Count Min-Sketch: The Heavy Hitters Problem
Outline

• Review Big Data Streaming Model
  • Bloom Filters

• Application: The Heavy Hitters Problem
  • (Detecting Viral Google Searches)

• Streaming Data Structure: Count Min-Sketch
Big Data

- **Problem.** Too much data to fit in memory (e.g., who can store the internet graph?)

ONE DOES NOT MERELY "STORE" BIG DATA.
Big Data

- **Problem.** Alternatively, maybe we could store our data, but it would take too long to process it, and we want a real time (or near real time) application.
Streaming Model

- **Solution.** In the **streaming model** of computation, we process the data one piece at a time, with limited memory.

- Equivalently: we develop algorithms that run in a *single* left to right pass over an array, with a small amount of auxiliary storage.

<table>
<thead>
<tr>
<th>Football</th>
<th>Duke</th>
<th>Politics</th>
<th>News</th>
<th>...</th>
<th>Weather</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>T</td>
</tr>
</tbody>
</table>

Auxiliary Storage of size $n \ll T$
Bloom Filter

• We have already seen how to construct a *bloom filter*, a form of *lossy compression* (as opposed to lossless compression, e.g., Huffman).

• Answers *membership queries*; i.e., “Have I seen element x before in the stream?”

• Applications include:
  • Web browser checking for known malicious urls
  • Checking for “one hit wonders” in web caching (remember consistent hashing?)
Bloom Filter

• Our auxiliary storage is just a hash table of size $n$. Initialize all values to 0.

• We also use $r$ independent hash functions $h_1, \ldots, h_r$.

• Whenever we see an element $x$ in the stream, set $h_1(x) = \ldots = h_r(x) = 1$.

• To check whether we have seen an element $y$:
  • If $h_1(y) = \ldots = h_r(y) = 1$, return True.
  • Else, return False.
Bloom Filter

Diagram of a Bloom Filter with five items: Football, Duke, Politics, News, and Weather. The Bloom Filter uses hash functions $h_1$, $h_2$, and $h_3$ to map items to bits in a bit array.
Bloom Filter

• Guarantees:
  • If we *have* seen $x$, we always correctly output True.
  • If we *have not* seen $x$, we correctly output False with high probability.

• What if we want to remember more than just whether we have seen $x$?

• How about “How many times have we seen $x$?”
Outline

• Review Big Data Streaming Model
  • Bloom Filters

• Application: The Heavy Hitters Problem
  • (Detecting Viral Google Searches)

• Streaming Data Structure: Count Min-Sketch
Heavy Hitters Problem

• In particular, suppose we want to construct an algorithm for detecting viral google searches.

• There are a few billion google searches every day, and we’ll say that a search is viral if it constitutes a constant fraction of those searches (e.g., 1%).

• Can we detect these viral google searches with a single pass over the stream of searches?
Heavy Hitters Problem

• We can formalize this as the heavy hitters problem.

• We are given a stream of length $T$ and a parameter $k$.
  • Think of $T >> k$.

• In a single pass over the stream, we want to find any elements that appear at least $T/k$ times.
Heavy Hitters Problem

• Bloom filters gets us part of the way there.

• In particular, if we had \( k=T \), the heavy hitters problem is the membership problem.

• Thus, the heavy hitters problem is \textit{at least} as hard (computationally, more on reductions later in the course) as the membership problem.

• Since we only had a correct algorithm with high probability for membership, we shouldn’t expect an exact answer here.
Heavy Hitters Problem

• Thus, we consider the **ε-approximate heavy hitters problem**. Still given a stream of length $T$ and a parameter $k$ ($T >> k$), but we are also given an “error tolerance” parameter $\epsilon$.

• In a single pass over the stream using just $O(1/\epsilon)$ auxiliary storage, we want to output a list $L$ of elements such that:
  - If $x$ occurs at least $T/k$ times in the stream, then $x$ is in $L$.
  - If $x$ is in $L$, then with high probability, $x$ occurs at least $T/k - \epsilon T$ times in the stream.
  - (e.g., if $\epsilon = 1/(2k)$, then we get $O(k)$ storage and should satisfy: if $x$ is in $L$, with high probability, $x$ occurs at least $T/2k$ times in the stream).
Outline

• Review Big Data Streaming Model
  • Bloom Filters

• Application: The Heavy Hitters Problem
  • (Detecting Viral Google Searches)

• Streaming Data Structure: Count Min-Sketch
Count Min-Sketch

• **Big Idea.** Just build a bloom filter that can count.

• Our auxiliary storage consists of $r$ hash tables, each of size $n$ and initialized to 0’s, with corresponding $r$ independent hash functions $h_1, \ldots, h_r$.

• Whenever we see an element $x$ in the stream:
  • For all $i=1$ to $i=r$: \( h_i(x) = h_i(x) + 1 \)
  • if $\min_i h_i(x) \geq T/k$, add $x$ to L.
Count Min-Sketch

Football → Duke → Football → News → ...

\( h_1 \) → +1  
\( h_2 \) → +1  
\( h_3 \) → +1  
\( h_4 \) → +1  
\( h_{r=5} \) → +1
Count Min-Sketch

<table>
<thead>
<tr>
<th>Football</th>
<th>Duke</th>
<th>Football</th>
<th>News</th>
<th>...</th>
<th>Weather</th>
</tr>
</thead>
</table>

- $h_1$: +1
- $h_2$: 1
- $h_3$: +1
- $h_4$: 1+1
- $h_{r=5}$: +1

$r = 5$
### Count Min-Sketch

<table>
<thead>
<tr>
<th>Football</th>
<th>Duke</th>
<th>Football</th>
<th>News</th>
<th>...</th>
<th>Weather</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>h₁</th>
<th>1</th>
<th>1+1</th>
<th></th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>h₂</td>
<td>1+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h₃</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1+1</td>
</tr>
<tr>
<td>h₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hᵣₕ=₅</td>
<td>1</td>
<td></td>
<td></td>
<td>1+1</td>
<td></td>
</tr>
</tbody>
</table>
Count Min-Sketch

• Note that we occasionally overestimate frequencies, but we never underestimate frequencies.

• So it is easy to satisfy the first part of the heavy hitter’s problem: “If $x$ occurs at least $T/k$ times in the stream, then $x$ is in $L$.”

• **Problem.** We need to argue that it is unlikely we overestimate so badly that we violate the other part: “If $x$ is in $L$, then with high probability, $x$ occurs at least $T/k - \epsilon T$ times in the stream.”
Count Min-Sketch

• Let $f_x$ be the frequency (#{ of times appearing in stream}) of element $x$.

• Let $\hat{f}_x[1], ..., \hat{f}_x[r]$ be our estimated frequencies, that is, $\hat{f}_x[i] = h_i(x)$ at the end of our pass through the stream.

• Let $I_{x,y}[i]$ be an indicator random variable equal to 1 if $h_i(x) = h_i(y)$, and 0 otherwise.

• What is $\mathbb{E} \left[ \hat{f}_x[i] \right]$?
Count Min-Sketch

- We make the assumption of *universal hashing*: For all \( x \neq y \), \( \Pr(h(x) = h(y)) \leq \frac{1}{n} \).

\[
\mathbb{E} \left[ \hat{f}_x[i] \right] = f_x + \mathbb{E} \left[ \sum_{y \neq x} f_y \times I_{x,y}[i] \right]
\]

\[
= f_x + \sum_{y \neq x} f_y \mathbb{E} \left[ I_{x,y}[i] \right]
\]

\[
= f_x + \sum_{y \neq x} \frac{f_y}{n} \leq f_x + \frac{T}{n}
\]
Count Min-Sketch

• Recall we want to use $O(1/\varepsilon)$ storage: set $n$ (size of each hash table) to $3/\varepsilon$. Let $\varepsilon = 1/(2k)$. Then

$$
\mathbb{E} \left[ \hat{f}_x[i] \right] \leq f_x + \varepsilon \frac{T}{3} = f_x + \frac{T}{6k}.
$$

• To bound the probability that we get a large overestimate, we can use Markov’s inequality: For any constant $c > 1$ and random variable $X$, $\Pr(X > c \mathbb{E}[X]) \leq \frac{1}{c}$. For $c = 3/2$,

$$
\Pr \left( \hat{f}_x[i] > \frac{3}{2} \mathbb{E} \left[ \hat{f}_x[i] \right] \right) = \frac{3}{2} f_x + \frac{T}{4k} \leq \frac{2}{3}.
$$
Count Min-Sketch

• Recall however, that we output the minimum estimate. Exploiting the fact that the $r$ hash functions are chosen independently:

$$\Pr \left( \min_i \hat{f}_x[i] > \frac{3}{2} \mathbb{E} [\hat{f}_x[i]] \right) = \Pr \left( \forall i, \hat{f}_x[i] > \frac{3}{2} \mathbb{E} [\hat{f}_x[i]] \right)$$

$$= \prod_i \Pr \left( \hat{f}_x[i] > \frac{3}{2} \mathbb{E} [\hat{f}_x[i]] \right)$$

$$\leq \left( \frac{2}{3} \right)^r$$
Count Min-Sketch

• Recall that the problem for $\epsilon = 1/2k$ is: we get $O(k)$ storage and should satisfy: if $x$ is in $L$, with high probability, $x$ occurs at least $T/(2k)$ times in the stream).

• Consider some $x$ with $f_x < \frac{T}{2k}$. We have shown that

$$\Pr \left( \min_i \hat{f}_x[i] > \frac{3T}{4k} + \frac{T}{4k} = \frac{T}{k} \right) \leq \left( \frac{2}{3} \right)^r.$$  

• So if $x$ is in $L$, then it occurs at least $T/(2k)$ times in the stream with probability at least $1-(2/3)^r$.

• So if we want an error with probability at most 2% (say), we just need to use $r = \left\lceil \log_{3/2}(50) \right\rceil = 10$ independent hash functions.
Count Min-Sketch

• In summary, we can use $20/\varepsilon = O(1/\varepsilon)$ space to:
  • find all elements that appear at least $T/k$ times in the stream, and
  • output elements that appear less than $T/2k$ times in stream with probability at most 2%.
  • And in practice, even fewer hash functions often suffice for good performance.

• Note that we can do all of this with just a single linear scan over the stream (and only constant time operations per element), and just $O(1/\varepsilon)$ storage.
  • The amount of auxiliary storage we use is completely independent of $T$!
Count Min-Sketch

• **Food for Thought.** What if you didn’t know T beforehand?
  • Maybe this is just a real time application, and you want to maintain a list of any elements that are heavy hitters among what you have seen so far.