CS 466/666 Algorithm Design and Analysis . Spring 2020

Lecture 6 : 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 these problems.

Sublinear algorithms

In some applications where we have a massive data set, the data is too much that we cannot afford to store them all or read them once.

Yet we can design sublinear space or sublinear time algorithms to achieve something non-trivial.

Randomness is crucial in these settings, as it can be proved that deterministic algorithms cannot guarantee anything nontrivial for these 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.

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 threshold q (eg $q = Q/100$), we say an ID i is a heavy hitter if $\sum_{t: i_t = i} c_t \geq q$.

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

Informally, the algorithm should report all heavy hitters, and having a small probability of returning an ID that is far from being a heavy hitter.

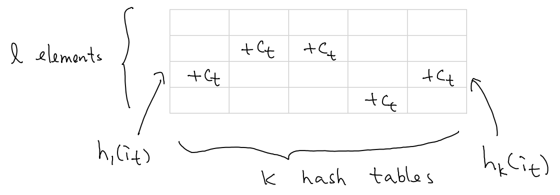
The idea is to use 2-universal hash functions.

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 $1 \le a \le k$, $1 \le j \le \ell$.

Algorithm

The algorithm is simple. Given $X_t = (i_t, c_t)$, for each $1 \le a \le k$, 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 $\min_{j=h_a(i),\, 1 \le a \le k} C_{a,j} \ge q$. These can be reported on the fly.

In words, an ID is reported if all counters associated with it have value at least q.

Analysis

It should be clear that a heavy hitter is always reported, since by the algorithm all counters associated with it must have value at least q (just from this ID).

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 $\text{Count}(i,T) \le q - \varepsilon Q$. What is the probability that i is reported?

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

But since $\text{Count}(i,T) \le q - \varepsilon Q$, the ID i has only contributed at most $q - \varepsilon Q$ to counter $C_{a,h_a(i)}$.

This means that other IDs have contributed at least εQ to the counter $C_{a, h_a(i)}$.

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 $\le \frac{1}{\ell}$

Therefore, $E[Z_a] \le Q/\ell$.

By Markov's inequality (!), $\Pr(Z_a \ge \varepsilon Q) \le E[Z_a]/\varepsilon Q \le 1/\varepsilon\ell$.

So, $\Pr(\min_a Z_a \ge \varepsilon Q) \le \left(\frac{1}{\varepsilon \ell}\right)^k$, as the hash functions are chosen independently.

Set $\ell = \frac{e}{\varepsilon}$ and $k = \ln(1/\delta)$, 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 (e.g. $\varepsilon=0.01$ and $\delta=0.01$).

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

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

Distinct Elements

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

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 the items to a table of size n^2 or larger, then with high probability there will be no collision pairs. and thus we can assume that $\log m = O(\log n)$.

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

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

As we have seen in L05, there will be no collision pairs with high probability.

Suppose there are D distinct elements, say $b_1, \dots, b_D$.

Ideally, we want the D hash values $h(b_1), h(b_2), \dots, h(b_D)$ to be evenly distributed in $\{1,2,\dots,m^3\}$.

If that's the case, then the t-th smallest hash value (among these D values) should be close to $\frac{tm^3}{D}$.

So, if we know that the t-th smallest hash value is T, then $T \approx \frac{tm^3}{D}$ 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 hash values that we've seen.

Storing more numbers will give us a better estimate, but of course it requires more space

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

Algorithm

- Choose a random hash function from a strongly 2-universal hash family.
- For each item a_i in the data stream, compute $h(a_i)$.

    Update the list / heap that store the t smallest hash values seen so far.

- After all data read, let T be the t-th smallest hash value in the data stream.

  Return $Y = \frac{tm^3}{T}$ as our estimate.

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

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

This means that $T < \frac{tm^3}{(1+\varepsilon)D} \le \frac{tm^3(1-\frac{\varepsilon}{2})}{D}$, where the second inequality holds for $\varepsilon \le 1$.

This implies 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 random variable that $h(a_i) \le \frac{tm^3(1-\frac{\varepsilon}{2})}{D}$.

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

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

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

To bound the derivation, we compute the variance.

Let $X = \sum_{i=1}^{D} X_i$. Then $E[X_i] = \frac{t(1-\frac{\varepsilon}{2})}{D}$, and so $E[X] = t(1-\frac{\varepsilon}{2})$.

Since h is chosen from a strongly 2-universal family, the hash values of the D distinct elements

are pairwise independent, and thus the indicator random variables X_i are also pairwise independent.

Therefore, by L04, $\mathrm{Var}[X] = \sum_{i=1}^{D} \mathrm{Var}[X_i] \le \sum_{i=1}^{D} E[X_i] = D \cdot \frac{t(1-\frac{\varepsilon}{2})}{D} = t(1-\frac{\varepsilon}{2})$, where the inequality

follows as $\mathrm{Var}[X_i] = p(1-p) \le p = E[X_i]$ for an indicator variable X_i with probability p being 1.

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

$$\le \frac{4\,\mathrm{Var}[X]}{\varepsilon^2 t^2} \le \frac{4(1-\frac{\varepsilon}{2})}{\varepsilon^2 t} \le \frac{4}{\varepsilon^2 t}.$$

Therefore, if we set $t = \frac{24}{\varepsilon^2}$, then $\Pr(X > t) \le \frac{1}{6}$.

Lower bound: We next bound the probability that $Y < (1-\varepsilon)D$. This is very similar to above.

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

By a similar calculation, $E[X] = (1+\varepsilon)t$ and $\mathrm{Var}[X] \le (1+\varepsilon)t$.

By Chebyshev's inequality, $\Pr(X < t) \le \Pr(|X - E[X]| \ge \varepsilon t) \le \frac{\mathrm{Var}[X]}{\varepsilon^2 t^2} \le \frac{2}{\varepsilon^2 t} \le \frac{1}{12}$ for $t = \frac{24}{\varepsilon^2}$. $\square$

Therefore, with probability at least $3/4$, we have $(1-\varepsilon)D \le Y \le (1+\varepsilon)D$.

Error probability: How should we boost the success probability to be at least $1-\delta$ for any $\delta > 0$?

We can do this by running $O(\log\frac{1}{\delta})$ parallel copies and taking the median (prove this).

Space requirement: The total space used is $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\cdot\log m)$ bits, since each copy stores $O(\frac{1}{\varepsilon^2})$

hash values each of $O(\log m)$ bits, and the hash function only requires $O(\log m)$ bits to store.

Running time: The list can be implemented by a heap to support fast updates.

Since there are at most $O(1/\varepsilon^2)$ elements, each operation can be supported in $O(\log \frac{1}{\varepsilon})$ steps.

There is a neat trick to do each operation in amortized $O(1)$ steps.

Optimal algorithm: Kane, Nelson and Woodruff gave an algorithm that uses $O(\frac{1}{\varepsilon^2} + \log m)$ space and $O(1)$ update time. They proved that it is optimal.

Frequency Moments

This is the first result in data streaming.

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

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 mentions results for other p.

Algorithm

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

Analysis

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

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

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

When $i=j$, $E[r_i r_j] = E[r_i^2] = 1$ as $r_i \in \{+1, -1\}$. When $i \ne j$, $E[r_i r_j] = E[r_i]E[r_j] = 0$ as they are independent.

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

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

So, we compute $E[Y^4] = \sum_{1 \le i,j,k,l \le m} x_i x_j x_k x_l \, E[r_i r_j r_k r_l]$.

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

Therefore, there are only two situations when $E[r_i r_j r_k r_\ell] \neq 0$.

- when $i=j=k=\ell$, and this contributes $\sum_{i=1}^{m} x_i^4$ to $E[Y^4]$.
- when two indices appear twice, and this contributes $6\sum_{i<j} x_i^2 x_j^2$ to $E[Y^4]$.

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

Then. $Var[Y^2] = E[Y^4] - (E[Y^2])^2 = \sum_{i=1}^{m} x_i^4 + 6\sum_{i<j} x_i^2x_j^2 - \left(\sum_{i=1}^{m} x_i^2\right)^2$

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

$$= 4\sum_{i<j} x_i^2 x_j^2$$

$$\leq 2\left(\sum_{i=1}^{m} x_i^2\right)^2$$

$$= 2(E[Y^2])^2$$

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

Plugging in $Var[Y^2] \leq 2(E[Y^2])^2$, we have $Pr\left(|Y^2 - E[Y^2]| \geq c\sqrt{2}\,E[Y^2]\right) \leq \frac{1}{c^2}$.

This implies that the output Y^2 is a constant factor approximation to $E[Y^2] = \sum_{i=1}^{m} x_i^2$ with good probability.

Better approximation

To get a $(1\pm\varepsilon)$-approximation, we would like to find a variable $\bar{Y}^2$ with $E[Y^2] = E[\bar{Y}^2]$ but $Var[\bar{Y}^2] < Var[Y^2]$.

The standard way to do this is to compute $Y_1^2, Y_2^2, \ldots, Y_k^2$ independently and take the average.

Let $\bar{Y}^2 = \left(\sum_{i=1}^{k} Y_i^2\right)/k$. Then clearly $E[\bar{Y}^2] = E[Y^2]$.

By independence, $Var[\bar{Y}^2] = \sum_{i=1}^{k} Var[Y_i^2/k] = \frac{1}{k^2}\sum_{i=1}^{k} Var[Y_i^2] = \frac{1}{k} Var[Y^2]$, which is at most $\frac{2}{k}(E[Y])^2$.

Applying Chebyshev's inequality on $\bar{Y}^2$, it follows that $Pr\left(|\bar{Y}^2 - E[\bar{Y}^2]| \geq c\sqrt{2/k}\,E[\bar{Y}^2]\right) \leq 1/c^2$.

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

Space requirement

The space required to compute one Y^2 is to store one number Y.

Since we take the average of $O(1/\varepsilon^2)$ independent copies, so the total space required is $O(1/\varepsilon^2)$ numbers.

Wait, how do we compute Y if we don't store $r_1, r_2, \ldots, r_m$? This requires m bits!

A key point of the analysis is that we only need the variables to be 4-wise independent,

so that the calculations about $E[r_i r_j]$ and $E[r_i r_j r_k r_\ell]$ still hold.

Now, m 4-wise independent bits can be generated from $O(\log m)$ random bits.

So, the extra storage required for the computation in one copy is only $O(\log m)$ bits.

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 MU.
- Distinct elements and frequency moments are from course notes of 6.895 in MIT by Piotr Indyk.
- An optimal algorithm for the distinct element problem, by Kane, Nelson, Woodruff.