Algorithms for Big Data Fall 2020

Lecture 06 & 07 — October 14 & 16, 2020

Prof. Cyrus Rashtchian *Topics: Distinct Elements*

Overview. In the last lecture, we considered Morris' algorithm for approximate counting with small space. The main idea was to use multiple estimators to improve the estimation: First the mean in Morris+, and then the median-of-means in Morris++.

In this lecture, we focus on another *streaming problem* – counting *distinct* elements in a stream (the distinct elements problem). For example, how many different hashtags were used on Instagram last year? Or how many different people visited a certain website in a given time frame? For these problem, we don't need exact answers. The approximation algorithms will provide another example of similar ideas, and it will hopefully cement the techniques that may be useful for other problems. We use the notation $[n] = \{1, 2, \ldots, n\}$. We start with a brief review of hash functions.

1 k-wise independent hash functions

We start with two motivating question.

For a random function $h : [a] \to [b]$, the probability that $h(i) = j$ is $1/b$ for all i, j. What about $h(i) = j$ and $h(i') = j'$? When is this equal to $(1/b)^2$ for $i \neq i'$ and any j, j'?

How much space does it take to store h? The total number of possible functions is b^a. So it takes $O(a \log b)$ bits to store h. We need to store $h(i)$ for each $i \in [a]$. Can we do better?

First, we need to understand *pseudorandom* hash functions. We will use **k-wise independent hash functions**. We provide one example. See Wikipedia for many other implementations (that are faster in practice and actually used in common libraries).

Definition 1. *A family $\mathcal{H}$ of functions mapping $[a]$ into $[b]$ is* k-wise independent *iff for all distinct $i_1, \ldots, i_k \in [a]$ and for all $j_1, \ldots, j_k \in [b]$,*

$$\Pr_{h \in \mathcal{H}}(h(i_1) = j_1 \wedge \cdots \wedge h(i_k) = j_k) = \frac{1}{b^k}.$$

Note that we can store $h \in \mathcal{H}$ in memory with $\log_2 |\mathcal{H}|$ bits.

One example of such a family $\mathcal{H}$ is the set of all functions mapping $[a]$ to $[b]$. Then $|\mathcal{H}| = b^a$, and so $\lg |\mathcal{H}| = a \lg b$.

Perhaps the most important/common case is when $k = 2$, and this is known as a *pairwise independent* set of random variables.

1.1 Example

A nice example is due to Carter and Wegman [2], where $\mathcal{H}$ is the set of all degree-$(k-1)$ polynomials over $\mathbb{F}_q$ such that $a = b = q$. Then $|\mathcal{H}| = q^k$, and so $\lg|\mathcal{H}| = k \lg q$. This can be much better when k is small (for example, $k = 2$).

It's a nice exercise to prove that this example has the desired properties.

Claim 2. *Let $\mathcal{H}$ be the set of all degree-$(k-1)$ polynomials over $\mathbb{F}_q$ such that $a = b = q$. Then, $\mathcal{H}$ is k-wise independent.*

Proof. Exercise for Homework 2. □

Having seen these examples, we will just assume that we have access to some 2-wise independent hash families, which will let us store in $\lg n$ bits.

1.2 Pairwise Independence is Great

We observe that *pairwise* independence suffices to get the nice additive property of purely independent random variables.

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

$$\operatorname{Var}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \operatorname{Var}[Y_i].$$

Proof. Exercise for Homework 2. □

2 Counting Distinct Elements in a Stream

The input to the algorithm will be stream of integers $i_1, i_2, i_3, \dots, i_m \in \{1, \dots, n\}$. We want our `query()` operation to approximate the number of distinct integers in the stream.

We can exactly solve this this in $O(\min\{m \lg n, n\})$ bits of memory, but it turns out (though we will not prove this in lecture) that this is essentially as good as we can get. More precisely, we need linear memory if we don't return an answer that is approximate and have an algorithm that is randomized.

Let's say that the true number of distinct elements is d and we output an approximation $\tilde{d}$. More precisely, we want to output some answer $\tilde{d}$ such that $\Pr(|\tilde{d} - d| > \varepsilon d) < \delta$. As usual, the ε is the approximation factor and the δ is the failure probability.

Our goal is to eventually use space $O\left(\frac{\log(1/\delta)}{\varepsilon^2}\right)$.

The first work to achieve this was by Flajolet and Martin in 1985 [1]. We will do something similar, but not quite the same. We call our algorithms 'FM' because of their names. The analysis presented here is similar to the algorithm due to Alon, Matias, and Szegedy [3].

Idealized Solution. We will start with an idealized solution that uses real numbers – and therefore requires infinite memory! But, computers only have finite precision... However, let's pretend that the real numbers don't require infinite memory for now, because the non-idealized solution is similar in spirit.

2.1 Useful Expectation Lemma

We will need the following convenient expression for the expectation (the analogous statement holds for discrete random variables too, that is, if X were discrete and taking non-negative integer values, then $\mathbb{E}X = \sum_{i=0}^{\infty} \Pr[X \geq i]$).

Lemma 4. *If $X \in [0,1]$ is a continuous random variable, then*

$$\mathbb{E}[X] = \int_0^1 \Pr(X \geq z)\, dz$$

Proof. We start with the definition and use a 'dummy' variable t to make it precise:

$$\begin{aligned}
\mathbb{E}X &= \int_0^1 z \cdot f(z)\ dz \\
&= \int_0^1 \left(\int_0^z 1 dt \right) \cdot f(z)\ dz \\
&= \int_0^1 \int_0^z f(z)\ dtdz \\
&= \int_0^1 \int_t^1 f(z)\ dzdt \qquad \text{[integral change explained below]} \\
&= \int_0^1 \Pr(X \geq t)\ dt
\end{aligned}$$

For the swapping of the integrals, notice that the area of integration can be equivalently stated as follows:

$$\begin{aligned}
&\{(t,z) \mid z \in [0,1] \text{ and } t \in [0,z]\} \\
&= \{(t,z) \mid 0 \leq t \leq z \leq 1\} \\
&= \{(t,z) \mid t \in [0,1] \text{ and } z \in [t,1]\}
\end{aligned}$$

□

2.2 FM

Our basic algorithm (which we will call **FM** and subsequently upgrade into **FM+** and **FM++** as in the previous lecture) proceeds as follows:

1. Choose a random hash function $h : [n] \to [0,1]$.
2. Maintain in memory the smallest hash we've seen so far: $X = \min_{i \in \text{stream}} h(i)$.
3. `query()`: output $\frac{1}{X} - 1$.

Intuition: Notice that h is fixed, so if we see a number i at any point in the stream, we have that $h(i)$ is the same. So, seeing i for the second or later time will not change X.

What do we know about X? For example, what is the expectation $\mathbb{E}[X]$ and variance? Let's focus on this, instead of the entire estimator. Then, we will analyze the output $\tilde{d} = \frac{1}{X} - 1$. By linearity of expectation, showing that $\mathbb{E}[\tilde{d}] = d$ is equivalent to showing that $\mathbb{E}[X] = \frac{1}{d+1}$.

We know that d is the true number of distinct elements. Informally, the random hash function h along with the min can be viewed as randomly partitioning the interval $[0, 1]$ into bins of size $\approx 1/(d + 1)$. With this in mind, we claim the following:

Claim 5. $\mathbb{E}[X] = \dfrac{1}{d+1}$.

Proof. We start with the expectation equality in Lemma 4, and then we use the fact that h is uniform is $[0, 1]$ to work out the proof.

$$\begin{aligned}
\mathbb{E}[X] &= \int_0^1 \Pr(X \geq z)\, dz \\
&= \int_0^1 \Pr(\forall i \in \text{stream}, h(i) \geq z)\ dz \qquad \text{[because min is at least } z\text{]} \\
&= \int_0^1 \prod_{i \in \text{stream}} \Pr(h(i) \geq z)\ dz \qquad \text{[by independence]} \\
&= \int_0^1 (1-z)^d\, dz \\
&= \frac{1}{d+1}
\end{aligned}$$

□

We similarly bound $\mathbb{E}[X^2]$, which gives us the variance $\mathrm{Var}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

Claim 6. $\mathbb{E}[X^2] = \dfrac{2}{(d+1)(d+2)}$.

Proof. The proof is similar to the previous claim (start with Lemma 4 for X^2 this time).

$$\begin{aligned}
\mathbb{E}[X^2] &= \int_0^1 \Pr(X^2 \geq z)\, dz \\
&= \int_0^1 \Pr(X \geq \sqrt{z})\, dz \\
&= \int_0^1 (1-\sqrt{z})^d\, dz \qquad \text{[same idea as before]} \\
&= 2\int_0^1 u^d(1-u)\, du \qquad [u = 1-\sqrt{z}] \\
&= 2\int_0^1 u^d - u^{d+1}\, du \\
&= 2\left(\frac{1}{d+1} - \frac{1}{d+2}\right) = \frac{2}{(d+1)(d+2)}
\end{aligned}$$

□

So we can bound the variance now (which we will use soon)

$$\mathrm{Var}[X] = \frac{2}{(d+1)(d+2)} - \frac{1}{(d+1)^2} = \frac{d}{(d+1)^2(d+2)} < \frac{1}{(d+1)^2}. \tag{1}$$

Notice that $\mathrm{Var}[cX] = c^2 \mathrm{Var}[X]$ for any number $c > 0$ by properties of the variance (and this holds for any random variable). We also used this last lecture.

2.3 FM+

We improve things by running the algorithm many times. Specifically, we can upgrade our basic algorithm into **FM+** by running it $s = \frac{1}{\varepsilon^2 \eta}$ times in parallel to obtain $X_1, \ldots, X_s$. Then `query()` should output

$$\tilde{d}^+ = \frac{1}{\frac{1}{s}\sum_{i=1}^s X_i} - 1.$$

Notice that by linearity of expectation, we have that the expectation is still d because the expectation of $\frac{1}{s}\sum_{i=1}^s X_i$ is equal to $1/(d+1)$.

We want to know how good this might be in practice, or in other words, what's the probability of being far from the expectation? We can use Chebyshev's inequality for this.

Claim 7. *If $s = \frac{1}{\varepsilon^2 \eta}$, then*

$$\Pr\left(\left|\frac{1}{s}\sum_{i=1}^s X_i - \frac{1}{d+1}\right| > \frac{\varepsilon}{d+1}\right) < \eta.$$

Proof. By Chebyshev's inequality, we can bound this as

$$\Pr\left(\left|\frac{1}{s}\sum_{i=1}^s X_i - \frac{1}{d+1}\right| > \frac{\varepsilon}{d+1}\right) \le \frac{\mathrm{Var}[\frac{1}{s}\sum_i X_i]}{\frac{\varepsilon^2}{(d+1)^2}} < \frac{1}{\varepsilon^2 s} = \eta,$$

where the final inequality used Eq. (1). □

We can now get a linear scaling failure probability (that is, s depends linearly on $1/\eta$). For our estimate, we will output

$$\tilde{d}^+ = \frac{1}{\frac{1}{s}\sum_{i=1}^s X_i} - 1$$

and we claim that will be correct with good probability.

Claim 8. *Let $\tilde{d}^+ = \frac{1}{\frac{1}{s}\sum_{i=1}^s X_i} - 1$ be the output of the algorithm. Then, we have*

$$\Pr\left(\left|\tilde{d}^+ - d\right| \ge 2\varepsilon d\right) \le \eta$$

assuming that $\frac{2}{d} < \varepsilon < \frac{1}{4}$.

Proof. Exercise. □

This gives us a *linear* dependence on the failure probability, but we want *logarithmic.*

Just like in the previous lecture, we will take the median of independent copies, and then the final analysis will apply a Chernoff bound.

2.4 FM++

This is another example of the **median-of-means** idea.

We use $t = O(\ln(1/\delta))$ independent copies of **FM+** to get a better algorithm. Precisely, we will set $\eta = 1/3$ in the **FM+** instantiations to get a success probability of at least $2/3$ for each. Then `query()` outputs the *median* across all **FM+** estimates. We call this **FM++**, denoted by

$$\tilde{d}^{++} = \text{median}(\tilde{d}_1^+, \tilde{d}_2^+, \dots, \tilde{d}_t^+),$$

where $\tilde{d}_i^+$ is the ith **FM+** instantiation for $i = 1, 2, \dots, t$.

The new space for **FM++** is now $O\left(\frac{1}{\varepsilon^2} \lg \frac{1}{\delta}\right)$. This does not include the space for h.

Say that the ith copy of **FM+** succeeds if **succeeds** if $|\tilde{d}_i^+ - d| < \varepsilon d$, and otherwise it fails.

Claim 9. *Let $\tilde{d}^{++}$ be defined as above. Then, for $t = O(\log(1/\delta))$, we have*

$$\Pr(|\tilde{d}^{++} - d| \geq \varepsilon d) < \delta$$

Proof. To analyze **FM++**, we define the following t indicator variables:

$$Y_i = \begin{cases} 1, & \text{if the } i\text{-th } \textbf{FM+} \text{ instantiation succeeds.} \\ 0, & \text{otherwise.} \end{cases}$$

From Claim 8, we know that for all i we have that $\mathbb{E}[Y_i] \geq 2/3$. In other words, we have

$$\mathbb{E} \sum_{i=1}^{t} Y_i \geq \frac{2t}{3}$$

The median fails if more than half the copies fail. Then by the Chernoff bound,

$$\Pr\left(\sum_{i=1}^{t} Y_i \leq \frac{t}{2}\right) \leq \Pr\left(\left|\sum_{i=1}^{t} Y_i - \mathbb{E}\sum_{i=1}^{t} Y_i\right| \geq \frac{t}{6}\right) \leq 2e^{-ct} < \delta,$$

for a constant c, where $c = 1/48$ seems to work. The final inequality "$< \delta$" holds by setting the number of estimators to be $t = \Theta(\log(1/\delta))$.

This implies that $|\tilde{d}^{++} - d| < \varepsilon d$ with probability at least $1 - \delta$, as desired. In other words, because we set t appropriately, we get that this fails with probability $e^{-\ln(1/\delta)} = \delta$ □

References

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

[2] 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.

[3] Noga Alon, Yossi Matias, Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pp. 20–29, 1996.