COMP 480/580 — Probabilistic Algorithms and Data Structure Feb 6, 2020

Lecture 8: Introduction to Stream Computing and Reservoir Sampling

Lecturer: Li Chou *Scribe By: Xin Yang*

1 Data Steam

Nowadays users generate massive data on the internet. These data, which is increasing over time, is called data stream. There are some important properties about the data stream that distinguish it from the other data. First, we do not know the entire dataset in advance, at any time step, what the following data looks like is invisible for us. Second, the data elements enter one by one, forming a time series. Third, the data stream is typically very large so that we can't store the entire data stream.

These properties prevent the traditional deterministic algorithms to be applied and finding out the objectives or characteristics of the data stream is therefore difficult. However, analyzing data stream has many applications. The data stream techniques are used to track the online query tendency, unusual user behavior, social networks news feeds, sensor networks, telephone call records, IP packets monitoring, etc. Therefore, what we concern is the methods how we can make critical calculations about the data stream using limited amount of memory.

2 Formulating the Problem

We formulate the data stream as $D_n = \{x_1, x_2, x_3, \cdots, x_n\}$, where x_i are the elements entering over time and being observed at time t. We do not know the entire data stream in advance, so at any time t, what we observe is $D_t = \{x_1, x_2, x_3, \cdots, x_t\}$.

At any time t, we have limited memory budget less than t. Therefore, we can't store the entire data in the memory.

The problem we are concerning is to find a algorithm that computes $f(D_t)$ at any time t.

A typical solution is to sample representative elements from the data stream and use the sample to estimate the calculations we need. This solution, in term, raises the problem: how to uniformly sample elements from the data stream. In this lecture, we discuss the problem of sampling.

3 Sampling from the Data Stream

3.1 Naive Solution

Let's consider a simplified problem: how can we sample 1/10 of the data uniformly from the original data stream. We can have several straightforward and similar solutions to problem, one of them can be describe in Algorithm 1.

However, the algorithm raises 2 issues. First, we do not bound the number of elements selected from the data stream, so the sample size can be bigger or smaller than the 1/10 of the data stream size. Also it may go unbounded and out of memory. Second, in some applications, this may cause estimation bias.

Algorithm 1: Naively Sampling 1/10 data from data stream.

Input: $D_t \leftarrow$ data stream at time t.
Result: $s \leftarrow$ uniformly selected from data stream D.

```
i ← 0 ;
s ← ∅ ;
while i ≤ t do
    r ← random interger in range of [1, 10] inclusive ;
    i ← i + r ;
    s ← s ∪ D_i ;
end
return s
```

3.2 Sampling a Fraction

To see the second issue, consider we have a data stream with U unique elements and D elements each has one duplicate. If we want to estimate the fraction of duplicated elements $\frac{2D}{U+2D}$ and using Algorithm 1 to sample 1/10 of U and 1/10 of $2D$, the total sample size is $\frac{1}{10}(U+2D)$. However, the probability that the 2 duplicated elements are both in the sample is $\frac{1}{10}\frac{1}{10} = 1/100$. So the sampling will recognize $(2D)/100$ duplicate elements instead of $(2D)/10$. This method will underestimate the duplicate fraction by 10 times.

3.3 Reservoir Sampling

Here we discuss a more complex sampling problem. If we want to sample s elements from data stream at any time t, we want our samples satisfies the following two conditions:

1. Every elements from 1 to t has same probability to be sampled, i.e. $\frac{s}{t}$.
2. The number of elements is exactly s.

The above naive method can't solve this problem as the number of samples is not bounded. So we use Reservoir Sampling shown in Algorithm 2 to solve the problem.

We can easily see that the algorithm satisfies the second requirement of the problem. So we should only prove that it can also uniformly select elements from data stream D_t. We prove this by induction. First we observe that the first s time steps of the algorithm satisfies the requirement. Second, we observe that the probability of x_t to be selected in the sample is $\frac{s}{t}$. For the rest of other elements in the reservoir, the probability that it still stay in the reservoir is:

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

So the probability that it is sampled from data stream D_t is

Algorithm 2: Reservoir Sampling

Input: $x_t \leftarrow$ new observation at time t; $S_{t-1} \leftarrow$ samples at time $t-1$; $s \leftarrow$ sample size
Result: $S \leftarrow$ uniform samples from D_t of time t.

```
if |S_{t-1}| < s then
    S_t ← S_{t-1} ∪ {x_t} ;
else
    if with probability s/t then
        x ← uniformly selected from S_{t-1} ;
        S_t ← S_{t-1} − {x} ∪ {x_t} ;
    else
        S_t ← S_{t-1};
    end
end
return S_t
```

Algorithm 3: Weighted Sampling by Pavlos Efraimidis and Paul Spirakis

Input: $\langle x_t, w_t \rangle \leftarrow$ new observation at time t; $S_{t-1} \leftarrow$ global memory storing s elements and their scores at time $t-1$
Result: $S \leftarrow$ uniform samples from D_t of time t.

```
r_t ← Uniform(0, 1) ;
e_t ← r_t^{1/w_t} be the score of element x_t ;
S_t ← top s scored elements in S_{t-1} ∪ {⟨x_t, e_t⟩} ;
return S_t
```

$$\begin{aligned} P(x \in S_t) &= P(x \in S_{t-1}) P(x \text{ survive}) \\ &= \frac{s}{t-1} \frac{t-1}{t} \\ &= \frac{s}{t} \end{aligned}$$

Therefore, we prove that the reservoir is uniformly selected from the data stream.

3.4 Weighted Sampling

We want to general the above method to solve a more sophisticate problem: weighted sampling. Besides the requirements we mentioned above, we assign each element x_i a weight w_i to indicate its importance. We should at each time t samples elements according to their weight, i.e., $P(x_i \text{ sampled}) = \frac{w_i}{\sum_{j=1} t w_j}$.

The naive solution can be letting w_i be integers equal or larger than 1. In this case we replicate each element w_i times, and use Reservoir Sampling to select from it.

However, this solution works for only integer weights. We proposed a more elegent solution, Algorithm 3 to solve the problem.

Note that at each timestamp, we returns exactly s elements as the sample of the data stream. If we keep the top s records in heap, the insertion complexity should be $O(\log s)$.

Another issue is that if we restrain the weights be in the range of $(0, 1)$, the scores will go closer to 1 and therefore indistinguishable when using float point number. This problem can be solved by applying log to the scores.

The correctness of this algorithm is difficult, we just see one of its lemmas to have a glance of its correctness.

Lemma 1. *Let r_1 and r_2 be independent random variables following $Uniform(0, 1)$. Denote random variables $X_i = r_i^{\frac{1}{w_i}}$ for $i = 1, 2$, where $w_i \geq 0$. We have $P(X_1 \leq X_2) = \frac{w_2}{w_1+w_2}$.*

Proof.

$$\begin{aligned}
P(X_1 \leq X_2) &= P(r_1^{\frac{1}{w_1}} \leq r_2^{\frac{1}{w_2}}) \\
&= P(r_1 \leq r_2^{\frac{w_1}{w_2}}) \\
&= \int_{r_2=0}^{1} \int_{r_1=0}^{r_2^{\frac{w_1}{w_2}}} dr_1 dr_2 \\
&= \int_{r_2=0}^{1} r_2^{\frac{w_1}{w_2}} dr_2 \\
&= \frac{w2}{w_1 + w_2}
\end{aligned}$$

□