6.897: CSCI8980 Algorithmic Techniques for Big Data September 5, 2013

Lecture 1

Dr. Barna Saha *Scribe: Vivek Mishra*

Overview

We introduce data streaming model where streams of elements are coming in and main memory space is not sufficient to hold all the data. We then look at the problem of finding frequent items deterministically. This is a rare instance of data streaming algorithms that provides non-trivial approximation guarantee deterministically. For most algorithms as we will see the answer is approximate (close to optimal) and randomization is crucially used. To emphasize on the need of randomization in designing data streaming algorithms, we next show that computing distinct items (known as F_0 computation) over a stream deterministically cannot achieve any approximation without essentially storing the entire stream. Our next goal is to analyze the algorithm for distinct items that have been covered in Lecture 1. Towards that goal, we study some basic concentration inequality such as, Markov inequality, Chebyshev bound and The Chernoff bound.

1 Introduction to Data Streams

In a data streaming model sequence of elements $a_1\ a_2\ a_3\ a_m$ arrive from a domain $[1, n]$. Each element a_i is a tuple (j, ν) where $j \in [1, n]$ is an element from the domain and $\nu \in \mathbb{I}$. For simplicity, we can consider $\nu = \pm 1$ where $+1$ implies j is inserted in the stream and -1 implies j is deleted.

The goal is to process these elements using space (ideally) polylog of m and n, but definitely in sub-linear in m and n. In the basic streaming setting, only a single pass over the data is allowed. The time to process each update must be low.

When only insertions are allowed (all $\nu > 0$), the model is known as *cash-register* model. On the other hand, if both insertions and deletions are allowed, it is called *turnstile* model. For any element, we generally do not allow the number of deletions (total negative frequency) to be more than total number of insertions of that elements. However, if we do allow such scenarios, we will refer it as *general turnstile model*.

2 Finding Frequent Items Deterministically

Here we describe an algorithm by Mishra and Gries [1] to find frequent items in the cash-register model that is when only insertions are allowed. The precise problem is as follows.

Given a sequence of m elements from $[1, n]$ and any $k \in \mathbb{N}$, find all the elements with frequency more than $\frac{m}{k}$ where frequency simply implies the number of times an element occurs using space $O(k)$ (in words) and using only a single pass. Hence in bits, the total space usage is $O(k \log n)$.

We would like to have the following guarantees.

- No false negative. All items that have frequency $> \frac{m}{k}$ will be reported.

There might be false positives, that is elements with frequency lower than m/k may be reported. However, if we allow two passes, in the second pass all the elements reported in the first pass can be checked for actual frequency and any false positive can be eliminated.

Description of the Algorithm

- **Data structure:** An associative array of $k-1$ elements initialized to empty. An associative array contains in each of its cell a key (the item) and a value (its count) and may be maintained in a balanced binary search tree by its key. Whenever we see an element $a_i = (j, 1)$ we can search in the array if j exists or not in $O(\log k)$ time.
- **Procedure:** Given arrival of $a_i = (j, 1)$, we search if j exists in the associative array. If the element is already there, we increment its count. If it is not there and there is an empty cell, we store it and make its count 1. If no space is left and the element is not there, we do not store the element and decrement the counters of all the stored elements by 1. If a counter reaches 0, that element is dropped from the associative array and the cell becomes empty. In the following code, we use A to represent the associative array, and give the update process when a_i is seen in the stream.

Algorithm 1 [Process $a_i = (j, 1)$]

```
Search for j ∈ A
if j is already present as key at cell A[l] then
  Count[l] = Count[l] + 1 {i}ncrement its count
else if j is not present in A and ∃l such that A[l] is empty then
  Insert j as key to A[l]
  Count[l] = 1
else
  Drop j {comment: j is not present and there is no free space}
  for i = 1 to k − 1 do
    Count[i] = Count[i] − 1
    if Count[i] == 0 then
      Drop the key associated with A[i] and mark A[i] empty
    end if
  end for
end if
```

The entire algorithm for a given $k \in \mathbb{N}$ is as follows

2.1 Analysis

We now prove the following theorem.

Theorem 1. *For any given $k \in \mathbb{N}$ Algorithm 2 returns all items with frequency more than $\frac{m}{k}$.*

Algorithm 2 Mishra-Gries Algorithm

Initialize A of size $k-1$ to empty
for $i = 1$ to m **do**
  Process(a_i)
end for
return All the elements in A

Proof. Clearly, the number of items having frequency more than $\frac{m}{k}$ is at most $k-1$ because stream size is m. Let f_j denote the actual frequency of item j and let $\hat{f}_j$ be the frequency of the item j as observed in A at the end of the processing. If j is not in A, we assume $\hat{f}_j = 0$. We therefore return all elements with $\hat{f}_j > 0$.

Note that we increment frequency for j only if we see the actual item. Hence $\hat{f}_j \leq f_j$. We want to find out how low it can be. If we never drop the element then $\hat{f}_j = f_j$. Otherwise, either at some point when j occurs, array A is full and j is not already there. Or because it is dropped from array due to its count being decremented to 0.

Note that whenever an item is dropped (due to no space or count getting decremented to 0), there are other distinct $k-1$ elements whose count also gets decremented. Here we view the count of an element on arrival is increased to 1, but it is dropped to 0 if it cannot be stored. Therefore, there can be at most $\frac{m}{k}$ steps on which element counts are decremented, k distinct elements at one shot. The reasoning being the stream size is m and all frequencies are non-negative.

For each of these events, the difference between the computed frequency and the actual frequency can increase by 1 and hence altogether we have $\hat{f}_j \geq f_j - \frac{m}{k}$ for all $j \in [1, n]$.

Therefore, for all $j \in [1, n]$ if $f_j > \frac{m}{k}$ then $\hat{f}_j > 0$ and hence it is stored in the array at the end and will be reported. □

In most data streaming algorithms, one cannot achieve any non-trivial approximation deterministically. In the following section, we come back to counting distinct items and show no deterministic algorithm is possible that in $o(m)$ space can give an exact count for distinct elements.

3 Lower Bound for Deterministic Computation of Distinct Elements

We prove the following theorem here.

Theorem 2. *There exists no deterministic algorithm that returns the exact count for distinct items in stream of size m in $o(m)$ bits.*

Proof. We will prove this by the by contradiction. Suppose it is possible to have an exact estimate of distinct elements using space $o(m)$ bits. Let R be such an algorithm. Since R uses $o(m)$ bits, the number of different possible configurations that R can maintain is at-most $2^{o(m)}$. We let $n = m$ and consider all the different streams which have exactly $\frac{m}{2}$ distinct elements. How many such

streams are possible ? Clearly, the number of such streams is at least

$$\binom{m}{m/2} \approx \left(\frac{me}{m/2}\right)^{\frac{m}{2}} = (2e)^{\frac{me}{2}} = 2^{\Theta(m)}$$

where the second inequality comes from Stirling's approximation.

Since the number of such streams is more than the number of available configurations of R, there must exist two streams y, y', $y \neq y'$ such that both have the same configuration, that is, $R(y) = R(y')$ and the distinct items in y are not identical to distinct items of y'. We now consider two streams $Y_1 = y + y$ and $Y_2 = y' + y$ where $+$ represents concatenation here, that is stream Y_1 y is followed by y and in stream Y_2, y' is followed by y. Since $R(y) = R(y')$, we must have $R(y + y) = R(y + y')$. Therefore R will return same distinct elements counts for both Y_1 and Y_2 which is wrong because the number of distinct elements in Y_1 is $\frac{m}{2}$ where for Y_2 it is $> \frac{m}{2}$. □

The above proof can be extended to show that there does not exist any deterministic algorithm with space $o(m)$ that returns a count of distinct items within a multiplicative factor less than 2. Similarly, one can show that no exact randomized algorithm can exist either in $o(m)$ space.

Surprisingly, when we allow both approximation and randomization, the space usage can be drastically reduced (next lecture).

4 Basic Concentration Inequalities

Here we study three basic concentration inequalities which bound deviation from expectation.

1. Markov inequality (The 1^{st} moment inequality)
2. Chebyshev inequality(The 2^{nd} moment inequality)
3. The Chernoff Bound

Theorem 3 (Markov Bound)**.** *For any positive random variable X, and for any $t > 0$*

$$\Pr(X \geq t) \leq \frac{E[x]}{t} \tag{1}$$

Proof.

$$\begin{aligned} E[x] &= \sum_x x \cdot \Pr(X = x) \\ &= \sum_{x<t} x \cdot \Pr(X = x) + \sum_{x \geq t} x \cdot \Pr(X = x) \\ &\geq 0 + t \cdot \sum_{x \geq t} \Pr(X = x) \\ &= t \cdot P(X \geq t) \end{aligned}$$

□

Theorem 4 (Chebyshev Inequality)**.** *For any random variable X and for any $t > 0$*

$$\Pr(|X - E[x]| \geq t) \leq \frac{Var(x)}{t^2} \tag{2}$$

Proof.

$$\begin{aligned} & \Pr(|X - E[x]| \geq t) \\ = \; & Pr([X - E[x]]^2 \geq t^2) \\ \leq \; & \frac{E\left[(X - E[x])^2\right]}{t^2} \\ = \; & \frac{Var(X)}{t^2} \end{aligned}$$

□

Theorem 5 (The Chernoff Bound)**.** *Let $X_1, X_2 ... X_n$ be n independent Bernoulli random variables with $\Pr(X_i = 1) = p_i$. Let $X = \sum X_i$. Hence,*

$$E[X] = E\left[\sum X_i\right] = \sum E\left[X_i\right] = \sum \Pr(X_i = 1) = \sum p_i = \mu \textit{(say)}.$$

Then the Chernoff Bound says for any $\epsilon > 0$

$$\begin{aligned} \Pr(X > (1+\epsilon)\mu) & \leq \left(\frac{e^{\epsilon}}{(1+\epsilon)^{\epsilon}}\right)^{\mu} \textit{ and} \\ \Pr(X < (1-\epsilon)\mu) & \leq \left(\frac{e^{-\epsilon}}{(1-\epsilon)^{1-\epsilon}}\right)^{\mu} \end{aligned}$$

When $0 < \epsilon < 1$ the above expression can be further simplified to

$$\begin{aligned} \Pr(X > (1+\epsilon)\mu) & \leq e^{\frac{-\mu\epsilon^2}{3}} \textit{ and} \\ \Pr(X < (1-\epsilon)\mu) & \leq e^{\frac{-\mu\epsilon^2}{2}} \end{aligned}$$

Hence

$$\Pr(|X - \mu| > \epsilon\mu) \leq 2e^{\frac{-\mu\epsilon^2}{3}}$$

References

[1] Jayadev Misra and David Gries. Finding repeated elements. Sci. Comput. Program., 2(2):143152, 1982.