CS 761 : Randomized Algorithms, Spring 2018, Waterloo

Lecture 7 : Data Streaming

We study some interesting sublinear space algorithms, including detecting heavy hitters, estimating number of distinct elements, and estimating frequency moments.

We will see that ideas from hashing and k-wise independence are very useful in this setting.

Sublinear algorithms In some applications where we have a massive data set, the data is too much that we can't afford to store them all or read them once. Yet we can design sublinear space or sublinear time algorithms to do something nontrivial. Randomness is crucial in these settings, as it can be shown that deterministic algorithms cannot guarantee anything nontrivial for most problems.

For sublinear space algorithms, we focus on the data streaming model, in which the algorithm can take one pass on the data but has very limited space for computation. A good motivating example is in computer networks, where routers would like to get good statistics of the traffic but could not afford to store all the data. We will study some basic problems such as estimating the number of distinct elements, the second moment of the data, etc.

Since we use so little space, we could not hope to solve the problem exactly, and we work with a weaker requirement that the algorithm can distinguish between YES-instances and instances that are "far from YES" (e.g. # distinct elements $\geq D$, or # distinct elements $\ll D$).

Of course, we also allow some error probability δ; we will make them precise in the following

---

Heavy Hitters

Given a data stream $X_1, X_2, X_3, \ldots, X_T$, each $X_t = (i_t, c_t)$ where i_t is the ID of the t-th data and c_t is the weight associated with it, e.g. i_t is the IP address and c_t is the number of bytes of the data transmitted.

Our goal is to find all the <u>heavy hitters</u> of the data stream. Let $Q = \sum_t c_t$ be the total count. Given a number f, we say an ID i is a heavy hitter if $\sum_{t: i_t = i} c_t \geq f$. (e.g. $f \geq Q/100$)

Let $\text{Count}(i, T) = \sum_{t=1, i_t=i}^{T} c_i$ be the total count for ID i.

We will show a sublinear space algorithm with the following properties:

① All heavy hitters are reported, i.e. those i with $\text{Count}(i,T) \geq \phi$,

② If $\text{Count}(i,T) \leq \phi - \varepsilon Q$, then i is reported with probability at most δ.

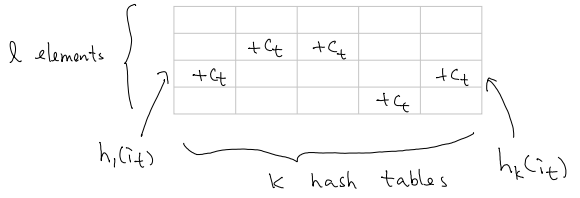
Note that the performance guarantee is nontrivial only when $\phi = \Omega(Q)$. In that range, the algorithm will report all heavy hitters, only having a small probability reporting an ID that is far from a heavy hitter.

The idea is to use 2-universal hash functions.

We will use k hash functions, $h_1, h_2, \ldots, h_k$, each maps the universe into $\{1, 2, \ldots, \ell\}$.

We maintain $k \cdot \ell$ counters, each counter $C_{a,j}$ adds the weight of the items that are mapped to the j-th entry by the a-th hash function. Initially $C_{a,j} = 0$ for $1 \leq a \leq k$, $1 \leq j \leq \ell$.

The algorithm is simple. Given $X_t = (i_t, c_t)$, for each $1 \leq a \leq k$, increment $C_{a,h_a(i_t)}$ by c_t.



when (i_t, c_t) comes
compute $h_1(i_t), h_2(i_t), \ldots, h_k(i_t)$
increment those entries by c_t

After we read all the data, we report all i with $\min_{j=h_a(i),\, 1 \leq a \leq k} C_{a,j} \geq \phi$. In words, we report an ID if the minimum counter associated with it is at least ϕ.

Analysis

It should be clear that a heavy hitter is always reported, since by our algorithm all the counters associated with it must be at least ϕ.

The key is to show that if an ID is not a heavy hitter, then it is reported with small probability.

Let us consider a specific ID i. Suppose $\text{Count}(i,T) \leq \phi - \varepsilon Q$. What is the prob that i is reported?

Consider $C_{a,h_a(i)}$ for a specific a. Since i is reported, we must have $C_{a,H_a(i)} \geq \phi$.

Let Z_a be the value of this counter that is incremented by other IDs.

Since h_a is chosen from a 2-universal family, another item is mapped to $H_a(i)$ with probability $\leq \frac{1}{\ell}$

Therefore $E[Z_a] \leq Q/\ell$.

By Markov's inequality, $\Pr(Z_a \geq \varepsilon Q) \leq \frac{1}{\varepsilon \ell}$.

So $Pr(\min Z_a \geq \varepsilon Q) \leq (\frac{1}{e^{\ell}})^k$

Set $\ell = \frac{e}{\varepsilon}$ and $k = \ln(\frac{1}{\delta})$, this probability is at most δ.

So, we just need to maintain $O(\frac{1}{\varepsilon}\ln\frac{1}{\delta})$ counters for this task, which is just a constant for constant ε and δ.

The hash functions can be stored efficiently, using $O(k) = O(\ln\frac{1}{\delta})$ calls.

And the evaluation time is $O(k) = O(\ln\frac{1}{\delta})$ word operations.

Distinct Elements

The input is a stream of a sequence $a_1, a_2, \ldots, a_n$, where each a_i is chosen from $\{1, 2, \ldots, m\}$.

Our algorithm is allowed to take one pass of the stream, using very limited space (e.g. $O(\log(m+n))$), and return a good estimate on the number of distinct elements in the data stream.

We will give bounds in terms of $\log(m)$ instead of $\log(n)$. It is good in the case when $n >> m$.

In the case when $m > n$, we can first hash each item to a table of size n^2 or larger, then with high probability there will be no collisions, and thus we can assume $\log m = O(\log n)$.

We show how to use a 2-universal hash family to solve the problem.

The idea is to map the universe $\{1, 2, \ldots, m\}$ to $\{1, 2, \ldots, m^3\}$

As we have seen last time, there will be no collisions with high probability.

Ideally, we want the hash values $h(1), h(2), \ldots, h(m)$ to be evenly distributed in $\{1, 2, \ldots, m^3\}$.

Then, if there are D distinct elements, the t-th smallest hash value should be close to $t \cdot \frac{m^3}{D}$.

Therefore, if we know the t-th smallest non-empty bin is T, then $T \approx t \cdot \frac{m^3}{D}$ and thus $D \approx \frac{t \cdot m^3}{T}$.

We must be careful about the space requirement. First of all, we are not going to store the whole hash table, as it is really unnecessary.

We only need to store t numbers, corresponding to the t smallest non-empty bins.

Storing more numbers will help get a better estimate, but of course it requires more space.

We will find a good t to balance both the error probability and the space requirement.

Let's state the algorithm again.

- Pick a random hash function h from a 2-universal family.
- For each item a_i in the data stream, compute $h(a_i)$. Maintain the list of the t smallest hash values seen so far.
- Let T be the t-th smallest hash value after all the data read. Return $Y = \frac{tm^3}{T}$

Theorem By setting $t = O(\frac{1}{\varepsilon^2})$, we have $(1-\varepsilon)D \le Y \le (1+\varepsilon)D$ with constant probability.

Case 1: We first bound the probability that $Y > (1+\varepsilon)D$.

This implies that $T < \frac{tm^3}{(1+\varepsilon)D}$, which is at most $\frac{tm^3(1-\frac{\varepsilon}{2})}{D}$ for small enough ε.

This means that there are at least t hash values that are smaller than $\frac{tm^3(1-\frac{\varepsilon}{2})}{D}$.

On the other hand, let X_i be the indicator variable that $h(a_i) \le \frac{tm^3(1-\frac{\varepsilon}{2})}{D}$.

Then $E[X_i] = Pr\left(h(a_i) \le \frac{tm^3(1-\frac{\varepsilon}{2})}{D}\right) \le \frac{t(1-\frac{\varepsilon}{2})}{D}$ since $h(a_i)$ is equally likely to be in $\{1, \dots, m^3\}$.

Therefore, if there are D distinct elements, $E[\#\text{ of elements with hash values} \le \frac{tm^3(1-\frac{\varepsilon}{2})}{D}] \le t(1-\frac{\varepsilon}{2})$.

So, the expected number of such hash values is $\le t(1-\frac{\varepsilon}{2})$, but we have $\ge t$ now.

To bound the derivation, we compute the variance. Let $X = \sum_{i=1}^{D} X_i$. Then $E[X] \le t(1-\frac{\varepsilon}{2})$.

Since X_i are pairwise independent, $Var[X] = \sum_{i=1}^{D} Var[X_i] \le t(1-\frac{\varepsilon}{2})$.

By Chebyshev's inequality, $Pr(X > t) = Pr\left(X > t(1-\frac{\varepsilon}{2}) + \frac{\varepsilon}{2}\cdot t\right) \le Pr\left(|X - E[X]| \ge \frac{\varepsilon t}{2}\right)$

$$\le \frac{4\,Var[X]}{\varepsilon^2 t^2} \le \frac{4(1-\frac{\varepsilon}{2})}{\varepsilon^2 t} \le \frac{4}{\varepsilon^2 t} \le \frac{1}{6} \quad \text{for } t = O(\tfrac{1}{\varepsilon^2}).$$

Case 2: We bound the probability that $Y < (1-\varepsilon)D$. This is very similar to case 1.

The condition implies that there are less than t distinct elements with hash values $\le \frac{(1+\varepsilon)tm^3}{D}$

By a very similar calculation, $E[X] \ge (1+\frac{\varepsilon}{2})t$ and $Var[X] \le (1+\frac{\varepsilon}{2})t$.

By Chebyshev's inequality, $Pr(X < t) \le Pr\left(|X - E[X]| \ge \frac{t\varepsilon}{2}\right) \le \frac{4\,Var[X]}{\varepsilon^2 t^2} \le \frac{1}{6}$ for $t = O(\frac{1}{\varepsilon^2})$. □

Therefore, with probability at most $1/3$, $(1-\varepsilon)D \le Y \le (1+\varepsilon)D$.

By running $O(\log\frac{1}{\delta})$ copies and taking the median, the error probability is at most δ.

The total space used is $O(\frac{1}{\varepsilon^2}\log m \log(\frac{1}{\delta}))$, since each copy stores $O(\frac{1}{\varepsilon^2})$ numbers each of $O(\log m)$ bits, and the hash function only requires $O(\log m)$ bits to store.

Note that the list can be implemented by balanced binary search trees or heaps to support fast update. Since there are at most $O(\frac{1}{\varepsilon^2})$ elements, each operation can be supported in $O(\log \frac{1}{\varepsilon})$ steps.

Recently there is an optimal algorithm that uses only $O(\frac{1}{\varepsilon^2} + \log m)$ space and $O(1)$ update time. This is best possible; see the paper "An optimal algorithm for the distinct element problem", by Kane, Nelson, and Woodruff.

Frequency Moments

Again, let $a_1, a_2, \ldots, a_n$ be the items in the data stream, where each a_i is from $\{1, 2, \ldots, m\}$.

For $1 \le i \le m$, let x_i be the number of items equal to i.

We would like to estimate $\sum_{i=1}^{m} x_i^p$ where p is a given constant.

Note that when $p=0$ this is just the distinct element problem, when $p=1$ it is trivial.

Here we focus on the problem when $p=2$, and later mention results for other p.

The algorithm is very simple and elegant. This is an important result in data streaming algorithms.

- Set $r_1, r_2, \ldots, r_m$ independently to be $+1$ with prob $1/2$ and -1 with prob $1/2$.
- Maintain $Z = \sum_{i=1}^{m} r_i x_i$ (this can be done by just storing the current value of Z)
- Return $Y = Z^2$

We will prove that Y is a good approximation to $\sum_{i=1}^{m} x_i^2$ with high probability.

First we show that $E[Y] = \sum_{i=1}^{m} x_i^2$.

Since $Y = Z^2 = \left(\sum_{i=1}^{m} r_i x_i\right)^2$, $E[Y] = E[Z^2] = \sum_{1 \le i,j \le m} E[r_i x_i r_j x_j] = \sum_{1 \le i,j \le m} x_i x_j E[r_i r_j]$

When $i=j$, $E[r_i r_j] = 1$; when $i \ne j$, $E[r_i r_j] = E[r_i]E[r_j] = 0$.

Therefore, $E[Y] = E[Z^2] = \sum_{i=1}^{m} x_i^2$.

To show that it is a good approximation with high probability, we need to show that Z^2 is concentrated around its expected value, for which we'll use the Chebyshev's inequality.

So we compute $E[Y^2] = E[Z^4] = \sum_{1 \le i,j,k,\ell \le m} x_i x_j x_k x_\ell E[r_i r_j r_k r_\ell]$.

Note that $E[r_i r_j r_k r_\ell] = 0$ whenever one index appears only once.

Therefore there are only two situations when $E[r_i r_j r_k r_\ell] \ne 0$:

- when $i=j=k=l$, and this contributes $\sum_{i=1}^{m} x_i^4$ to $E[Y^2]$.

- when two indexes appear twice, and this contributes $6\sum_{i<j} x_i^2 x_j^2$ to $E[Y^2]$.

So $E[Y^2] = E[Z^4] = \sum_{i=1}^{m} x_i^4 + 6\sum_{i<j} x_i^2 x_j^2$.

Thus $Var[Y] = E[Y^2] - (E[Y])^2 = E[Z^4] - (E[Z^2])^2$

$$= \sum_{i=1}^{m} x_i^4 + 6\sum_{i<j} x_i^2 x_j^2 - \left(\sum_{i=1}^{m} x_i^2\right)^2$$

$$= \sum_{i=1}^{m} x_i^4 + 6\sum_{i<j} x_i^2 x_j^2 - \left(\sum_{i=1}^{m} x_i^4 + 2\sum_{i<j} x_i^2 x_j^2\right)$$

$$= 4\sum_{i<j} x_i^2 x_j^2 \le 2\left(\sum_{i=1}^{m} x_i^2\right)^2 = 2(E[Y])^2$$

By Chebyshev's inequality, $Pr(|Y - E[Y]| \ge c\sqrt{Var[Y]}) \le \frac{1}{c^2}$.

This already implies that $Pr(|Y - E[Y]| \ge c\sqrt{2}\,E[Y]) \le \frac{1}{c^2}$, getting a constant factor approximation.

To get a $(1\pm\varepsilon)$-approximation, we need to find a variable Y' with $E[Y] = E[Y']$ but the variance of Y' is smaller. The standard way to do this is to compute $Y_1, Y_2, \ldots, Y_k$ and take the average.

Let $Y' = \left(\sum_{i=1}^{k} Y_i\right)/k$. Then $E[Y'] = E[Y]$ and $Var[Y'] = \sum_{i=1}^{k} Var[Y_i/k] = \frac{1}{k} Var[Y] \le \frac{2}{k}(E[Y])^2$

Therefore, $Pr(|Y' - E[Y']| \ge c\sqrt{2/k}\,E[Y']) \le 1/c^2$.

Setting c to be a constant and $k = O(1/\varepsilon^2)$ will get an $(1\pm\varepsilon)$-approximation with constant probability.

The space required to compute one Y is to store one number Z. So the total space required is $O(1/\varepsilon^2)$ numbers.

Wait, how do we compute Z if we don't store $r_1, r_2, \ldots, r_m$? This requires m bits, although it is not as bad as storing all the data, it is still undesirable if the universe is large (e.g. IP address).

A key point about this approach is that we only need the variables to be 4-wise independent, so that the analysis about $E[r_i r_j r_k r_\ell]$ would hold.

Since m 4-wise independent bits can be generated from $O(\log m)$ random bits, the extra storage required for the computation is only $O(\log m)$.

This approach is called sketching, which is very useful in data streaming algorithms. It is closely related to the idea of dimension reduction.

What about $\sum_{i=1}^{m} x_i^3$? It turns out that $\Theta(n^{1-\frac{2}{p}}\,\text{polylog}(n))$ space is required to estimate $\sum_{i=1}^{m} x_i^p$ for any $p > 2$. On the other hand, for $0 \le p \le 2$ (p can be fractional), then $O(\text{polylog}(m))$ space is enough.

References Heavy hitter is from Chapter 13 of "Probability and Computing".

Distinct elements and frequency moments are from course notes of 6.895 in MIT by Piotr Indyk.

In the same course page, you can find other interesting sublinear algorithms such as Compressed sensing.