COMP 480: Introduction to Stream Computing and Reservoir Sampling

Louie Lu (yl231), Allen Dong (ad102)

September 2023

1 Data Streams

Data streams center around the idea that users generate vast amounts of data at a rapid rate. They are essentially massive, continuously growing sets of data, often seen in scenarios like Google search queries or trending Twitter topics. Three key properties define data streams. Firstly, the complete dataset is not known in advance, meaning we can't predict the nature of incoming data. Secondly, data enters sequentially, creating a time series. Lastly, due to its immense size, it's impossible to store the entire data stream. For instance, consider tracking Google search queries for flu symptoms, which aids in efficient flu virus tracking. Given the sheer velocity and unpredictability in the size of this data, one might often think of it as being infinite. Traditional storage solutions fall short as we usually only have 10-20 GB for calculations or inferences. This poses a challenging question: "How can we make crucial computations about these data streams utilizing a restricted memory capacity?" Analyzing these streams has significant real-world applications, from monitoring online query trends and unusual user behaviors to overseeing IP packets, telephone records, and sensor networks.

2 One-pass model

In the realm of data streams, we consider the stream $D_n = \{x_1, x_2, x_3, \ldots, x_n\}$ where x_i represents elements arriving over time and observed at a specific instance t. As we progress in time, our observation becomes $D_t = x_1, x_2, x_3, \ldots, x_t$. Given that we can't predict the entire data stream, and at any point in time t, our memory allocation is constrained and less than t, storing all of D_t becomes infeasible. This leads to the central challenge: designing an algorithm that can compute $f(D_t)$ at any time t, where f is a particular function of interest. A common approach to this issue is to selectively sample representative elements from the stream, using these samples to approximate the necessary computations. But this method introduces another query: how can we ensure a uniform sampling of elements from the stream? Consolidating these principles into the "one pass model," our objective becomes to compute (or closely estimate) $f(D_t)$ for any given x_t and prior state from time $t-1$, acknowledging the storage constraints. This framework then aids in addressing foundational and pragmatic concerns related to inferring insights from data streams.

3 Sampling

If we can get a representative sample of the data stream, then we can do analysis on it. The question then becomes how do we obtain that sample?

3.1 Sampling (example 1)

When handling data streams, say we've encountered a set $\{x_1, \ldots, x_{1000}\}$, where the size of the data is known in advance. However, our memory can only accommodate 100 of these examples, prompting us to sample roughly 10

3.2 Sampling (example 2)

Consider a data set comprising U unique elements and $2D$ elements appearing as duplicate pairs, resulting in a total of $N = U + 2D$ elements. Our objective is to deduce the fraction of duplicated elements, which mathematically is represented as $\frac{2D}{U+2D}$. When attempting to estimate this ratio through a 10

4 Reservoir Sampling

When faced with a complex sampling problem from a data stream, the goal is to select s elements by the time we reach the nth element. There are two essential conditions to meet during this process:

1. Every element, from the first up to the nth, should have an equal chance of being selected, which translates to a $\frac{s}{n}$ probability.
2. By the end of the sampling, the total number of chosen elements should be precisely s.

Algorithm:
We sample elements with the following protocol:

1. Observe x_n
2. if $n < s$: keep x_n
3. else: with probability $\frac{s}{n}$, select x_n. Then choose (uniformly) one of the previously sampled elements and replace it with x_n.

We claim that at any time n, any element in the sequence $\{x_1, x_2, \ldots, x_n\}$ has exactly a $\frac{s}{n}$ chance of being in the sample.

Proof by Induction:

After observing t elements, each element in the reservoir was sampled with probability $\frac{s}{t}$.

For the rest of the elements in the reservoir, the probability that it still stays in the reservoir is:

$$P(x \text{ survive}) = P(x_t \text{ rejected}) + P(x_t \text{ accept}) P(x \text{ not selected}) = \left(1 - \frac{s}{t}\right) + \left(\frac{s}{t}\right)\left(\frac{s-1}{s}\right) = \frac{t-1}{t}$$

Then, the probability that x survives in the sample at time t is:

$$P(x \text{ in } S \text{ at time } t) = P(x \text{ in } S \text{ at time } t-1) \cdot P(x \text{ survive}) = \frac{s}{t-1} \cdot \frac{t-1}{t} = \frac{s}{t}.$$

QED.

5 Weighted Reservoir sampling:

5.1 Introduction:

Weighted Reservoir sampling is a generalized version of Reservoir sampling where each element x_i in the stream $\{x_1, x_2, \cdots, x_j, \cdots\}$ is associated with a positive weight w_i. But we are still sampling m elements from the first n elements at the time $t = n$. Here are some notations we will use:

1. $D_n = \{x_1, x_2, \cdots, x_n\}$, which is the stream at a given time.
2. S represents the set of elements we choose to preserve from the stream.
3. S_n represents the indexes of elements in S.
4. $s(\cdot)$, which represents the score of a particular element in the stream. (We will use it later)

Suppose our S is empty at the beginning, we want to make sure that for each x_i in the stream with weight w_i, x_i has a probability of being selected:

$$Pr(x_i) = \frac{w_i}{\sum_{j \in [n] - S_n} w_j}$$

This simply means we sum the weights of elements that are not in S and the probability of selecting a particular x_i equals the proportion of w_i over the total weights.

5.2 Pavlos S Efraimidis and Paul G Spirakis:

These guys come up with a smart way of implementing this. Here is how they achieve it.

Algorithm 1: WRS-A

$S \leftarrow \emptyset$;
$P \leftarrow \emptyset$;
for $t \leftarrow 1, 2, ...$ **do**
    $r_t \leftarrow U[0,1]$, uniform sample of [0,1];
    $X_t \leftarrow r_t^{\frac{1}{w_t}}$, the score of x_t;
    $P \leftarrow P \cup \{(X_t, x_t)\}$;
    $S \leftarrow top\_m(P)$, w.r.t. the first entry of P which is the scores.

We can use minheap as the data structure, which will give us constant time to retrieve the minimal element from $D_n - S$. What we will do next is to prove the algorithm that it is correct.

5.3 Proof:

The idea of the proof is that for any permutation of $S = \{x_1, x_2, ..., x_m\}$, the indexes do not represent the time sequence but merely for indexing. If we follow the method in 1.1 and sample S, we want to calculate its probability and compare it to the probability that it comes from the $WRS - A$. Since S is arbitrarily notated, we can show the implementation is precise in implementing WRS. Let's find $Pr_D(x_1, x_2, ..., x_m)$, which refers to the theoretical method in part 1.1.

$$Pr_D(x_1, x_2, ..., x_m) = \frac{W_n}{\sum_{j=1}^{n} W_j} \times \frac{W_{n-1}}{\sum_{j=1}^{n-1} W_j} \times \cdots \times \frac{W_1}{W_1} \tag{1}$$

$$= \prod_{i=1}^{n} \frac{W_i}{\sum_{j=1}^{i} W_j} \tag{2}$$

Now, let's look at the probability of getting this permutation in the second method:

$$Pr(X_n \geq X_{n-1} \geq \cdots X_1) = \int_0^1 f_{X_n}(x) pr(x > X_{n-1} \geq X_{n-2} \geq \cdots X_1) dx \tag{3}$$

Notice: $\Pr(x > X_1) = s(X_1)^{w_1}$, where $s(X_1)$ is the score of X_1. We can apply this to the end of the integral and eventually we will get something equals to the above expression, which will show this is a correct implementation.