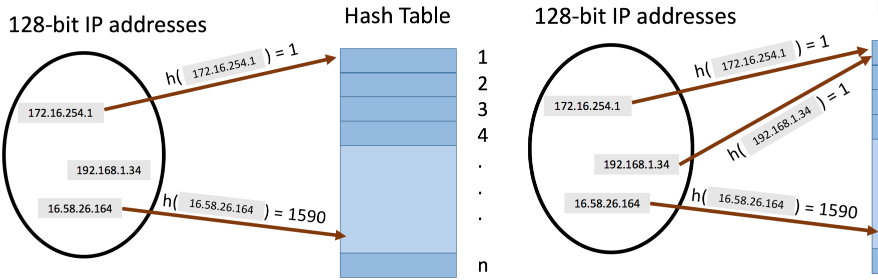
Want to store a set of items from some finite but massive universe of items (e.g., images of a certain size, text documents, 128-bit IP addresses).

Goal: support *query*(x) to check if x is in the set in $O(1)$ time.

Classic Solution: Hash tables

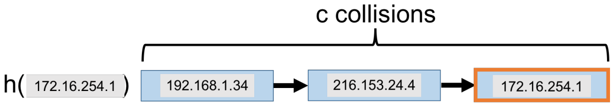
- *Static hashing* since we won't worry about insertion and deletion today.

HASH TABLES



- **hash function** $h : U \rightarrow [n]$ maps elements from the universe to indices $1, \cdots, n$ of an array.
- Typically $|U| \gg n$. Many elements map to the same index.
- **Collisions:** when we insert m items into the hash table we may have to store multiple items in the same location (typically as a linked list).

Query runtime: $O(c)$ when the maximum number of collisions in a table entry is c (i.e., must traverse a linked list of size c).



How Can We Bound c?

- In the worst case could have $c = m$ (all items hash to the same location).
- Two approaches: 1) we assume the items inserted are chosen randomly from the universe U or 2) we assume the hash function is random.

Let $\mathbf{h} : U \to [n]$ be a random hash function.

- I.e., for $x \in U$, $\Pr(\mathbf{h}(x) = i) = \frac{1}{n}$ for all $i = 1, \ldots, n$ and $\mathbf{h}(x), \mathbf{h}(y)$ are independent for any two items $x \neq y$.
- **Caveat 1:** It is *very expensive* to represent and compute such a random function. We will see how a hash function computable in $O(1)$ time function can be used instead.
- **Caveat 2:** In practice, often suffices to use hash functions like MD5, SHA-2, etc. that 'look random enough'.

Short Breakout: Assuming we insert m elements into a hash table of size n, what is the expected total number of pairwise collisions?

Let $\mathsf{C}_{i,j} = 1$ if items i and j collide ($\mathsf{h}(x_i) = \mathsf{h}(x_j)$), and 0 otherwise. The number of pairwise duplicates is:

$$\mathsf{C} = \sum_{i,j \in [m], i \neq j} \mathsf{C}_{i,j} . \mathbb{E}[\mathsf{C}] = \sum_{i,j \in [m], i \neq j} \mathbb{E}[\mathsf{C}_{i,j}].$$

(linearity of expectation)

For any pair i, j, $i \neq j$:

$\mathbb{E}[\mathsf{C}_{i,j}] = \Pr[\mathsf{C}_{i,j} = 1] = \Pr[\mathsf{h}(x_i) = \mathsf{h}(x_j)] = \frac{1}{n}$.

$$\mathbb{E}[\mathsf{C}] = \sum_{i,j \in [m], i \neq j} \frac{1}{n} = \frac{\binom{m}{2}}{n} = \frac{m(m-1)}{2n}.$$

Identical to the CAPTCHA analysis!

x_i, x_j: pair of stored items, m: total number of stored items, n: hash table size, C: total pairwise collisions in table, h: random hash function.

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

- For $n = 4m^2$ we have: $\mathbb{E}[\mathbf{C}] = \frac{m(m-1)}{8m^2} \leq \frac{1}{8}$.
- **Breakout:** Give a lower bound on the probability that we have no collisions, i.e., $\Pr[\mathbf{C} = 0]$?

Apply Markov's Inequality: $\Pr[\mathbf{C} \geq 1] \leq \frac{\mathbb{E}[\mathbf{C}]}{1} = \frac{1}{8}$.

$$\Pr[\mathbf{C} = 0] = 1 - \Pr[\mathbf{C} \geq 1] \geq 1 - \frac{1}{8} = \frac{7}{8}.$$

Pretty good...but we are using $O(m^2)$ space to store m items...

> m: total number of stored items, n: hash table size, $\mathbf{C}$: total pairwise collisions in table.