Data Science Through a Geometric Lens Fall 2019

Lecture 03 — October 7, 2019

Prof. Cyrus Rashtchian *Topics: Better Distinct Elements*

Overview. In the last lecture we estimated the number of distinct elements d in a stream. First we had an idealized algorithm which uses hash function $h : [n] \to [0,1]$ and inspects the minimum one. Then we explored hash functions that are more practical.

In this lecture, we will fix the idealized algorithm to get an actual (implementable) algorithm by using pairwise independent hash functions. The space we need in the final version of the algorithm is going to be $O(\frac{1}{\epsilon^2}\log^2 n \log\frac{1}{\delta})$. Recent work shows that $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta} + \log n)$ space is achievable and this also matches the lower bound.

We will need pairwise independent hash functions, which we introduced last time:

Definition 1. *A family $\mathcal{H}$ of functions mapping $[a]$ into $[b]$ is* pairwise independent *if for all distinct $i \neq i' \in [a]$ and for $j, j' \in [b]$,*

$$\Pr_{h\in\mathcal{H}}(h(i) = j \text{ and } h(i') = j) = \frac{1}{b^2}.$$

We can also extend this definition to random variables.

Definition 2. *Random variables $Y_1, \ldots, Y_n$ taking values in $[b]$ are* pairwise independent *if for all distinct $i \neq i' \in [n]$ and for $j, j' \in [b]$,*

$$\Pr(Y_i = j \text{ and } Y_{i'} = j) = \Pr(Y_i = j) \cdot \Pr(Y_{i'} = j').$$

Notice that if $Y_i = h(i)$ then the two definitions coincide, because $\Pr_{h\in\mathcal{H}}(h(i) = j) = 1/b$.

The general definition, which makes no assumptions about them being discrete or having the same range, is the following: **"A pairwise independent collection of random variables is a set of random variables such that any two of them are independent."**

The most useful property of such random variables is the following.

Claim 3. *Let $Y_1, \ldots, Y_n$ be some pairwise independent random variables. Then*

$$\mathrm{Var}[\sum_i Y_i] = \sum_i \mathrm{Var}[Y_i].$$

1 Non-Idealized Solution for Distinct Elements

We call the first algorithm NI for non-idealized. We will be able to do NI++ but not NI+. To get a better approximation, we will use a different idea. The idea will be about using geometric sampling to get multiple estimators (buckets), where one of them will be right depending on the value of d, which we don't know. To deal with this, we will get a crude estimate $\widetilde{d}$ using NI and then guess $\widetilde{d} \approx d$ to use the right bucket.

1.1 Constant Factor Non-Idealized Solution: NI

Recall the notation from before. We have elements $i_1, i_2, \ldots \in [n]$ arriving in a stream. We want to estimate $d = |\{i_j\}|$, the number of distinct values i_j that we see.

First, we develop an algorithm NI get a $O(1)$ approximation in $O(\log n)$ bits. That is, we have that with good probability the approximate output $\widetilde{d}$ will satisfy

$$\frac{d}{16} \;\leq\; \widetilde{d} \;\leq\; 16d.$$

The value 16 is not important; any constant will do. After explaining and analyzing NI, we will improve the approximation to $(1+\epsilon)$.

Algorithm NI

Assume that n is a power of two (otherwise, round up). We will use the least significant bit (lsb) of an integer i that is set to one, which we denote $\mathsf{lsb}(i)$. Let's index starting at 1. So the smallest possible index is 1, and the largest is $\log(n) + 1$.

1. Choose h from a 2-wise family mapping $[n]$ to $[n]$.
2. Maintain the largest lsb seen so far, that is,
$$X = \max_i \mathsf{lsb}(h(i))$$
over any hash value $h(i)$ for some i in the stream.
3. Output $\widetilde{d} = 2^X$.

Why does this make sense? We hope $X \approx \log d$, and therefore $\widetilde{d} = 2^X \approx d$.

Focus on some i in the stream. What's the probability that the lsb is j (and the others to the right are 0). This is $1/2^j$ because there are j bits that need to be exactly how you want, e.g., $\mathsf{lsb}(i) = j$ iff $i = [\cdots 10000]_{\text{base } 2}$ with the 1 in position j.

The expected number with this lsb is exactly $d/2^j$. So, if $j \approx \log d$, we expect one value i with $\mathsf{lsb}(h(i)) = j$. But if j is bigger, you aren't going to see these values.

Fix some $j \in \{1, \ldots, \log n + 1\}$. Define the following random variables:

$$Z_j = \#\ i \textit{ such that } \mathsf{lsb}(g(i)) = j.$$

$$Z_{>j} = \#\ i \textit{ such that } \mathsf{lsb}(g(i)) > j.$$

Note that

$$\mathbb{E}[Z_j] = \frac{d}{2^j},$$

and

$$\mathbb{E}[Z_{>j}] = d\left(\frac{1}{2^{j+1}} + \frac{1}{2^{j+2}} + \frac{1}{2^{j+3}} + \cdots\right) < \frac{d}{2^j}.$$

We also need to bound the variance. Fix j and let $Y_i = 1$ if $\mathsf{lsb}(h(i)) = j$. We have that $\mathrm{Var}[\sum_i Y_i] = \sum_i \mathrm{Var}[Y_i]$ due to the pairwise independence we have inherited from our 2-wise hash function. (In fact, that is why we required the 2-wise independence in the first place). Of course, Y_i is just a Bernoulli random variable with probability $\frac{1}{2^j}$ of being heads, so that the variance is

$$\mathrm{Var}[Y_i] = \frac{1}{2^j} \cdot \left(1 - \frac{1}{2^j}\right) < \frac{1}{2^j}.$$

We can then conclude that

$$\mathrm{Var}\left[\sum_{i=1}^{d} Y_i\right] = \sum_{i=1}^{d} \mathrm{Var}[Y_i] < \frac{d}{2^j}.$$

Approximation Guarantee

Two things to handle, X being too small ($X < \log d - 5$) or too big ($X > \log d + 5$).

X is not too small with good probability. We will use Chebyshev's Inequality. Consider some lsb value $j^- = \log d - 4$. We want to show that some element i in the stream has $\mathsf{lsb}(h(i)) \geq j^-$. We know that the expectation of Z_{j^-} is pretty big, and in particular, we have that

$$\mathbb{E}[Z_{j^-}] = \frac{d}{2^{j^-}} = 16.$$

In other words, $Z_{j^-} = 0$ iff $\left|Z_{j^-} - \mathbb{E}\, Z_{j^-}\right| \geq 16$. We can bound the probability of this using Chebyshev. We have already seen that $\mathrm{Var}[Z_j] \leq \frac{d}{2^j}$, and so in this case, the variance is at most 16 as well. Therefore,

$$\Pr(Z_{j^-} = 0) < \frac{\mathrm{Var}[Z_{j^-}]}{16^2} = \frac{1}{16}.$$

When $Z_{j^-} \geq 1$, we will have $X \geq j^-$ because at least one element will have lsb at least j^-.

X is not too big with good probability. We will use Markov's Inequality, and this time, we need to consider that there is no lsb that is too much bigger than $\log d$. Consider $j^+ = \log d + 4$, where we have that

$$\mathbb{E}[Z_{>j^+}] \leq \frac{1}{16}.$$

So by Markov

$$\Pr(Z_{>j^+} \geq 1) \leq \frac{1}{16}.$$

In other words, we have that $Z_i = 0$ for all $i \geq j + 1$, and hence, no lsb bigger than j will be seen with probability at least $1 - 1/16$. That is, $X \leq j^+$ with this probability as well.

X is just right with good probability. By a union bound the probability that either of these happen is at most $5/16$. Hence, we have that X is within ± 4 of $\log d$ with probability at least $11/16 > 2/3$.

Finishing the proof. We have just shown that

$$\log d - 4 \leq X \leq \log d + 4$$

with probability at least 2/3. Therefore, when we output the esimate $\widetilde{d} = 2^X$, we have that

$$\frac{d}{16} \leq \widetilde{d} \leq 16d.$$

How do we improve the error probability? Well, we can go straight to NI++ by taking a median of $O(\log(1/\delta))$ of these. But unfortunately, we can't reduce the approximation by taking the average (that is, we can't do NI+) as far as I can tell. The reason is that the variance is too large to apply Chebyshev. So we need another idea.

1.2 Improving the Approximation: Geometric Sampling of Streams

Suppose we have a substitute that gives us $\widetilde{d}$ as a 16-approximation to d. To get the $(1+\varepsilon)$-approximation, we will use the common strategy of ***geometric sampling of streams***. The idea of using a crude estimate along with geometric sampling is incredible powerful in randomized algorithm.

Recall that the trivial solution was to remember all the elements in the stream. This would take $d \log n$ space. But what if we only remember the first K distinct elements in the stream, with $K = c/\varepsilon^2$. If the $d < K$, then we have the exact answer, and otherwise we don't know much. We can always run this algorithm in parallel, and therefore, we can assume $d \geq K$.

The key idea is to randomly assign stream elements to buckets, so that one of the buckets will receive the 'right' number of elements. Specifically, we will have $\log n + 1$ buckets, and we send an element i to bucket j if $\mathsf{lsb}(g(i)) = j$. In this way, we non-uniformly partition all the elements in the stream.

We use B_j to denote the capped size of bucket j, that is, either it outputs the number of elements it has, or it outputs the maximum possible K. In terms of random variables:

$$B_j = \min\{Z_j, K\},$$

where Z_j from before was the number of elements i sent to bucket j because $\mathsf{lsb}(g(i)) = j$.

Geometric Sampling Algorithm

1. In parallel, run NI++ to get an estimate $\widetilde{d}$ with probability $1 - \delta$.
2. Choose $g : [n] \to [n]$ from a 2-wise family.
3. `init()`: create $\log n + 1$ empty buckets.
4. `update(`i`)`: feed i to bucket number $\mathsf{lsb}(g(i))$.
5. Output: let $j^* = \log(\varepsilon^2 \tilde{d}) - 5$, and return the value $\widetilde{d}^* = 2^{j^*} \cdot B_{j^*}$.

Intuition. The key idea is that we want a bucket to have size $\Theta(1/\varepsilon^2)$. First, observe that j^* ensures this. In particular, if for the true d, we had $j = \log(\varepsilon^2 d)$, then the expected size of bucket j is $\mathbb{E}[Z_j] = 1/\varepsilon^2$. Since our estimate $\tilde{d}$ is pretty good, our value of j^* will also be pretty good, and the bucket will have size roughly $\Theta(1/\varepsilon^2)$.

Now, we want to understand why the estimator is good. The size $\Theta(1/\varepsilon^2)$ is useful because it is big enough. We actually would like bigger buckets, but we capped their max size so we don't use too much space. If they have max size $100/\varepsilon^2$, then they readily handle $1/\varepsilon^2$ elements without becoming full. And if they have size $\Theta(1/\varepsilon^2)$, without being full, then the point is that their variance will behave nicely. One way to do this proof is to say that if the bucket has expected size $B_j = \Theta(1/\varepsilon^2)$, then the probability $B_j * 2^j$ is much bigger than d is actually pretty small. More precisely, we will be within εd with probability at least 3/4. You can formalize this, but we will go a somewhat different route with the calcu;ations (it might be worth trying this yourself).

Analysis. There are two things to understand. First, what is j^* doing? Well, we have that $j^* = \log(\varepsilon^2 \tilde{d}) - 4$. From the NI analysis, we know that $\widetilde{d} \leq 16d$. Therefore, we have that

$$2^{j^*} \leq \varepsilon^2 d.$$

Then, the output is $\widetilde{d}^* = 2^{j^*} \cdot B_{j^*}$. How does this behave? Let's assume we are in the case where $B_{j^*} = Z_{j^*}$ because the bucket did not get full too soon. This is true except with probability δ, so from now on, we will just condition on $B_{j^*} = Z_{j^*} < K$.

Then, since $\mathbb{E}[Z_{j^*}] = d/2^{j^*}$, we have that

$$\mathbb{E}[\widetilde{d}^*] = \mathbb{E}[2^{j^*} \cdot B_{j^*}] = 2^{j^*} \cdot \mathbb{E}[B_{j^*}] = 2^{j^*} \cdot \mathbb{E}[Z_{j^*}] = d.$$

So $\widetilde{d}^*$ is an unbiased estimator.

Now we just need to understand the variance/concentration. This is where $\widetilde{d}$ comes in. We know already know that that $\mathrm{Var}[Z_{j^*}] \leq d/2^{j^*}$. So then we have

$$\mathrm{Var}[\widetilde{d}^*] = \mathrm{Var}[2^{j^*} \cdot B_{j^*}] = 2^{2j^*} \cdot \mathrm{Var}[Z_{j^*}] \leq 2^{j^*} \cdot d.$$

And then by the above calculation that $2^{j^*} \leq \varepsilon^2 d$, we get that

$$\mathrm{Var}[\widetilde{d}^*] \leq \varepsilon^2 d^2.$$

Now we can apply Chebyshev:

$$\Pr(|\widetilde{d}^* - d| \geq 2\varepsilon d) \leq \frac{\mathrm{Var}[\widetilde{d}^*]}{4\varepsilon^2 d^2} \leq \frac{1}{4}.$$

.

To get this to $1 - \delta$ probability, we can just"++" this whole algorithm with $O(\log(1/\delta))$ copies and take the median output value.

How much space do we need? We need to store g with $\log n$ bits, but this won't matter. The main space is to store bucket j for $j \in \{1, \ldots, \log n + 1\}$. Each held up to $O(1/\varepsilon^2)$ elements, and it has to store $\log n$ bits for each element. So this is $O(\log^2(n)/\varepsilon^2)$. In total, we need space

$$O\left(\frac{1}{\varepsilon^2} \cdot \log^2 n \cdot \log \frac{1}{\delta}\right)$$

.

1.3 State-of-the-Art Bounds in Literature

1.3.1 Lower Bound

The lower bound is $\Omega\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta} + \log n\right)$ bits. For those interested in the history of this lower bound, see the following references: [1] [5] [4] [6]

1.3.2 Upper Bound

First was the work on "HyperLogLog", which established

$$O\left(\frac{1}{\varepsilon^2} \log \log n + \log n\right).$$

Recent work from Błasiok (2019) has established

$$O\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta^2} + \log n\right),$$

and so the problem is pretty much completely solved (in the standard streaming setting... many other extensions and other setting to consider of course!!).

References

[1] Noga Alon, Yossi Matias, Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[2] Philippe Flajolet, G. Nigel Martin. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[3] J. Lawrence Carter, Mark N. Wegman. Universal Classes of Hash Functions. *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pp. 106–112, 1997.

[4] Piotr Indyk, David Woodruff. Optimal Approximations of the Frequency Moments of Data Streams. *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pp. 202–208, 2005.

[5] David Woodruff. Optimal Space Lower Bounds for All Frequency Moments. *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 167–175, 2004.

[6] Thathachar Jayram, David Woodruff. The Data Stream Space Complexity of Cascaded Norms. *50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 765–774, 2009.

[7] Philippe Flajolet, Éric Fusy, Olivier Gandouet, Frédéric Meunier. Hyperloglog: The Analysis of a Near-Optimal Cardinality Estimation Algorithm. *AofA: Analysis of Algorithms*, pp. 137–156, 2007.