COMS 4995-8: Advanced Algorithms (Spring'21) Jan 26, 2021

Lecture 5: Heavy Hitters

Instructor: *Alex Andoni* Scribes: *Joseph Yang (zy2431), Hanna Kartynnik (hk3129)*

1 Announcements

- A new TA for the course, office hour will be posted later on courseworks.
- Homework 1 will be due on Thursday, right before class.

2 Recap

Recall from last time, we talked about **streaming** and **sketching** algorithms. The basic model of a streaming problem is: we have a stream of objects $x_1, x_2, \ldots, x_m$, and for example they come from some universe of numbers from 1 to n. Namely, $x_1, x_2, \ldots, x_m \in [n]$.

Goal: estimate the number of distinct objects in the stream.
Algorithm: Bottom-k algorithm, using a hash function $h : [n] \to [0,1]$. It is enough to set $k = \Theta(1/\epsilon^2)$ for a $(1 \pm \epsilon)$-approximation, supposing $d \gg 1/\epsilon^2$. Here, recall from last lecture, d is the number of unique objects in the stream.

Example 1. *n is the number of different IP ($n = 2^{32}$).*

Additional Remarks

1. Since the output of the hash function is any real number on the $[0,1]$ interval, the number of buckets of our hash function is uncountable (one bucket for one real number), which is impossible to store. So we need to modify the hash function to $h : [n] \to \{0, 1, \ldots, M-1\}$ or $h : [n] \to \{0, \frac{1}{M}, \frac{2}{M} \ldots, \frac{M-1}{M}\}$ for some M large enough.
   The only tiny issue is: when we use real number outputs, the probability that $h(x) = h(y)$ for random x and y is actually 0. But if we switch to discrete values, the probability of collision is nonzero now. Suppose there are n different items in the stream, if we pick $M = n^3$, then $\mathbb{E}[\#\text{collisions}] \leq \frac{n^2}{M} = \frac{1}{n}$. So if we have M sufficiently large, the number of collisions would be very few, in expectation.

2. In the Bottom-k algorithm, the hash function has to be fully random. In fact, it is possible to use hash functions with "limited randomness" to achieve what we want.

   Definition 2. *Let H be a family of hash functions. H is k-wise independent if for any $a_1, a_2, \ldots, a_k \in \{0, \frac{1}{M}, \ldots, \frac{M-1}{M}\}$, and for any $x_1, \ldots, x_k \in [n]$ (all x_i's are distinct), and any hash function $h \in H$, we have $Pr[h(x_i) = a_i, \forall i = 1, \ldots, k] = \frac{1}{M^k}$.*

Remark: if H is 2-wise independent, then it's universal.
Remark: $\forall k \geq 2$, we can construct k-wise independent hash function class H of size $\log|H| \leq O(k \cdot \log n)$.

Theorem 3 (Thorup '13)**.** *For the Bottom-k algorithm, it's enough to use 2-wise independent hash function.*

The idea behind k-wise independent is: whenever we consider probability, we consider the probability over at most k objects at the same time. To better illustrate what this means: when we compute expectation, we often take one random variable; when we compute something like variance, we take two random variables. In the bottom-k algorithm, we consider k variables at the same time.

3 Most Frequent Object Problem

Suppose we are still working with a stream of objects, namely $x_1, x_2, \ldots, x_m \in [n]$. What is the most frequent object?

Definition 4. *We define the* ***frequency vector*** $f \in \mathbb{R}_+^n$ *as the following: for any dimension* f_x *of the vector* f*, it represents how many times the object* x *has appeared in the stream.*

One quick observation would be $||f||_1 = \sum_x f_x = m =$ length of the stream. Here, $||f||_1$ is simply the 1-norm of the vector f, since $f_x \in \mathbb{R}_+$ (i.e. all $f_x \geq 0$).

Problem Revision: we would like to find x where f_x is maximal among all dimensions of the frequency vector f.

Simple Solution: Simply store the vector f explicitly. This needs $O(n)$ space, which is too large.

Theorem

- We can't store the exact frequency vector f with less space.
- We can't use 2-approximation for this problem, even with randomization.

  Here's a bad case: all items appear once, except exactly 1 item appears twice. In order to detect whether an item is a new item or an item we have seen, we must know every item that has appeared so far. So 2-approximation is ineffective for this problem.

Relaxed Version of the Problem: report items which are **sufficiently** frequent.

Definition 5. *Suppose* $\phi \in (0,1)$*, item* $x \in [n]$ *is a* ϕ*-heavy hitter if* $f_x \geq \phi \cdot \sum_y f_y = \phi \cdot m$*. In other words, the frequency of item* x *is at least* ϕ *fraction of the entire length of the stream.*

New Goal: find all items such that are ϕ-HH (heavy hitters). Acceptable space complexity depends on ϕ.

4 The CountMin Algorithm

Basic Idea: to get some estimate $\hat{f}_x$ for f_x.

In order to reduce space needed to store the vector, we need to reduce the number of dimensions in the vector. In other words, we need to "transform" the original frequency vector f which has n dimensions, to a vector S which has w dimensions, such that $w < n$. There could be collisions, since we are reducing from something bigger to something smaller, but we'll take that as trade-off of less space needed.

An intuitive idea would be: we use a hash function $h : [n] \to [w]$ to try to store the information of f in S.

Suppose $f_{i_1}, f_{i_2}, f_{i_3}$ are the dimensions in f that were assigned to the same bucket i by the hash function, then we would like to store the sum of $f_{i_1}, f_{i_2}, f_{i_3}$ in the i^{th} dimension of vector S, namely S_i. This is because we want the value stored in S_i also change whenever any one of $f_{i_1}, f_{i_2}, f_{i_3}$ changes.

Algorithm 1: Bucket-Algorithm

```
Result: approximated count of how many times object $x$ appears in the stream (for each $x \in [n]$)
Initial: Set $S[1] = S[2] = \cdots = S[w] = 0$, in other words, $S$ is just the zero vector with $w$
  dimensions;
for each time we see an object $x$ in the stream do
    $S[h(x)] \leftarrow S[h(x)] + 1$
end
return estimators $\hat{f}_x = S[h(x)]$ (for each $x \in [n]$)
```

We want to prove that this estimator $\hat{f}_x$ is good.

Observation 6. *It is easy to see that $\hat{f}_x \geq f_x$. Our estimator takes into account the occurrences of x and other elements for which their hash value is equal to $h(x)$. So the estimator must be at least f_x.*

The estimator $\hat{f}_x$ can be much larger than the actual frequency f_x if many other items also fall into the same bucket as x. Let us show that the estimator is not too large.

Claim 7. $\Pr_h[\hat{f}_x > f_x + \epsilon\phi m] \leq 0.1$ *as long as* $w \geq \Omega(\frac{1}{\epsilon\phi})$.

Intuition. We care about ϕ-heavy hitters, i.e about frequencies f_x which are at least ϕm. Intuitively, the approximation comes from the fact that we will never be able to be precisely sure whether the item's frequency is slightly more than this threshold ϕm or slightly less. Generally speaking, we need to be able to detect f_x whose magnitude is about ϕm. Note that $\epsilon\phi m$ is an additive error.

Proof. We assume that h is a fully random hash function. Let us find $\mathbb{E}_h[\hat{f}_x]$:

$$\begin{aligned}
\mathbb{E}_h[\hat{f}_x] &= \mathbb{E}_h\left[\sum_{y\in[n]} \mathbb{1}[h(x)=h(y)]\cdot f_y\right] \\
&= \mathbb{E}_h\left[f_x + \sum_{y\neq x} \mathbb{1}[h(x)=h(y)]\cdot f_y\right] \\
&= f_x + \sum_{y\neq x} f_y \underbrace{\Pr[h(x)=h(y)]}_{\frac{1}{w}} \\
&\leq f_x + \frac{\sum_y f_y}{w} \\
&= f_x + \frac{m}{w}
\end{aligned}$$

We can rewrite it as $\mathbb{E}[\hat{f}_x - f_x] \leq m/w$. $\hat{f}_x - f_x$ is a positive random variable and we can use Markov's inequality:

$$\Pr\left[\hat{f}_x - f_x > 10\cdot\frac{m}{w}\right] \leq 0.1.$$

In order to get the result from the claim statement, we need for $10m/w$ to be equal to $\epsilon\phi m$, which we can achieve by setting $w = \frac{10}{\epsilon\phi}$. ☐

Algorithm 2: The ϕ-heavy hitter algorithm

```
Result: all $\phi$-heavy hitters in the stream
for each $x \in [n]$ do
    Compute $\hat{f}_x$ from $S$
    if $\hat{f}_x \geq \phi m$ then
        Report $x$ as $\phi$-HH
end
```

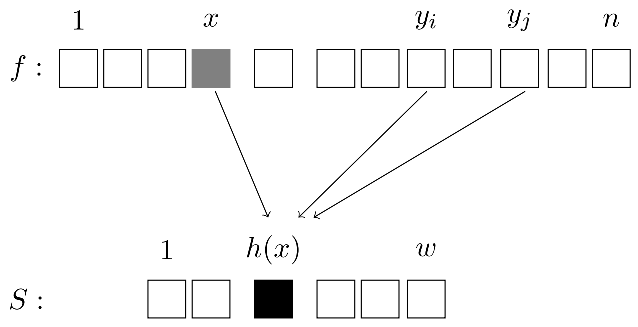
Corollary 8.

- *If $f_x \geq \phi m$, the algorithm reports x as a ϕ-heavy hitter ($f_x \geq \phi m$ implies $\hat{f}_x \geq \phi m$).*
- *If $f_x < \phi m - \epsilon\phi m = (1-\epsilon)\phi m$, the algorithm doesn't report x as a ϕ-heavy hitter with at least 90% probability.*

By Claim **??**, $\hat{f}_x$ is less than $f_x + \epsilon\phi m$ with greater than 90% probability. Hence, if $f_x < \phi m - \epsilon\phi m$,

$$\Pr[\hat{f}_x < f_x + \epsilon\phi m < \phi m] \geq 0.9.$$

The issue of this algorithm: with 10% probability some non-HH x's are reported as HH's. It is not just theoretical issue, it will actually happen. We have a very large frequency vector f and we map a large number of n items into a relatively small vector S of size $w \ll n$. Say we have a frequent item x that falls into a bucket $h(x)$. In expectation, a fraction of $1/w$ of all other items in the stream will also fall into the bucket $h(x)$, so we cannot distinguish which one contributed to the "heaviness" of this bucket.



In fact, all y's such that $h(y) = h(x)$ will be considered heavy and that's an issue because there are a lot of items. How can we solve such an issue? We can make sure there are no collisions but then we need w to be $\Omega(n)$ which does not save the space. That's why we will do something different.

4.1 Full CountMin algorithm

The idea is to repeat the "basic idea" (the Bucket algorithm) $L = O(\log n)$ times.

```
Algorithm 3: Full CountMin algorithm
  Result: approximated count of how many times $x$ appears in the stream (for each $x \in [n]$)
  Initialize $S_i[1..w]$, $h_i : [n] \to [w]$, $\forall i = 1..L$
  for each time we see an object $x$ in the stream do
  |   $S_i[h_i(x)] \leftarrow S_i[h_i(x)] + 1$, $\forall i = 1..L$
  end
  return estimators $\hat{f}_x \leftarrow \min_{1 \le i \le L} S_i[h_i(x)]$ (for each $x \in [n]$)
```

The reason we return the minimum is that all the estimators are biased towards one side since they return frequencies larger than the actual frequencies. Thus, the minimum estimator for $x \in [n]$ is closest to the true value. Let us analyze the algorithm.

Claim 9. *For $w = \frac{10}{\epsilon\phi}$, $L = O(\log n)$, we have*

$$\hat{f}_x - f_x \le \epsilon\phi m$$

for all $x \in [n]$ with probability at least $1 - 1/n$.

Proof. By Claim **??**, for any fixed $i \in [L]$,

$$\Pr_{h_i}\left[S_i[h_i(x)] - f_x > \epsilon\phi m\right] < \frac{1}{10}.$$

For a fixed $x \in [n]$,

$$\begin{aligned}\Pr_{h_1,\dots,h_L}[\hat{f}_x - f_x > \epsilon\phi m] &= \prod_{i=1}^{L} \Pr_{h_i}[S_i[h_i(x)] - f_x > \epsilon\phi m] \\ &\leq \left(\frac{1}{10}\right)^L = e^{-L\ln 10} = \frac{1}{n^2},\end{aligned}$$

as long as $L = \frac{2\ln n}{\ln 10} = O(\log n)$. (The first equality holds since the hash functions are independent and each $S_i[h_i(x)]$ is greater or equal to their common lower bound $\hat{f}_x$.) Hence

$$\Pr[\exists x \text{ s.t. } \hat{f}_x - f_x > \epsilon\phi m] \underset{\text{by union bound}}{\leq} \sum_{x\in[n]} \Pr[\hat{f}_x - f_x > \epsilon\phi m] \leq n \cdot \frac{1}{n^2} = \frac{1}{n}.$$

□

We conclude that the fraction of items incorrectly reported as heavy hitters drops to $1/n$.

The **space** needed for Full CountMin algorithm is $O(Lw)$ words (that count up to m) which implies $O(\frac{\log n}{\epsilon\phi})$ words for ϕ-HH (up to $1 \pm \epsilon$ approximation, see Corollary **??**).

Observation 10 (change of the model)**.** *Consider a situation where we have a network with several routers f^1, f^2, f^3. Suppose we have a separate stream for each router. Assume we are interested in the entire frequency vector f for the network, not frequency vectors per router. Note that $f = f^1 + f^2 + f^3$ which implies $f_x = f_x^1 + f_x^2 + f_x^3$ for any $x \in [n]$. How can we compute a sketch that is able to estimate $\hat{f}_x$ for all $x \in [n]$?*

Solution. Each router computes a CountMin sketch of its frequency vector (using the same hash functions) and then we sum them all up. In particular, a CM sketch of some x is a vector from $\mathbb{R}^{Lw}$. Each coordinate in S_i is a linear combination of the coordinates of the frequency vector f. The CM sketch can be written in the form $A \cdot f$ where A is a matrix of size $Lw \times n$ ($A_{(i,j),x} = 1$ iff $h_i(x) = j$, $i \in [L]$, $j \in [w]$, $x \in [n]$). This is another way to write the sketch function. So the sketch can be described as a linear operation to be applied to the frequency vector.

$$\underbrace{Af^1}_{\text{sketch at router 1}} + Af^2 + Af^3 = \underbrace{A(f^1 + f^2 + f^3)}_{\text{sketch of frequency vector } f^1+f^2+f^3}.$$

We conclude that CountMin is a **linear** sketch.

Questions for the next lecture: what are the potential issues of the ϕ-HH algorithm? How can they be fixed?