6.1220 LECTURE 21

BRENDON JIANG

Contents

1 Sketching and Streaming . . . 1
1.1 Counting . . . 1
1.2 Distinct Elements . . . 3
1.3 Similarity Search . . . 5

1. Sketching and Streaming

1.1. Counting

Suppose we are given a dataset X, we want to compress X into a sketch $S(X)$ such that $f(X)$ can be computed/approximated given only $S(X)$. X will be given via a data stream which you get to read one-step at a time. We will need to update $S(X)$ "on-the-fly".

Let us start with a simple example.

Example 1.1

Data Stream: $X = 1, 1, 1, \ldots$ of some unknown length n. The goal is to compute $f(X) = n =$ # of 1s .

Solution 1.2

The easy algorithm is to just maintain a counter $S = \#of1$s seen so far. The space used is then $\log n$. Note that this algorithm computes the optimal solution.

What if we want $o(\log n)$ space and an approximation instead?

Solution 1.3: First Attempt

Increment the counter S every kth 1 for constant k. At the end, output $k \cdot S$. However, we didn't really save space here, as the space is $\log(n) - \log(k)$.

Date: November 2024

Example 1.4: Second Attempt

Instead of counting n, we approximate n by counting $\log n$. However, currently it is unclear when we should increment $S(X)$. We can get around this through randomization, and achieve $O(\log \log n)$ space until return.

Algorithm 1: Morris' Algorithm

```
Initialize S ← 0
while stream is not empty do
    Sample b ∈ {0,1} where b = 1 with probability 1/2^s.
    if b=1 then
        Increment S ← S + 1
return 2^s − 1.
```

Now let us prove this algorithm is actually a good approximation. Suppose $\tilde{f}(n) = 2^S - 1$ for S produced from running this algorithm on a length-n stream of 1s.

Lemma 1.5

$\forall n \in \mathbb{N}, \mathbb{E}[\tilde{f}(n)] = n$ and $Var(\tilde{f}(n)) < \frac{1}{2}n^2$.

Proof

We will prove by induction. In the base case, we note that $\tilde{f}(0) = 0, \tilde{f}(1) = 1$ with probability 1. In the inductive step, we find

$$\begin{aligned}
\mathbb{E}[2^{S(k+1)}] &= \sum_{j=0}^{k} \mathbb{E}[2^{S(k+1)} | S(k) = j] \cdot Pr[S(k) = j] \\
&= \sum_{j=0}^{k} (\frac{1}{2^j} \cdot 2^{j+1} + (1 - \frac{1}{2^j}) \cdot 2^j) \cdot Pr[S(k) = j] \\
&= \sum_{j=0}^{k} (2^j + 1) \cdot Pr[S(k) = j] \\
&= \sum_{j=0}^{k} 2^j \cdot Pr[S(k) = j] + \sum_{j=0}^{k} Pr[S(k) = j] \\
&= k + 1 + 1
\end{aligned}$$

A similar proof establishes the variant part of the lemma.

Corollary 1.6

$$\Pr[|\tilde{f}(n) - n| > \epsilon] \leq \frac{1}{2\epsilon^2}$$

Proof

Use Chebyshev.

Corollary 1.7

$\forall n \in \mathbb{N}, \Pr[(1-\epsilon)n \leq \tilde{f}(n) \leq (1+\epsilon)n] \geq 1 - \dfrac{1}{2\epsilon^2}$.

To boost the accuracy and success probability of our algorithm, we can simply run it a bunch of times in the same data stream and return the median of the mean.

1.2. Distinct Elements

Data Stream: Length-n sequence X of elements from some universe $\mathcal{U}$. Our goal is to estimate $d = d(X) = \#$ of distinct elements occuring in the stream X. Note that there are two easy algorithms, which is to maintain a boolean array and a container of the distinct elements respectively.

Algorithm 2: Flajolet-Martin Algorithm

Initialize $h : \mathcal{U} \to [0,1]$ where $h(u) \sim Unif[0,1]$ independently for all $u \in \mathcal{U}$ Initialize $S \leftarrow 1$
while *stream is not empty:* **do**
  Let x be the next stream element Update $S \leftarrow \min\{S, h(x)\}$
return $\dfrac{1}{S} - 1$

Lemma 1.8

Let $S(X)$ denote the sketch produced given a sequence X.

$$\forall X \in \mathcal{U}^*, \mathbb{E}[S(x)] = \frac{1}{d+1} \text{ and } Var(S(x)) = \frac{d}{(d+1)^2}(d+2)$$

Proof

Using layered cake representation, we find

$$\begin{aligned} LHS &= \int_0^1 \Pr[\min\{u_1,\ldots,u_d\} \geq y]\mathrm{d}y \\ &= \int_0^1 \Pr[u_i \geq y \forall i]\mathrm{d}y \\ &= \int_0^1 \Pr[u_i \geq y]^d\mathrm{d}y \\ &= \int_0^1 (1-y)^d\mathrm{d}y \\ &= \frac{1}{d+1} \end{aligned}$$

Again, we can boost the probability of this algorithm by running it multiple times and merging the answers.

Right now, there are two problems with this algorithms. We can efficiently store a truly uniformly random hash function $h : \mathcal{U} \to [0, 1]$. We also want h to map to a discrete set instead. Recall that

Definition 1.9

A family of hash functions is 2-wise independent if $\forall x, y \in \mathcal{U}$, and $\forall s, t \in \{0, \ldots, m-1\}$,

$$\mathbb{P}_{h\sim\mathcal{H}}[h(x) = s, h(y) = t] = \frac{1}{m^2}$$

Algorithm 3: k-Minimum Value (KMV) Algorithm:

```
Initialize h : U → {0, ..., m − 1} from 2-wise independent hash family H
Initialize S =
Initialize k = 24/ε²
while stream is not empty: do
    Let x be the next stream element
    Update S ← {k smallest elements of S ∪ {h(x)}}
if |S| = k then
    return km / max(S)
else
    return |S|
```

This algorithm has space $O(\frac{1}{\epsilon^2}\log|\mathcal{U}|)$. Update Time: $O(\log(|\mathcal{U}|)\cdot\log(\frac{1}{\epsilon}))$

1.3. Similarity Search

We are now not using the streaming model. The input to this problem is a collection of sets $A_1, \dots, A_n \subseteq \mathcal{U}$ where $|A_i| \le d$ for all $i \in [n]$. The queries we want is for a new set $A \subseteq \mathcal{U}$ satisfying $|A| \le d$ and $s \in [0,1]$, does there exist A_i such that the Jaccard similarity satisfies $J(A, A_i) \ge s$? Here, we define

$$J(A, A_i) := \frac{|A \cap A_i|}{|A \cup A_i|}$$

The easy algorithm is to just compute $J(A, A_i)$ exactly for each $i \in n$, and obtain a time bound of $O(nd)$.

Our goal is to construct a hash function $\sigma : 2^{\mathcal{U}} \to [0,1]$ such that similar sets collide. Then you just search within the bucket indexed by $\sigma(A)$. We can perhaps build a perfect static hash table with key-value pairs: Key $= \sigma(A_i)$, Value $= A_i$.

Lemma 1.10: Min-Hash Lemma

For a uniformly random $h : \mathcal{U} \to [0,1]$, define the min-hash

$$\sigma_h(A) := \min_{x \in A} h(x)$$

Then for all sets $A, B \subseteq \mathcal{U}$,

$$\Pr_h[\sigma_h(A) = \sigma_h(B)] = J(A, B)$$

Proof

The function h induces an ordering π on $\mathcal{U}$ where $x < y$ iff $h(x) < h(y)$. This means

$$\begin{aligned} LHS &= \Pr[\text{min of A under } \pi = \text{min of B under } \pi] \\ &= \Pr_\pi[\{\text{min of } A \cup B \text{ with respect to } \pi\} \in A \cap B] \\ &= \frac{|A \cap B|}{|A \cup B|} \end{aligned}$$

We want to boost the probability. If $J(A, B) \ge s$, boost the probability that $\sigma(A) = \sigma(B)$. If $J(A, B) < \eta < s$, we reduce the probability that $\sigma(A) = \sigma(B)$ (so that we don't have to search through junk).