15:853: Algorithms in the Real World
Fall 2019

Lecture 20

November 12 2019

Lecturer: Rashmi Vinayak *Scribe: Chrisma Pakha*

20.1 Recap From Last Class

In the previous lecture we introduced Bloom Filter, a space efficient data structure for approximate membership.

20.2 Overview

In this lecture, we want to calculate the probability of false positive for a Bloom Filter, what factor affects it. We will then introduce Data Streaming Model.

20.3 Bloom Filter

20.3.1 Calculating the probability of false positive in a Bloom Filter

Assume we have a Bloom Filter with the following stucture and parameters:

- An array T with length M
- k, which is the number of hash functions
- The items that we want to hash belongs to a set, S, where $|S| = N$

First the probability that a particular bit has not been set by any N items using the k hash function:

$$(1 - \frac{1}{M})^{kN}$$

We can transform this equation to

$$(1 - \frac{1}{M})^{\frac{MkN}{M}}$$

Now, using the equation:

$$\lim_{x \to \infty} (1 - \frac{1}{x})^x = e$$

The probability that a particular bit have not been set by any N items using the k hash function is:

$$e^{\frac{-kN}{M}}$$

Note that in order for this to hold, M should be a large value. Hence the probability of false positive becomes:

$$(1 - e^{\frac{-kN}{M}})^k$$

We have the power of k since all the position needs to be set to 1.

20.3.2 How to minimize probability of false positive

$$(1 - e^{\frac{-kN}{M}})^k \tag{20.1}$$

Here, we want to know what we can do to minimize the false positive. Based on (20.1), one of the knobs we can tweak is the number of hash functions, k. To find the desired k, we can set the derivative of (20.1) to zero.
Since we know that (20.1) is monotonic, we can simplify the equation into the following equation

$$\frac{d}{dk} \ln (1 - e^{\frac{-kN}{M}})^k = 0 \tag{20.2}$$

(20.2) becomes

$$\ln (1 - e^{-\frac{kN}{M}}) + \frac{k}{1 - e^{-\frac{kN}{M}}} \frac{N}{M} e^{-\frac{kN}{M}} = 0$$

The solution for this equation is

$$k = \frac{M}{N} \ln 2 \tag{20.3}$$

(20.3) is the value of k the minimize the probability of false positive.
Say ϵ denotes the probability of false positive, using k from (20.3),

$$\epsilon = \frac{1}{2}^{\frac{M}{N} \ln 2}$$

We can also write the equation as follows :

$$M = \ln (2) N log(\frac{1}{\epsilon})$$

$$M = 1.44 N log(\frac{1}{\epsilon})$$

M here determines the storage space. Hence the number of bits used per element is

$$\frac{M}{N} = 1.44 log \frac{1}{\epsilon}$$

20.4 Data Streaming Model

The real world problem Data streaming Model is trying to solve is the following :
Given a stream $a_1, a_2, ..., a_i$, where each elements belong to the Universe set, where each element takes a certain bit, and given the limited storage space, what is an efficient way to aggregate statistics based on the element seen so far.
Example of Statistics or Function of interest

- Sum of elements (easy to compute, keep adding the number)
- Maximum of elements(easy to compute, keep the maximum number and keep comparing items seens so far)
- Median (tricky)
- Heavy Hitters (most popular element)
- Distinct elements

20.4.1 An approach using sampling

A natural choice would be to reduce amount of data by sampling. However done incorrectly, we could get the wrong results

Example

Say we know that in a stream there are n elements. Here we want to know the number of unique elements. Say the number of elements that occur once is $n/2$. The remaining $n/4$ elements occurs twice. Say we use 10% sampling, each element in the stream is independently sampled with $p = 0.1$. After sampling, the expected length is $n/10$. Number of unique elements we expect to see is

$$0.1\frac{n}{2} + \frac{n}{4}(2 * 0.1 - 0.1^2) = \frac{n}{10}$$

Approximately, by scaling, the number of unique elements is $10 * \frac{n}{10}$ which is n. Here we are a factor of 2 from the real answer which is not good for this case.

The problem is that each element is sampled independently. What should be done is that if an element is sampled, then if it has another occurence, its other copy should also be sampled.

How would hashing help? Say we have universal hash function $h : U \to [M]$, where $M = 10$.

$$p(h(a) = \alpha) \leq \frac{1}{M} = \frac{1}{10}$$

$\frac{1}{10}$ of the elements map to one of the 0-9 bucket. It helps, since an element and its copy are placed into the same bucket.

20.4.2 Useful abstraction in stream processing

Streams can be viewed as a vector in a high dimensional space. A stream at time t can be denoted as

$$x^t = [x_1^t, x_2^t, ..., x_u^t]$$

x_i^t indicates the number of times the i^th element has appeared by time t.
This leads to an extended model

- New element (add e)
- An existing element departing (del e)

Here, we make an assumption that at any point in time, number of deletion e $\leq$ number of addition e. This ensures that x_i^t are non-negative.
Example of using this abstraction

- To count number of total elements using this abstraction : Denote $||x||$ as the number of total elements

$$||x|| = \sum_{i=1}^{|U|} x_i^t$$

- To count the number of distinct elements : number of non-zero entries

20.4.3 ϵ-Heavy-Hitters

Under this model we can formulate *Heavy-hitters* : ***ϵ-Heavy-Hitter***
ϵ-Heavy-Hitter is all indices, i, such that $x_i \geq \epsilon||x||$

20.4.4 Count Query

To solve ***ϵ-Heavy-Hitter***, we first need to understand **Count Query**
Count Query : At any time t, given an index i, output the value of x_i^t with an error of at most $\epsilon||x||$

Solution for count query using sampling

Would sampling work for count query? An example that would not work is as follows : Say, we have a stream of element as follows

$$S = addA, addA, addA, ..., addA, delA, delA, delA, ..., delA, addB, addB, ..., addB$$

The number of adding A elements is N, the number of deleting A elements is N, and the number of adding B element is $\sqrt{N}$. At the end of the stream of our example, we know that B should be the heavy hitters. In order to see B, our sample probability should be greater than $\sqrt{N}$. Anything less, we do not expect to see B.

Solution for count query using hashing

Solution for count query using hash is **Count-min Sketch**. Say we have a Universal hash function :

$$h : u \to [M]$$

and an Array, A, of non-negative integer.

$$A[0...M-1]$$

When a_i arrives, we use a hash function to give us an index. The update rule for A are as

- If $a_i =$ **add i**, then $A[h(i)] + +$
- If $a_i =$ **delete i**, then $A[h(i)] - -$

The goal is to estimate x_i^t based on our data structure. Say, our estimate of x_i^t is computed as follows

$$y_i = A[h(i)]$$

The question we want to ask is how good is our estimate.

$$A[h(i)] = \sum_{j \in U} x_j^t I(h(j) = h(i)) \tag{20.4}$$

where, I is an indicator function. We can partition the sum in (20.4) into the following

$$A[h(i)] = x_i^t + \sum_{j \in U, j \neq i} x_j^t I(h(j) = h(i)) \tag{20.5}$$

In (20.5), the second term is the error. The expected error is

$$E[error] = E[\sum_{j \in U, j \neq i} x_j^t I(h(j) = h(i))] \tag{20.6}$$

Here the randomness comes from the indicator function. (20.7) becomes

$$E[error] = \sum_{j \in U, j \neq i} x_j^t E[I(h(j) = h(i))] \tag{20.7}$$

$$E[error] = \sum_{j \in U, j \neq i} x_j^t p(I(h(j) = h(i))) \tag{20.8}$$

Since we assume our hash function is universal, $p(h(i) = h(j)) = \frac{1}{M}$
(20.8) becomes

$$E[error] = \frac{1}{M} \sum_{j \in U, j \neq i} x_j^t \tag{20.9}$$

$$E[error] = \frac{1}{M} \sum_{j \in U, j \neq i} x_j^t \tag{20.10}$$

$$E[error] = \frac{||x_j|| - x_i}{M} \tag{20.11}$$

$$E[error] \leq \frac{||x||}{M} \tag{20.12}$$

The error in expectation is $\epsilon||x||$, where $\epsilon = \frac{1}{M}$