CS 677: Big Data

Data Sketches: Frequent Items

Lecture 14

Streaming Big Data

- In many cases, data is produced much faster than we can analyze it
- Batch processing systems like MapReduce let us do analysis offline, after the fact
  - Good: studying long-term trends
  - Bad: reacting quickly…
    - Health monitoring, rerouting traffic, etc.
- We can use a stream processing system, but what happens when that can't handle the workload?

Sketching

- Rather than storing/processing everything, we can build data sketches of the datasets
- Some information is thrown away…
  - …but we can store a wider breadth of information.
- These approaches have memory and processing benefits
  - Also well-suited to IoT devices, low-powered cloud instances, etc

Counting Events

- What if we have a lot of elements in a stream and want to know which happen the most frequently?
- These are "Heavy hitters" – most popular videos, websites, network users, etc.
- Tracking this in a small amount of space is a challenging problem
  - Simple but **inefficient** approach: store everything and sort it!
    - Correction: not inefficient. **Impossible** with big data.

Majority Sketch

- Imagine we are electing a new US president
  - Whoever receives the majority of votes wins
- We have a data stream of votes
  - Easy! Tally up all the votes for **Candidate A** and **Candidate B** and report the winner
- Except it's not really that simple…
  - Many US citizens will be shocked to learn that there are more than two possible presidential candidates
  - In fact, it's possible to **write in** a vote for whoever you want

Tracking Votes

- Fine. We'll just store **ALL** the votes and see who wins.
  - It's still gonna either be **Candidate A** or **Candidate B**!
- Unfortunately, to make matters worse, society has collapsed in an Idiocracy-style dystopia
  - All computing power is devoted to TikTok
- You are able to salvage the original *Apollo Guidance Computer* (AGC) from a museum, but it doesn't have a lot of memory
  - Instead of the actual vote count, can we at least find out who got over 50% of votes?

Streaming Majority Algorithm

- Enough *tomfoolery*. Let's look at this algorithm.

1. Initialize a counter to zero. (`c = 0`)
2. For each element in the stream:
   - If the counter is zero, set `majority = element`
   - If `majority == element`, increment counter (`c++`)
     - Else, decrement the counter. (`c--`)

- When our stream is finished, if `c > 0` then we have successfully determined the majority

Exceptions

- If the final count is `0`, then the only thing we know is the last element recorded is **NOT** the majority
- It could have occurred up to $\frac{n}{2}$ times... (meaning we have a tie) but we don't know for sure
  - Time for a recount!
- If we have three strong candidates, this algorithm won't help
  - We need a situation where knowing what element occurred more than 50% of the time is useful

Time Wasted?

- Maybe the previous situation doesn't seem very likely or useful
- This algorithm sometimes gets taught in undergrad courses to introduce streams
  - Might be less common now though
- But the algorithm **IS** used in many high-traffic, low computing power situations
  - (determine the destination the majority of packets are being sent to on a network switch)

A Better Majority Algorithm

- Around 2000 or so, a remixed version of this algorithm caught on
- Massive data streams can make revisiting simple, not-so-useful algorithms a bit more interesting
- Frequent Elements Sketch
  - Concerned with "top N" type queries or "iceberg queries"
  - Useful for network monitoring, log analysis, and data mining

Frequent Elements Sketch [1/2]

- Let's say we want to know the top N YouTube videos based on URL clicks
- Initialize an array with size N and store $(URL, count)$ pairs in the array:

$$[0] \rightarrow (URL_1, count)$$
$$[1] \rightarrow (URL_2, count)$$

...

$$[N] \rightarrow (URL_N, count)$$

Frequent Elements Sketch [2/2]

- Start reading items from the data stream…
- When an element (URL in our case) comes in:
  1. If it's in the sketch already, increment its count
  2. If it's not in the sketch, but there's a free space in the array, insert it in the empty slot
  3. If it's not in the sketch and there's no room for it in the array, decrement all counts
  4. If any count drops to 0, remove it from the array, which will free up a slot

Using the Sketch

- First, sort the array by the element counts
- Boom! Report the results. They are your top N elements.

Enhancements

- We can use a map / dictionary / etc. to track the elements instead
  - Easier to test for set membership, increment existing counts
- Track the total number of elements
  - Allows us to calculate the percentage each element represents of the overall dataset

Weaknesses

- This algorithm requires us to know N – how many things we want
  - That is not always possible
- It is susceptible to attacks if we cannot trust the data stream
  - Feeding it with the right (unique or random) inputs will cause frequent elements to be removed
  - The answer will be ***technically*** right, but since we threw most of the data away we will not be able to detect the attack

Where to go next

- We have only begun to see the tip of the iceberg when it comes to streaming algorithms
  - Get it?? Get it?? Iceberg queries!!
- If we are willing to throw away more data and use a bit of probability we can make some cool predictions with very little storage space…