Lecture Notes: Data Stream

Miao Qiao
Computer Science Department
University of Auckland
miao.qiao@auckland.ac.nz

You may have heard about streaming data which is prevalent in our daily life: credit card transactions stream to the banking system, internet IP packets stream through routers and phone calls stream across switches. Streaming data features a high velocity while keeping real-time statistical information over the streaming data becomes challenging.

Definition 1 (Data Stream)**.** *A data stream is an unbounded sequence of data elements where the i-th element, denoted as e_i, arrives the system at time t_i.*

In the simplest form of a data stream, $t_i = i$. We denote the set of elements that arrive the system by time n as $D_n = \{e_1, e_2, \cdots, e_n\}$. Given an integer k, we call a subset S of D_n of size $|S| = k$ a k-sized subset of D_n. A *without-replacement sample set* R of D_n is a k-sized subset of D_n such that for any k-sized subset S of D_n, $Pr[R = S] = \frac{1}{\binom{n}{k}}$.

1 Full Stream Sampling

Problem 1 (Full Stream Sampling)**.** *Design a structure/algorithm to keep a without-replacement sample set S_n on a data stream with arrived data element set D_n for any time $n > 0$.*

The structure/algorithm will be evaluated in two metrics:

- The space occupied by the structure, and
- The time consumed for updating the structure.

1.1 Reservoir Sampling

Algorithm[2]. Keep an initially empty sample set R. For each new arrival e_i at time i:

- If $i \leq k$, add e_i to R;
- If $i > k$, toss a coin with head probability $\frac{k}{i}$ and if it turns out to be head, then replace a random element in R with e_i — each element in R will be replaced with a probability $\frac{1}{k}$.

Theorem 1. *At any time $n \geq k$, the sample set kept by the reservoir sampling algorithm, denoted as R_n, is a without-replacement sample set of D_n.*

Proof.

- Base case: When $n = k$, $R_k = D_k$ so $Pr[R_k = D_k] = 1 = \frac{1}{\binom{n}{k}}$.
- Inductive step: Let l be an integer $> k$. If for time $n = l - 1$, for any k-sized subset S of D_n, $Pr[R_n = S] = \frac{1}{\binom{l-1}{k}}$, then when $n = l$, for each k-sized subset S' of D_n,
  - If $e_l \notin S'$, then $Pr[R_n = S'] = Pr[R_{l-1} = S'] \times \frac{l-k}{l} = \frac{1}{\binom{l}{k}}$.
  - If $e_l \in S'$, then $Pr[R_n = S'] = \Sigma_{e' \in D_{l-1} \setminus S'} Pr[R_{l-1} = S' \cup \{e'\} \setminus \{e_l\}] \times \frac{k}{l} \times \frac{1}{k} = \frac{1}{\binom{l}{k}}$.

□

Analysis. Space complexity $O(k)$. Time complexity $O(k \log \frac{n}{k})$ for processing n elements.

Remarks. Consider, how to efficiently maintain a large sample whose size k exceeds the memory capacity M on a data stream?

1.2 Priority Sampling

Algorithm. Assign a randomly and uniformly selected value $p_i \in (0, 1]$ as the priority of e_i for each element e_i when it arrives the system. At any time n, keep the sample set R_n as the k elements with the highest priorities.

Theorem 2. *At any time $n \geq k$, the sample set kept by the priority sampling algorithm, denoted as R_n, is a without-replacement sample set of D_n.*

Proof. Denote by $R = \{r_1, r_2, \cdots, r_k\}$ the k elements in D_n with the highest priorities in *descending* order of their priorities. Let $E = \{e_{i_1}, e_{i_2}, e_{i_3}, \cdots e_{i_k}\}$ be k distinct elements in R_n. The probability that $Pr[e_{i_1} = r_1] = \frac{1}{n}$ — the independent and uniform sampling guarantees that all possible outcome (of r_1) is equally likely. Similarly, we have

- $Pr[e_{i_2} = r_2 | e_{i_1} = r_1] = \frac{1}{n-1}$,
- $Pr[e_{i_3} = r_3 | e_{i_1} = r_1 \text{ and } e_{i_2} = r_2] = \frac{1}{n-2}$,

  $\vdots$
- $Pr[e_{i_k} = r_k | e_{i_1} = r_1 \text{ and } e_{i_2} = r_2 \text{ and } \cdots \text{ and } e_{i_{k-1}} = r_{k-1}] = \frac{1}{n-k+1}$.

By multiplying them together, we have $Pr[e_{i_1} = r_1 \text{ and } e_{i_2} = r_2 \text{ and } \cdots \text{ and } e_{i_k} = r_k] = \frac{(n-k)!}{n!}$. Since any permutation of $r_1, r_2, \cdots, r_k$ will lead to set $R = E$, the probability that $R = E$ is $Pr[R = E] = \frac{k!(n-k)!}{n!} = \frac{1}{\binom{k}{n}}$. □

Implementation. Theoretically speaking, the probability that two elements share the same priority is zero since the priority of an element is uniformly sampled at random from an infinite number of real values in range $(0, 1]$. However, modern computers use limited number of binary digits to represent a floating point arithmetic, this means that when n is extremely large, the difference between the k-th highest priority and the $(k + 1)$-th highest priority may not be recognizable which is not good since the priority sampling is correct only if there is no tie among the top-k priorities. Therefore, we design the random assignment of priority in the following way.

The priority of each element will be represented as a sequence of h binary digits where h is a global variable which is initially 1. For each newly arrived element, we assign a random binary sequence of length h as its priority — each digit in the sequence is either 0 or 1 with equal probability. For example, if $h = 2$, the sequence can equally likely be $00, 01, 10$ or 11.

- If there is a tie between the priority of e_i with any of the top k highest priorities, we increase h by 1 and append the sequence of each element's priority with a random bit in $\{0, 1\}$, and we repeated this process until all ties are eliminated;
- Update the top-k prioritized elements based on e_i's priority.

Analysis. Space complexity $O(k)$. Time complexity $O(k \log \frac{n}{k})$ for processing n elements.

2 Sequence-Based Sliding Window Sampling

In this section we study the sampling on two kinds of sliding windows, fixed-sized sliding window and variable-sized sliding window, the latter problem is a simplification of the sampling on time-based sliding window [1].

2.1 Fixed-Sized Sliding Window

Definition 2. *Given an integer n, on a data stream of $e_1, e_2, \cdots, e_t, \cdots$, the sliding window W_t at time t includes the most recent n elements that arrived the system: $W_t = \{e_{t-n+1}, e_{t-n+2}, \cdots, e_{t-1}, e_t\}$.*

Keeping a sample set of size k over a fixed-sized sliding window can be extremely easy:

1. Keep a sample set R of size k on the first n elements using reservoir sampling;
2. For any time $t > n$, if there is an element in R expires at time t, that is, if $e_{t-n} \in R$, replace e_{t-n} with e_t.

This algorithm is obviously correct, however, since after time $t = n$, samples are taken periodically which becomes predictable. Such a predictability fails certain analytically tasks which require independence among the sample sets over disjoint windows. Therefore, we introduce another sampling algorithm called two-bucket sampling.

Equal-sized two-bucket sampling. Conceptually partition the stream into disjoint buckets while each bucket contains n elements. For example, $U_1 = \{e_1, e_2, \cdots, e_n\}$ is the first bucket while $U_2 = \{e_{n+1}, e_{n+2}, \cdots, e_{2n}\}$ is the second bucket. Since the buckets are partitioned conceptually, a bucket may contain elements that are yet arrived.

At time t, window W_t contains the most recent n elements that have arrived the system.

- An element e_i is alive if $e_i \in W_t$, that is, $i \in [t - n + 1, t]$;
- An element e_i is expired if $i \leq t - n$.

A bucket U is alive if it contains at least an alive element, that is, $U \cap W_t \neq \emptyset$; and is expired if all its elements are expired.

For the i-th bucket $U_i = \{e_{(i-1)n+1}, e_{(i-1)n+2}, \cdots, e_{in}\}$, we start from time $t = (i-1)n + 1$ to keep a k-sized sample set R_i for U_i using reservoir sampling and freeze the sample set of U_i after all elements of U have been in place, that is, time $t > i \times n$.

At any time $t > n$ there are at most two active buckets U_i and U_{i+1} where $i = \lfloor \frac{t}{n} \rfloor$. At time t, we denote

- U_i^e: the expired elements in U_i, that is, $U_i^e = U_i \setminus W_t$, and
- U_i^a: the active elements in U_i, that is, $U_i^a = U_i \cap W_t$.

Denote by R_i the k-sized sample set of U_i, R_{i+1} the k-sized sample set of the arrived elements in U_{i+1} — the reservoir sampling of U_{i+1} is still going on at time t. We generate the sample set of W_t as follows:

- Count the expired samples in R_i, that is, $k_e = |R_i \cap U_i^e|$;
- Get k_e random samples from R_{i+1} and denote the sample set as R';
- Report $R = R' \cup R_i^a$ as the sample set of W_t.

Theorem 3. *For any k-sized subset S of W_t, the sample set R reported by the equal-sized two-bucket sampling has $Pr[S = R] = \frac{1}{\binom{n}{k}}$.*

Proof. We leave the proof to the readers who have interest (see Section 2.2 [1]). □

2.2 Variable-Sized Sliding Window

The variable-sized sliding window is similar to but easier than time-based sliding window.

Problem 2. *On a data stream of $e_1, e_2, \cdots, e_t, \cdots$, the window W_t at time t can include a number of n_t recent elements, that is, $W_t = \{e_{t-n_t+1}, e_{t-n_t+2}, \cdots, e_t\}$. The constraint is that $n_{t+1} \leq n_t + 1$ for any time $t > 0$. Maintain a k-sized sample set for W_t for any time t.*

2.2.1 Sample Union

We first consider how to generate samples of the union of two sets based on the random samples of the two sets.

Problem 3. *Consider two sets, U_1 and U_2. Denote $l_1 = |U_1|$ and $l_2 = |U_2|$. Element r_1 is sampled uniformly at random from U_1 and r_2 is sampled uniformly at random from U_2. Generate a sample r for the union $U_1 \cup U_2$ such that for any $s \in U_1 \cup U_2$, $Pr[r = s] = \frac{1}{l_1+l_2}$.*

Two-step sampling. To report a random sample on $U_1 \cup U_2$,

- Step 1: toss a coin with head probability $p = \frac{l_1}{l_1+l_2}$;
- Step 2: if it heads, report r_1 — an element sampled from U_1 uniformly at random; if it tails, report r_2 — an element from U_2 uniformly at random.

Lemma 1. *For any element $s \in U_1 \cup U_2$, the probability that $Pr[r = s] = \frac{1}{l_1+l_2}$.*

Proof. If $s \in U_1$, $Pr[r = s] = p \times Pr[r_1 = s] = \frac{1}{l_1+l_2}$; if $s \in U_2$, $Pr[r = s] = (1-p) \times Pr[r_2 = s] = \frac{1}{l_1+l_2}$. □

2.2.2 Two bucket sampling

To solve Problem 2, we create the following structure. Let $l = \lceil \log_2 \frac{t}{k} \rceil$. Consider $l + 1$ levels, level 0 to level l. For the i-th level, $i \in [0, l]$, we conceptually partition the stream into disjoint $2^i k$-sized buckets. For example, level L_0 has the j-th bucket $U_{0,j} = \{e_{(j-1)k+1}, e_{(j-1)k+2}, \cdots, e_{(j-1)k+k}\}$ while level L_1 has the j-th bucket $U_{1,j} = \{e_{2(j-1)k+1}, e_{2(j-1)k+2}, \cdots, e_{2(j-1)k+2k}\}$. Note that $U_{1,j}$ is the union of $U_{0,2j-1}$ and $U_{0,2j}$ for any $j > 0$.

Similar to Section 2.1, we compute a k-sized sample set for each bucket in each level using a reservoir sampling. For each level, we need to keep the sample set for the 4 most recent buckets. In other words, for level i, at time t, the newest bucket is the g-th bucket $U_{i,g}$ where $g = \lceil \frac{t}{2^i k} \rceil$. We only need to keep the sample set $R_{i,g}$, $R_{i,g-1}$, $R_{i,g-2}$, and $R_{i,g-3}$. Note that when $U_{i,g}$ has less than k elements arrived, $R_{i,g} = U_{i,g}$ under reservoir sampling.

At time t, when the size n_t of window W_t is given, we find a proper level — the level such that the expiration line $t - n_t$ cuts the third or fourth latest bucket. In other words, we need to find the i-th level with the newest bucket id $g = \lceil \frac{t}{2^i k} \rceil$ such that the expiring element e_{t-n_t} falls in $U_{i,g-3} \cup U_{i,g-2}$.

Denote by U the expiring bucket at level i that element e_{t-n_t} belongs to; denote its sample set R_U. Get a sample set R_V of the union V of all the buckets newer than U at level i using the two-step sampling. Denote by $l_1 = |U|$ and $l_2 = |V|$.

- Flip a coin k times with head probability $\frac{l_1}{n_t}$; denote by h the total number of heads.
- Select h samples uniformly at random from the sample set R_U, denote by R_a the alive samples reported, and denote by $h' = h - |R_a|$.
- Select $h' + k - h$ samples uniformly at random from R_V, denote the set of reported samples R'_V.
- Report $R = R'_V \cup R_a$ as the sample set of W_t.

References

[1] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *PODS*, pages 147–156, 2009.

[2] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.