Massive Data Algorithmics

The Streaming Model

Lecture 14: The Number of Distinct Elements

The Problem

- Input: the stream $\sigma =< a_1, \cdots, a_m >$ where $a_i \in [n]$
- Output: The number of distinct elements denoted by d ($d = ||F||_0$)

Some Known Facts

- It is provably impossible to solve this problem in sublinear space if one is restricted to either deterministic algorithms (i.e. $\delta = 0$), or exact algorithms (i.e. $\varepsilon = 0$).
- Thus, we should seek a randomized approximation algorithm

AMS Algorithm

- A $(O(1), \sqrt{2}/3)$-algorithm to approximate d
- $\text{zeros}(p) = \max\{i : 2^i | p\}$

Initialize :

Choose a random hash function $h : [n] \to [n]$ from a 2-universal family ;

$z \leftarrow 0$;

Process j :

if $\text{zeros}(h(j)) > z$ **then** $z \leftarrow \text{zeros}(h(j))$;

Output : $2^{z+\frac{1}{2}}$

The Basic Intuition

- We expect 1 out of d distint tokens to hit $\text{zeros}(h(j)) \geq \log d$
  - Let $A = \{a : \text{zeros}(a) \geq \log d\}$. We know $|A| = n/d$
  - For j if $h(j) \in A$, we set $X_j = 1$. Otherwise we set $X_j = 0$.
  - Let $X = \sum_{j:f_j>0} X_j$. we know $E(X_j) = \Pr(X_j = 1) = |A|/n = 1/d$, then $E(X) = \sum_{j:f_j>0} 1/d = 1$
- we don't expect any token to hit $\text{zeros}(h(j)) >> \log d$
  - Let $A = \{a : \text{zeros}(a) \geq c \log d\}$. We know $|A| = n/d^c$
  - For j if $h(j) \in A$, we set $X_j = 1$. Otherwise we set $X_j = 0$.
  - Let $X = \sum_{j:f_j>0} X_j$. we know $E(X_j) = \Pr(X_j = 1) = |A|/n = 1/d^c$, then $E(X) = \sum_{j:f_j>0} \mathrm{1/d\^{}c} = 1/d^{c-1}$
- Thus, the maximum value of $\text{zeros}(h(j))$ over the stream—which is maintained in z—should give us a good approximation to $\log d$

Analysis

- For given r, j, let $X_{r,j}$ be an indicator random variable for the event "zeros$(h(j)) \geq r$"
- Let $Y_r = \sum_{j:f_j>0} X_{r,j}$
- Observation: $Y_r > 0 \Leftrightarrow t \geq r$ where t is the value of z when the algorithm finishes the stream (or $Y_r = 0 \Leftrightarrow t \leq r-1$)
- $E(X_{r,j}) = \Pr(\text{zeros}(h(j)) \geq r) = \Pr(2^r | h(j)) = 1/2^r$
- $E(Y_r) = \sum_{j:f_j>0} E(X_{r,j}) = d/2^r$
- $\mathrm{Var}(Y_r) = \sum_{j:f_j>0} \mathrm{Var}(X_{r,j}) \leq \sum_{j:f_j>0} E(X_{r,j}^2) = \sum_{j:f_j>0} E(X_{r,j}) = d/2^r$
- $\Pr(Y_r > 0) = \Pr(Y_r \geq 1) \leq E(Y_r)/1 = d/2^r$
- $\Pr(Y_r = 0) \leq \Pr(|Y_r - E(Y_r)| \geq d/2^r) \leq \mathrm{Var}(Y_r)/(d/2^r)^2 \leq 2^r/d$

Analysis

- Let $\hat{d}$ be the output of the algorithm. So, $\hat{d} = 2^{t+1/2}$
- Let a be the smallest integer s.t. $2^{a+1/2} \geq 3d$
- $\Pr(\hat{d} \geq 3d) = \Pr(t \geq a) = \Pr(Y_a > 0) \leq d/2^a \leq \sqrt{2}/3$
- Let b be the largest integer s.t. $2^{b+1/2} \leq d/3$
- $\Pr(\hat{d} \leq d/3) = \Pr(t \leq b) = \Pr(Y_{b+1} = 0) \leq 2^{b+1}/d \leq \sqrt{2}/3$

Analysis

- space: $O(\log n)$ bits to store and compute a suitable hash function, and $O(\log \log n)$ bits to store z
- The probability error is $\sqrt{2}/3$ which is almost 47%
- $\hat{d}$ is somehow 3-approximation of d
- One idea to decrease the probability error is to replace "3" with a large number. But, the approximation factor gets larger.

Median Trick

- Imagine running k copies of AMS algorithm in parallel using mutually independent random hash functions and output the median of the k answers
- If this median exceeds $3d$, then at least $k/2$ of the individual answers must exceed $3d$, whereas we only expect $k\sqrt{2}/3$ of them to exceed $3d$.
- By a Chernof bound, this event has probability $2^{-\Omega(k)}$.
- Similarly, the probability that the median is below $d/3$ is also $2^{-\Omega(k)}$
- Choosing $k = \Theta(\log(1/\delta))$, gives us an $(O(1), \delta)$-approximation to d

2-Universl Hash Function

- Ideally, we would like the token j uniformly at random (u.a.t) is mapped to an element in $[n]$, denoted by $h(j)$
- Since we may see several occurrence of j, then we should remember where j is mapped uppon the arrrival of the first occurrence of j.
  - We have to maintain all $h(j)$ in a data structure which of course needs a linear data structure, or
  - we use an explicit formula for the hash function
    - It seems we have a fix function and each input is uniquely mapped to an element of $[n]$

2-Universl Hash Function

- Instead of mapping j u.a.r to an element in $[n]$ in the online fashion, we can do all randomness in the offline fashion.
- At the begining, we specify $h(j)$ u.a.t. instead of waiting to receive the first j and specify $h(j)$. The result of course is the same. For isntance, for planning a single-elimination knockout tournament over n teams, we can construct a tree with n leaves and put a random permutation of teams in the leveas or in each round the opponent of each team is u.a.r. specified. The result is the same meaning that the probability that two teams meet each other is the same in both methods.
- This is equivalent to select a fucntion $h : [n] \to [n]$ u.a.r from all functions from $[n] \to [n]$.
- If we do that, we don't have an explicit formula for h and we need a data structure to maintain h.

2-Universl Hash Function

- A practical solution is to put some functions (not all functions) with explicit formula into a set H (called a family of hash functions) and select one of them u.a.r at the begining (offline fashion).
- As we restrict ourself to some functions, the main question is that whether do we have the following property or not $\forall k, \forall j_1, \cdots, j_k, i_1, \ldots, i_k \in [n] : \Pr(h(j_1) = i_1, \cdots, h(j_k) = i_k) = 1/n^k$
- Seems hard to have a family statisfying the property for any k.
- For most applications, having the above property for $k \leq 2$ sufficies. Such a family is called a 2-universal family.
- How to show the family H is 2-universal: for any distinct i and j, count the number of hash functions h that $h(i) = h(j)$. Divide this number by $|H|$. If it always (for any distinct i and j) is equal to $1/n$, H is 2-universal.

References

- **Data Stream Algorithms** (Chapter 2)
  Lecture notes by A. Chakrabbarti and D. College