COMS E6998-9: Algorithmic Techniques for Massive Data — Sep 10, 2015

Lecture 2 – Median trick, Distinct Count, Impossibility Results

Instructor: *Alex Andoni* — Scribes: *Shubham Bansal, Chen-Yu Yang*

1 Introduction

Today's lecture has three main topics that we'll go through, i.e. Median Trick (from previous lecture), Distinct element count and Impossibility Results.

2 Median Trick

So far, we have an algorithm A which estimates in correct range of ϵ with probability ≥ 0.9. Our new algorithm A^* will output in range of ϵ with probability $1-\delta$. Algorithm:

- Repeat A for $m = O(log(1/\delta))$ times
- Take median of all the m answers.

To prove the correctness, we'll use Chernoff/Hoeffding bounds.

Definition 1 (Chernoff/Hoeffding Bound)**.** *Let $X_1, X_2, \ldots, X_m$ be independent random variables $\in \{0,1\}$, $\mu = E[\Sigma_i X_i], \epsilon \in [0,1]$. Then $Pr[|\Sigma_i X_i - \mu| > \epsilon\mu] \leq 2e^{-\epsilon^2\mu/3}$*

Define $X_i = 1$ iff the i^{th} answer of A is correct (i.e. estimated value of A lies in correct range).

Claim 2. *$E[X_i] = 0.9$, and $E[\mu] = 0.9m$*

Proof. Since A is correct with probability 0.9, $E[X_i] = 0.9$. And $E[\mu] = 0.9m$ due to linearity of expectation. □

Claim 3. *New algorithm A^* is correct when $\Sigma_i X_i > 0.5m$*

Proof. Since we are considering median value to be our answer, if more than half the trials of A are correct, algorithm A^* is also correct. □

Claim 4. *To prove, $Pr[\Sigma_i X_i \geq 0.5m] \geq 1-\delta$ or $Pr[\Sigma_i X_i < 0.5m] < \delta$*

Proof.

$$\begin{aligned} Pr[\Sigma_i X_i < 0.5m] &= Pr[\Sigma_i X_i - 0.9m < -0.4m] \\ &\leq Pr[|\Sigma_i X_i - \mu| > 0.4m] \\ &= Pr[|\Sigma X_i - \mu| > 0.4/0.9\mu] \end{aligned} \tag{1}$$

Using Chernoff bound,

$$\begin{aligned} &\leq e^{-c*0.9m} \\ &< \delta \end{aligned} \quad (2)$$

Above equation holds for $m = O(log(1/\delta))$ □

3 Distinct Elements

Given, a stream of size m containing numbers from $[n]$, we have to approximate the number of elements with non-zero frequency. To calculate the exact value the space required:

- $O(n)$ bits. (maintain a vector of length n).
- $O(m \log(n))$ bits. (save m numbers, each taking $log(n)$ bits).

Since, this complexity is not feasible as m,n can be very large, we'll look at algorithm for approximating the distinct count value.

3.0.1 Hash Function

- $h : [n] \to [0, 1]$
- $h(i)$ is uniformly distributed in $[0, 1]$.

3.1 Algorithm [Flajolet-Martin 1985]

We maintain a variable z.

1. Initialize $z = 1$.
2. Whenever i is encountered: $z = \min(z, h(i))$
3. When done, output $1/z - 1$.

Now, we'll prove the algorithm works in a similar fashion followed in previous lecture. Let d be number of distinct elements.

Claim 5. $E[z] = d + 1$

Proof. z is the minimum of d random numbers in $[0, 1]$. Pick another random number $a \in [0, 1]$. The probability $a < z$:

1. exactly z
2. probability it's smallest among $d + 1$ reals : $1/(d + 1)$

Equating these two, one can prove the claim. □

Claim 6. $var[z] \leq 2/d^2$

Proof. It can be done in a similar fashion described in previous lecture. □

3.1.1 $(1+\epsilon)$ approximation Algorithm

We can take $Z = (z_1 + z_2 + ...z_k)/k$ for independent $z_1, ...z_k$

3.2 Alternate Algorithm: Bottom-k

Instead of just use the minimum value of hash function for i inputs, we'll maintain the k smallest hashes seen.

1. Initialize $(z_1, z_2, ...z_k) = 1$.
2. Keep k smallest hashes seen, s.t. $z_1 \le z_2 \le ...z_k$
3. When done, output $\hat{d} = k/z_k$

Claim 7. *The following claims are stated:*

- $Pr[\hat{d} > (1+\epsilon)d] \le 0.05$
- $Pr[\hat{d} < (1-\epsilon)d] \le 0.05$
- *Overall probability that* $\hat{d}$ *outside range is at most 0.1*

Proof. To compute $Pr[\hat{d} > (1+\epsilon)d]$:

- Define $X_i = 1$ iff $h(i) < \dfrac{k}{(1+\epsilon)d}$
- Then $\hat{d} > (1+\epsilon)d$ iff $\Sigma_i X_i > k$
- if $\Sigma_i X_i > k$
  $\iff \exists$ at least k numbers for which $h(i) < \dfrac{k}{(1+\epsilon)d}$
  $$\iff z_k < \frac{k}{(1+\epsilon)d} \iff \frac{k}{z_k} > (1+\epsilon)d \iff \hat{d} > (1+\epsilon)d \tag{3}$$
- $E[X_i] = \dfrac{k}{(1+\epsilon)d}$
  $E[\Sigma_i X_i] = dE[X_i] = \dfrac{k}{1+\epsilon}$
  $\text{var}[\Sigma_i X_i] = d\text{var}[X_i] \le dE[X_1^2] \le \dfrac{k}{1+\epsilon} \le k$
  (Since $X_1 \in \{0,1\}$, $E[X_1^2] = E[X_i]$)
- By Chebyshev: $Pr[|\Sigma X_i - \dfrac{k}{1+\epsilon}| > \sqrt{20k}] \le 0.05 \implies Pr[\Sigma X_i > \dfrac{k}{1+\epsilon} + \sqrt{20k}] \le 0.05$
  - (For $\epsilon < 1/2$ and $k = c/\epsilon^2$)
    $\dfrac{k}{1+\epsilon} + \sqrt{20k} \le k(1-\epsilon+\epsilon^2) + \sqrt{20k}$ (Taylor Series Expansion)
    $\le k - k\epsilon/2 + 5\sqrt{c}/\epsilon = k - c/2\epsilon + 5\sqrt{c}/\epsilon$
    $< k$ where $c > 100$

– Since $k > \frac{k}{1+\epsilon} + \sqrt{20k}$ in our case and ΣX_i is monotonically increasing, $Pr[\Sigma X_i > k] \leq Pr[\Sigma X_i > \frac{k}{1+\epsilon} + \sqrt{20k}] \leq 0.05$

□

3.3 Hash functions in stream

The hash function we used has two practical issues: (1) the return value should be a real number. (2) how do we store it?

Discretization can solve the first issue. Instead of all the real numbers in $[0, 1]$, we use hash function with range $\{0, \frac{1}{M}, \frac{2}{M}, \frac{3}{M}, \ldots, 1\}$. For large $M \gg n^3$, the probability that $d \leq n$ random numbers collide is at most $\frac{1}{n}$.

For the second issue, we use pairwise independent function instead of independent function.

Definition 8. *$h : [n] \to \{1, 2, \ldots M\}$ is pairwise independent if for all $i \neq j$ and $a, b \in [M]$, $Pr[h(i) = a \wedge h(j) = b] = \frac{1}{M^2}$*

It works because in previous calculation, we only care about pairs. We defined $X_i = 1$ iff $h(i)$ is small than a threshold, then we computed $\text{var}[\Sigma X_i] = E[(\Sigma X_i)^2] - E[(\Sigma X_i)^2] = E[X_1 X_1 + X_1 X_2 + \ldots] - E[(\Sigma X_i)^2]$. Notice that $E[X_i X_j]$ is the same for fully random h and pairwise independent h.

Example 9 (Construct a pairwise independent hash)**.** *Assume M is a prime number (if not, we can always pick a larger M that is a prime number). We pick $p, q \in \{0, 1, 2, \ldots M - 1\}$ and the hash function $h(i) = pi + q \mod M$. In this construction we only need $O(\log M) = O(\log n)$ space (to store p, q, M).*

Proof. $h(i) = a, h(j) = b$ is equivalent to $pi + q \equiv a, pj + q \equiv b$. So $p(i - j) \equiv a - b$ and $p \equiv (a - b)(i - j)^{-1}, q \equiv a - pi$. Since M is a prime number, the unique inverse implies that there is only one pair (p, q) satisfies it. And the probability that pair is chosen is exactly $\frac{1}{M^2}$. □

4 Impossibility Results

We have used both approximation and randomization to solve the distinct counting problem with space much less than $\min(m, n)$. Now we are wondering: can we omit either approximation or randomization to achieve the same space efficiency? The answer is no.

4.1 Deterministic Exact Won't Work

First, we will show that there is no deterministic (no randomization) and exact (no approximation) way to solve it.

Suppose there do exists a deterministic and exact algorithm A and an estimator function R that use space $s \ll n, m$. That is, for a given integer stream, we first run the algorithm A on the stream. As the stream goes A will return middle memory steps, and we obtain the final memory state σ after the stream ends. Then we apply R on σ to obtain our estimator $\hat{d}$. Since both A and R are deterministic and exact, $\hat{d}$ must equals to the distinct count for the stream.

We now build a binary representation x of the stream with the following rules: (1) $x \in \{0, 1\}^n$, (2) i in stream iff $x_i = 1$. For example, if 1, 3, 5, 6, 7 are in the stream and 2, 4 are not, x will start with

1, 0, 1, 0, 1, 1, 1. Notice that each stream has a corresponding representation and streams containing different numbers have different representations.

Claim 10. *We can recover the x of the stream given the memory state σ*

Proof. Denote $d = R(\sigma)$ be the original estimator. Now we treat σ as a middle snapshot of the memory and add integer i as the next element of the stream. Now A will return another memory state σ', and $d' = R(\sigma)'$ will be our new estimator. If $d' = d$, i must have appeared in the stream before since A and R are deterministic and exact. Similarly, if $d' > d$, i must have not appeared in the stream before. Using this method with $i = 1, 2, 3 \ldots$ and we can recover the x. □

Since we can recover x from σ, we can treat σ as an encoding of a string x of length n. But σ has only $s \ll n$ bits! Furthermore, we can treat A, the function that produces σ, as a function with domain $\{0,1\}^n$ and $\{0,1\}^s$. We can see that A must be injective because if $A(x) = A(x') = \sigma$, the recoverability implies $x = x'$.

Hence $s \geq n$. Which implies that there is no deterministic and exact algorithm A and an estimator function R that use space $s \ll n, m$.

4.2 Deterministic Approx. Won't Either

We can use the similar strategy to prove that deterministic approx. won't work. We pick $T \subset \{0,1\}^n$ that satisfies the following conditions: (1) for all distinct $x, y \in T$, the number of digits i that $y_i = 1$ and $x_i = 0$ should $\geq \frac{n}{6}$. (2) $|T| \geq 2^{\Omega(n)}$. Now we use algorithm A to encode an input x into $\sigma = A(x)$ and our estimator would be $\hat{d} = R(\sigma)$.

Now we want to recover x based on σ, as what we have done in the last section. For a given σ and any $y \in T$, we append y to the stream and apply A on it, and A will return a memory state σ'. Using σ' we have new estimator $\hat{d'} = R(\sigma')$.

Claim 11. *If $\hat{d'} > 1.01\hat{d}$, then $x \neq y$, else $x = y$.*

Proof. The idea is that when $x = y$, $\hat{d}$ would be really close to $\hat{d'}$ (up to $(1+\epsilon)^2$ because both of them are ϵ-approximated) and when $x \neq y$, the construction of T guarantee that $\hat{d} \geq \hat{d} + \frac{n}{6}$. So we can pick an ϵ that works for our claim. □

We can use this method to check every element $y \in T$ to see if $y = x$, and eventually we can recover x from it. Similar to last section, we can show that A is an injective function and it implies that $2^s \geq |T|$ or $s = \Omega(n)$.

5 Concluding Remarks

- We can use median trick and Chernoff bound to improve the probability of an existing algorithm.
- For distinct elements problem, we can also store the hashes $h(i)$ approximately. One example is to store the number of leading zeros, and it only cost $O(\log \log n)$ bits per hash value, and that is the idea behind another algorithm called HyperLogLog.
- For the impossibility results, we can also prove that randomized exact algorithm won't work.